
HPX Documentation

master

The STE||AR Group

May 25, 2022

USER DOCUMENTATION

1	What is <i>HPX</i>?	3
2	What's so special about <i>HPX</i>?	5
2.1	Quick start	5
2.2	Examples	9
2.3	Manual	34
2.4	Terminology	213
2.5	Why <i>HPX</i> ?	214
2.6	Additional material	220
2.7	Overview	220
2.8	API reference	246
2.9	Contributing to <i>HPX</i>	934
2.10	Releases	942
2.11	Citing <i>HPX</i>	1201
2.12	<i>HPX</i> users	1201
2.13	About <i>HPX</i>	1201
3	Index	1211

If you're new to *HPX* you can get started with the [Quick start](#) guide. Don't forget to read the [Terminology](#) section to learn about the most important concepts in *HPX*. The [Examples](#) give you a feel for how it is to write real *HPX* applications and the [Manual](#) contains detailed information about everything from building *HPX* to debugging it. There are links to blog posts and videos about *HPX* in [Additional material](#).

If you can't find what you're looking for in the documentation, please:

- open an issue on [GitHub](#)¹;
- contact us on [IRC](#), the *HPX* channel on the [C++ Slack](#)², or on our [mailing list](#)³; or
- read or ask questions tagged with *HPX* on [StackOverflow](#)⁴.

You can find a comprehensive list of contact options on [Support for deploying and using HPX](#)⁵.

See [Citing HPX](#) for details on how to cite *HPX* in publications. See [HPX users](#) for a list of institutions and projects using *HPX*.

¹ <https://github.com/STELLAR-GROUP/hpx/issues>

² <https://cpplang.slack.com>

³ hpx-users@stellar.cct.lsu.edu

⁴ <https://stackoverflow.com/questions/tagged/hpx>

⁵ <https://github.com/STELLAR-GROUP/hpx/blob/master/.github/SUPPORT.md>

**CHAPTER
ONE**

WHAT IS HPX?

HPX is a C++ Standard Library for Concurrency and Parallelism. It implements all of the corresponding facilities as defined by the C++ Standard. Additionally, in *HPX* we implement functionalities proposed as part of the ongoing C++ standardization process. We also extend the C++ Standard APIs to the distributed case. *HPX* is developed by the STE||AR group (see [People](#)).

The goal of *HPX* is to create a high quality, freely available, open source implementation of a new programming model for conventional systems, such as classic Linux based Beowulf clusters or multi-socket highly parallel SMP nodes. At the same time, we want to have a very modular and well designed runtime system architecture which would allow us to port our implementation onto new computer system architectures. We want to use real-world applications to drive the development of the runtime system, coining out required functionalities and converging onto a stable API which will provide a smooth migration path for developers.

The API exposed by *HPX* is not only modeled after the interfaces defined by the C++11/14/17/20 ISO standard. It also adheres to the programming guidelines used by the Boost collection of C++ libraries. We aim to improve the scalability of today's applications and to expose new levels of parallelism which are necessary to take advantage of the exascale systems of the future.

WHAT'S SO SPECIAL ABOUT *HPX*?

- HPX exposes a uniform, standards-oriented API for ease of programming parallel and distributed applications.
- It enables programmers to write fully asynchronous code using hundreds of millions of threads.
- HPX provides unified syntax and semantics for local and remote operations.
- HPX makes concurrency manageable with dataflow and future based synchronization.
- It implements a rich set of runtime services supporting a broad range of use cases.
- HPX exposes a uniform, flexible, and extendable performance counter framework which can enable runtime adaptivity
- It is designed to solve problems conventionally considered to be scaling-impaired.
- HPX has been designed and developed for systems of any scale, from hand-held devices to very large scale systems.
- It is the first fully functional implementation of the ParalleX execution model.
- HPX is published under a liberal open-source license and has an open, active, and thriving developer community.

2.1 Quick start

The following steps will help you get started with *HPX*.

2.1.1 Installing *HPX*

The easiest way to install *HPX* on your system is by choosing one of the steps below:

1. **vcpkg**

You can download and install *HPX* using the `vcpkg`⁶ dependency manager:

```
$ vcpkg install hpx
```

2. **Spack**

Another way to install *HPX* is using `Spack`⁷:

```
$ spack install hpx
```

⁶ <https://github.com/Microsoft/vcpkg>

⁷ <https://spack.readthedocs.io/en/latest/>

3. Fedora

Installation can be done with Fedora⁸ as well:

```
$ dnf install hpx*
```

4. Arch Linux

HPX is available in the [Arch User Repository \(AUR\)](#)⁹ as hpx too.

More information or alternatives regarding the installation can be found in the `hpx_build_system`, a detailed guide with thorough explanation of ways to build and use HPX.

2.1.2 Hello, World!

To get started with this minimal example you need to create a new project directory and a file `CMakeLists.txt` with the contents below in order to build an executable using CMake and HPX:

```
cmake_minimum_required(VERSION 3.18)
project(my_hpx_project CXX)
find_package(HPX REQUIRED)
add_executable(my_hpx_program main.cpp)
target_link_libraries(my_hpx_program HPX::hpx HPX::wrap_main HPX::iostreams_component)
```

The next step is to create a `main.cpp` with the contents below:

```
// Including 'hpx/hpx_main.hpp' instead of the usual 'hpx/hpx_init.hpp' enables
// to use the plain C-main below as the direct main HPX entry point.
#include <hpx/hpx_main.hpp>
#include <hpx/iostream.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << std::flush;
    return 0;
}
```

Then, in your project directory run the following:

```
$ mkdir build && cd build
$ cmake -DCMAKE_PREFIX_PATH=/path/to/hpx/installation ..
$ make all
$ ./my_hpx_program
```

```
$ ./my_hpx_program
Hello World!
```

The program looks almost like a regular C++ hello world with the exception of the two includes and `hpx::cout`.

- When you include `hpx_main.hpp` HPX makes sure that `main` actually gets launched on the HPX runtime. So while it looks almost the same you can now use futures, `async`, parallel algorithms and more which make use of the HPX runtime with lightweight threads.
- `hpx::cout` is a replacement for `std::cout` to make sure printing never blocks a lightweight thread. You can read more about `hpx::cout` in [The HPX I/O-streams component](#).

⁸ <https://fedoraproject.org/wiki/DNF>

⁹ https://wiki.archlinux.org/title/Arch_User_Repository

Note:

- You will most likely have more than one `main.cpp` file in your project. See the section on [Using HPX with CMake-based projects](#) for more details on how to use `add_hpx_executable`.
 - `HPX::wrap_main` is required if you are implicitly using `main()` as the runtime entry point. See [Re-use the main\(\) function as the main HPX entry point](#) for more information.
 - `HPX::iostreams_component` is optional for a minimal project but lets us use the *HPX* equivalent of `std::cout`, i.e., the [HPX The HPX I/O-streams component](#) functionality in our application.
 - You do not have to let *HPX* take over your main function like in the example. See [Starting the HPX runtime](#) for more details on how to initialize and run the *HPX* runtime.
-

Caution: When including `hpx_main.hpp` the user-defined `main` gets renamed and the real `main` function is defined by *HPX*. This means that the user-defined `main` must include a return statement, unlike the real `main`. If you do not include the return statement, you may end up with confusing compile time errors mentioning `user_main` or even runtime errors.

2.1.3 Writing task-based applications

So far we haven't done anything that can't be done using the C++ standard library. In this section we will give a short overview of what you can do with *HPX* on a single node. The essence is to avoid global synchronization and break up your application into small, composable tasks whose dependencies control the flow of your application. Remember, however, that *HPX* allows you to write distributed applications similarly to how you would write applications for a single node (see [Why HPX?](#) and [Writing distributed HPX applications](#)).

If you are already familiar with `async` and `futures` from the C++ standard library, the same functionality is available in *HPX*.

The following terminology is essential when talking about task-based C++ programs:

- **lightweight thread:** Essential for good performance with task-based programs. Lightweight refers to smaller stacks and faster context switching compared to OS threads. Smaller overheads allow the program to be broken up into smaller tasks, which in turns helps the runtime fully utilize all processing units.
- **async:** The most basic way of launching tasks asynchronously. Returns a `future<T>`.
- **future<T>:** Represents a value of type `T` that will be ready in the future. The value can be retrieved with `get` (blocking) and one can check if the value is ready with `is_ready` (non-blocking).
- **shared_future<T>:** Same as `future<T>` but can be copied (similar to `std::unique_ptr` vs `std::shared_ptr`).
- **continuation:** A function that is to be run after a previous task has run (represented by a `future`). `then` is a method of `future<T>` that takes a function to run next. Used to build up dataflow DAGs (directed acyclic graphs). `shared_futures` help you split up nodes in the DAG and functions like `when_all` help you join nodes in the DAG.

The following example is a collection of the most commonly used functionality in *HPX*:

```
#include <hpx/local/algorithm.hpp>
#include <hpx/local/future.hpp>
#include <hpx/local/init.hpp>
```

(continues on next page)

(continued from previous page)

```
#include <iostream>
#include <random>
#include <vector>

void final_task(hpx::future<hpx::tuple<hpx::future<double>, hpx::future<void>>>)
{
    std::cout << "in final_task" << std::endl;
}

int hpx_main()
{
    // A function can be launched asynchronously. The program will not block
    // here until the result is available.
    hpx::future<int> f = hpx::async([]() { return 42; });
    std::cout << "Just launched a task!" << std::endl;

    // Use get to retrieve the value from the future. This will block this task
    // until the future is ready, but the HPX runtime will schedule other tasks
    // if there are tasks available.
    std::cout << "f contains " << f.get() << std::endl;

    // Let's launch another task.
    hpx::future<double> g = hpx::async([]() { return 3.14; });

    // Tasks can be chained using the then method. The continuation takes the
    // future as an argument.
    hpx::future<double> result = g.then([](hpx::future<double>&& gg) {
        // This function will be called once g is ready. gg is g moved
        // into the continuation.
        return gg.get() * 42.0 * 42.0;
    });

    // You can check if a future is ready with the is_ready method.
    std::cout << "Result is ready? " << result.is_ready() << std::endl;

    // You can launch other work in the meantime. Let's sort a vector.
    std::vector<int> v(1000000);

    // We fill the vector synchronously and sequentially.
    hpx::generate(hpx::execution::seq, std::begin(v), std::end(v), &std::rand);

    // We can launch the sort in parallel and asynchronously.
    hpx::future<void> done_sorting =
        hpx::sort(hpx::execution::par,           // In parallel.
                  hpx::execution::task,      // Asynchronously.
                  std::begin(v), std::end(v));

    // We launch the final task when the vector has been sorted and result is
    // ready using when_all.
    auto all = hpx::when_all(result, done_sorting).then(&final_task);

    // We can wait for all to be ready.
    all.wait();

    // all must be ready at this point because we waited for it to be ready.
    std::cout << (all.is_ready() ? "all is ready!" : "all is not ready...")
        << std::endl;
}
```

(continues on next page)

(continued from previous page)

```

    return hpx::local::finalize();
}

int main(int argc, char* argv[])
{
    return hpx::local::init(hpx_main, argc, argv);
}

```

Try copying the contents to your `main.cpp` file and look at the output. It can be a good idea to go through the program step by step with a debugger. You can also try changing the types or adding new arguments to functions to make sure you can get the types to match. The type of the `then` method can be especially tricky to get right (the continuation needs to take the future as an argument).

Note: *HPX* programs accept command line arguments. The most important one is `--hpx:threads=N` to set the number of OS threads used by *HPX*. *HPX* uses one thread per core by default. Play around with the example above and see what difference the number of threads makes on the `sort` function. See [Launching and configuring HPX applications](#) for more details on how and what options you can pass to *HPX*.

Tip: The example above used the construction `hpx::when_all(...).then(...)`. For convenience and performance it is a good idea to replace uses of `hpx::when_all(...).then(...)` with dataflow. See [Dataflow](#) for more details on dataflow.

Tip: If possible, try to use the provided parallel algorithms instead of writing your own implementation. This can save you time and the resulting program is often faster.

2.1.4 Next steps

If you haven't done so already, reading the [Terminology](#) section will help you get familiar with the terms used in *HPX*.

The [Examples](#) section contains small, self-contained walkthroughs of example *HPX* programs. The [Local to remote](#) example is a thorough, realistic example starting from a single node implementation and going stepwise to a distributed implementation.

The [Manual](#) contains detailed information on writing, building and running *HPX* applications.

2.2 Examples

The following sections analyze some examples to help you get familiar with the *HPX* style of programming. We start off with simple examples that utilize basic *HPX* elements and then begin to expose the reader to the more complex and powerful *HPX* concepts.

2.2.1 Asynchronous execution

The Fibonacci sequence is a sequence of numbers starting with 0 and 1 where every subsequent number is the sum of the previous two numbers. In this example, we will use *HPX* to calculate the value of the n-th element of the Fibonacci sequence. In order to compute this problem in parallel, we will use a facility known as a future.

As shown in the Fig. 2.1 below, a future encapsulates a delayed computation. It acts as a proxy for a result initially not known, most of the time because the computation of the result has not completed yet. The future synchronizes the access of this value by optionally suspending any *HPX*-threads requesting the result until the value is available. When a future is created, it spawns a new *HPX*-thread (either remotely with a *parcel* or locally by placing it into the thread queue) which, when run, will execute the function associated with the future. The arguments of the function are bound when the future is created.

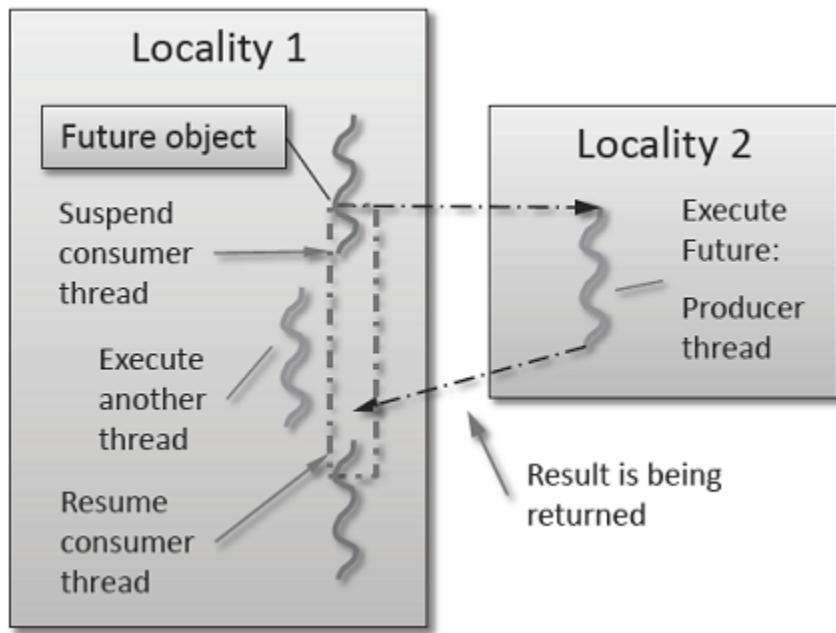


Fig. 2.1: Schematic of a future execution.

Once the function has finished executing, a write operation is performed on the future. The write operation marks the future as completed, and optionally stores data returned by the function. When the result of the delayed computation is needed, a read operation is performed on the future. If the future's function hasn't completed when a read operation is performed on it, the reader *HPX*-thread is suspended until the future is ready. The future facility allows *HPX* to schedule work early in a program so that when the function value is needed it will already be calculated and available. We use this property in our Fibonacci example below to enable its parallel execution.

Setup

The source code for this example can be found here: `fibonacci_local.cpp`.

To compile this program, go to your *HPX* build directory (see `hpx_build_system` for information on configuring and building *HPX*) and enter:

```
$ make examples.quickstart.fibonacci_local
```

To run the program type:

```
$ ./bin/fibonacci_local
```

This should print (time should be approximate):

```
fibonacci(10) == 55
elapsed time: 0.002430 [s]
```

This run used the default settings, which calculate the tenth element of the Fibonacci sequence. To declare which Fibonacci value you want to calculate, use the `--n-value` option. Additionally you can use the `--hpx:threads` option to declare how many OS-threads you wish to use when running the program. For instance, running:

```
$ ./bin/fibonacci --n-value 20 --hpx:threads 4
```

Will yield:

```
fibonacci(20) == 6765
elapsed time: 0.062854 [s]
```

Walkthrough

Now that you have compiled and run the code, let's look at how the code works. Since this code is written in C++, we will begin with the `main()` function. Here you can see that in *HPX*, `main()` is only used to initialize the runtime system. It is important to note that application-specific command line options are defined here. *HPX* uses [Boost.Program Options](#)¹⁰ for command line processing. You can see that our programs `--n-value` option is set by calling the `add_options()` method on an instance of `hpx::program_options::options_description`. The default value of the variable is set to 10. This is why when we ran the program for the first time without using the `--n-value` option the program returned the 10th value of the Fibonacci sequence. The constructor argument of the description is the text that appears when a user uses the `--hpx:help` option to see what command line options are available. `HPX_APPLICATION_STRING` is a macro that expands to a string constant containing the name of the *HPX* application currently being compiled.

In *HPX* `main()` is used to initialize the runtime system and pass the command line arguments to the program. If you wish to add command line options to your program you would add them here using the instance of the Boost class `options_description`, and invoking the public member function `.add_options()` (see [Boost Documentation](#)¹¹ for more details). `hpx::init` calls `hpx_main()` after setting up *HPX*, which is where the logic of our program is encoded.

```
int main(int argc, char* argv[])
{
    // Configure application-specific options
    hpx::program_options::options_description desc_commandline(
        "Usage: " HPX_APPLICATION_STRING " [options]");

    desc_commandline.add_options()("n-value",
        hpx::program_options::value<std::uint64_t>()>default_value(10),
        "n value for the Fibonacci function");

    // Initialize and run HPX
    hpx::local::init_params init_args;
    init_args.desc_cmdline = desc_commandline;

    return hpx::local::init(hpx_main, argc, argv, init_args);
}
```

¹⁰ https://www.boost.org/doc/html/program_options.html

¹¹ <https://www.boost.org/doc/>

The `hpx::init` function in `main()` starts the runtime system, and invokes `hpx_main()` as the first *HPX*-thread. Below we can see that the basic program is simple. The command line option `--n-value` is read in, a timer (`hpx::chrono::high_resolution_timer`) is set up to record the time it takes to do the computation, the `fibonacci` function is invoked synchronously, and the answer is printed out.

```
int hpx_main(hpx::program_options::variables_map& vm)
{
    // extract command line argument, i.e. fib(N)
    std::uint64_t n = vm["n-value"].as<std::uint64_t>();

    {
        // Keep track of the time required to execute.
        hpx::chrono::high_resolution_timer t;

        std::uint64_t r = fibonacci(n);

        char const* fmt = "fibonacci({1}) == {2}\nelapsed time: {3} [s]\n";
        hpx::util::format_to(std::cout, fmt, n, r, t.elapsed());
    }

    return hpx::local::finalize();      // Handles HPX shutdown
}
```

The `fibonacci` function itself is synchronous as the work done inside is asynchronous. To understand what is happening we have to look inside the `fibonacci` function:

```
std::uint64_t fibonacci(std::uint64_t n)
{
    if (n < 2)
        return n;

    // Invoking the Fibonacci algorithm twice is inefficient.
    // However, we intentionally demonstrate it this way to create some
    // heavy workload.

    hpx::future<std::uint64_t> n1 = hpx::async(fibonacci, n - 1);
    hpx::future<std::uint64_t> n2 = hpx::async(fibonacci, n - 2);

    return n1.get() +
           n2.get();      // wait for the Futures to return their values
}
```

This block of code looks similar to regular C++ code. First, `if (n < 2)`, meaning `n` is 0 or 1, then we return 0 or 1 (recall the first element of the Fibonacci sequence is 0 and the second is 1). If `n` is larger than 1 we spawn two new tasks whose results are contained in `n1` and `n2`. This is done using `hpx::async` which takes as arguments a function (function pointer, object or lambda) and the arguments to the function. Instead of returning a `std::uint64_t` like `fibonacci` does, `hpx::async` returns a future of a `std::uint64_t`, i.e. `hpx::future<std::uint64_t>`. Each of these futures represents an asynchronous, recursive call to `fibonacci`. After we've created the futures, we wait for both of them to finish computing, we add them together, and return that value as our result. We get the values from the futures using the `get` method. The recursive call tree will continue until `n` is equal to 0 or 1, at which point the value can be returned because it is implicitly known. When this termination condition is reached, the futures can then be added up, producing the `n`-th value of the Fibonacci sequence.

Note that calling `get` potentially blocks the calling *HPX*-thread, and lets other *HPX*-threads run in the meantime. There are, however, more efficient ways of doing this. `examples/quickstart/fibonacci_futures.cpp` contains many more variations of locally computing the Fibonacci numbers, where each method makes different tradeoffs in where asynchrony and parallelism is applied. To get started, however, the method above is sufficient and

optimizations can be applied once you are more familiar with *HPX*. The example [Dataflow](#) presents dataflow, which is a way to more efficiently chain together multiple tasks.

2.2.2 Parallel algorithms

This program will perform a matrix multiplication in parallel. The output will look something like this:

```
Matrix A is :
4 9 6
1 9 8

Matrix B is :
4 9
6 1
9 8

Resultant Matrix is :
124 93
111 127
```

Setup

The source code for this example can be found here: `matrix_multiplication.cpp`.

To compile this program, go to your *HPX* build directory (see `hpx_build_system` for information on configuring and building *HPX*) and enter:

```
$ make examples.quickstart.matrix_multiplication
```

To run the program type:

```
$ ./bin/matrix_multiplication
```

or:

```
$ ./bin/matrix_multiplication --n 2 --m 3 --k 2 --s 100 --l 0 --u 10
```

where the first matrix is $n \times m$ and the second $m \times k$, s is the seed for creating the random values of the matrices and the range of these values is $[l,u]$

This should print:

```
Matrix A is :
4 9 6
1 9 8

Matrix B is :
4 9
6 1
9 8

Resultant Matrix is :
124 93
111 127
```

Notice that the numbers may be different because of the random initialization of the matrices.

Walkthrough

Now that you have compiled and run the code, let's look at how the code works.

First, `main()` is used to initialize the runtime system and pass the command line arguments to the program. `hpx::init` calls `hpx_main()` after setting up HPX, which is where our program is implemented.

```
int main(int argc, char* argv[])
{
    using namespace hpx::program_options;
    options_description cmdline("usage: " HPX_APPLICATION_STRING " [options]");
    // clang-format off
    cmdline.add_options()
        ("n",
         hpx::program_options::value<std::size_t>()>default_value(2),
         "Number of rows of first matrix")
        ("m",
         hpx::program_options::value<std::size_t>()>default_value(3),
         "Number of columns of first matrix (equal to the number of rows of "
         "second matrix)")
        ("k",
         hpx::program_options::value<std::size_t>()>default_value(2),
         "Number of columns of second matrix")
        ("seed,s",
         hpx::program_options::value<unsigned int>(),
         "The random number generator seed to use for this run")
        ("l",
         hpx::program_options::value<std::size_t>()>default_value(0),
         "Lower limit of range of values")
        ("u",
         hpx::program_options::value<std::size_t>()>default_value(10),
         "Upper limit of range of values");
    // clang-format on
    hpx::local::init_params init_args;
    init_args.desc_cmdline = cmdline;

    return hpx::local::init(hpx_main, argc, argv, init_args);
}
```

Proceeding to the `hpx_main()` function, we can see that matrix multiplication can be done very easily.

```
int hpx_main(hpx::program_options::variables_map& vm)
{
    using element_type = int;

    // Define matrix sizes
    std::size_t rowsA = vm["n"].as<std::size_t>();
    std::size_t colsA = vm["m"].as<std::size_t>();
    std::size_t rowsB = colsA;
    std::size_t colsB = vm["k"].as<std::size_t>();
    std::size_t rowsR = rowsA;
    std::size_t colsR = colsB;

    // Initialize matrices A and B
    std::vector<int> A(rowsA * colsA);
    std::vector<int> B(rowsB * colsB);
    std::vector<int> R(rowsR * colsR);
```

(continues on next page)

(continued from previous page)

```

// Define seed
unsigned int seed = std::random_device{}();
if (vm.count("seed"))
    seed = vm["seed"].as<unsigned int>();

gen.seed(seed);
std::cout << "using seed: " << seed << std::endl;

// Define range of values
std::size_t lower = vm["l"].as<std::size_t>();
std::size_t upper = vm["u"].as<std::size_t>();

// Matrices have random values in the range [lower, upper]
std::uniform_int_distribution<element_type> dis(lower, upper);
auto generator = std::bind(dis, gen);
hpx::ranges::generate(A, generator);
hpx::ranges::generate(B, generator);

// Perform matrix multiplication
hpx::experimental::for_loop(hpx::execution::par, 0, rowsA, [&](auto i) {
    hpx::experimental::for_loop(0, colsB, [&](auto j) {
        R[i * colsR + j] = 0;
        hpx::experimental::for_loop(0, rowsB, [&](auto k) {
            R[i * colsR + j] += A[i * colsA + k] * B[k * colsB + j];
        });
    });
});

// Print all 3 matrices
print_matrix(A, rowsA, colsA, "A");
print_matrix(B, rowsB, colsB, "B");
print_matrix(R, rowsR, colsR, "R");

return hpx::local::finalize();
}

```

First, the dimensions of the matrices are defined. If they were not given as command-line arguments, their default values are 2×3 for the first matrix and 3×2 for the second. We use standard vectors to define the matrices to be multiplied as well as the resultant matrix.

To give some random initial values to our matrices, we use `std::uniform_int_distribution`¹². Then, `std::bind()` is used along with `hpx::ranges::generate()` to yield two matrices A and B, which contain values in the range of $[0, 10]$ or in the range defined by the user at the command-line arguments. The seed to generate the values can also be defined by the user.

The next step is to perform the matrix multiplication in parallel. This can be done by just using an `hpx::experimental::for_loop` combined with a parallel execution policy `hpx::execution::par` as the outer loop of the multiplication. Note that the execution of `hpx::experimental::for_loop` without specifying an execution policy is equivalent to specifying `hpx::execution::seq` as the execution policy.

Finally, the matrices A, B that are multiplied as well as the resultant matrix R are printed using the following function.

```

void print_matrix(
    std::vector<int> M, std::size_t rows, std::size_t cols, const char* message)
{

```

(continues on next page)

¹² https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution

(continued from previous page)

```
std::cout << "\nMatrix " << message << " is:" << std::endl;
for (std::size_t i = 0; i < rows; i++)
{
    for (std::size_t j = 0; j < cols; j++)
        std::cout << M[i * cols + j] << " ";
    std::cout << "\n";
}
}
```

2.2.3 Asynchronous execution with actions

This example extends the [previous example](#) by introducing *actions*: functions that can be run remotely. In this example, however, we will still only run the action locally. The mechanism to execute *actions* stays the same: `hpx::async`. Later examples will demonstrate running actions on remote *localities* (e.g. [Remote execution with actions](#)).

Setup

The source code for this example can be found here: `fibonacci.cpp`.

To compile this program, go to your *HPX* build directory (see `hpx_build_system` for information on configuring and building *HPX*) and enter:

```
$ make examples.quickstart.fibonacci
```

To run the program type:

```
$ ./bin/fibonacci
```

This should print (time should be approximate):

```
fibonacci(10) == 55
elapsed time: 0.00186288 [s]
```

This run used the default settings, which calculate the tenth element of the Fibonacci sequence. To declare which Fibonacci value you want to calculate, use the `--n-value` option. Additionally you can use the `--hpx:threads` option to declare how many OS-threads you wish to use when running the program. For instance, running:

```
$ ./bin/fibonacci --n-value 20 --hpx:threads 4
```

Will yield:

```
fibonacci(20) == 6765
elapsed time: 0.233827 [s]
```

Walkthrough

The code needed to initialize the *HPX* runtime is the same as in the *previous example*:

```
int main(int argc, char* argv[])
{
    // Configure application-specific options
    hpx::program_options::options_description desc_commandline(
        "Usage: " HPX_APPLICATION_STRING " [options]");

    desc_commandline.add_options()("n-value",
        hpx::program_options::value<std::uint64_t>() -> default_value(10),
        "n value for the Fibonacci function");

    // Initialize and run HPX
    hpx::init_params init_args;
    init_args.desc_cmdline = desc_commandline;

    return hpx::init(argc, argv, init_args);
}
```

The `hpx::init` function in `main()` starts the runtime system, and invokes `hpx_main()` as the first *HPX*-thread. The command line option `--n-value` is read in, a timer (`hpx::chrono::high_resolution_timer`) is set up to record the time it takes to do the computation, the `fibonacci` *action* is invoked synchronously, and the answer is printed out.

```
int hpx_main(hpx::program_options::variables_map& vm)
{
    // extract command line argument, i.e. fib(N)
    std::uint64_t n = vm["n-value"].as<std::uint64_t>();

    {
        // Keep track of the time required to execute.
        hpx::chrono::high_resolution_timer t;

        // Wait for fib() to return the value
        fibonacci_action fib;
        std::uint64_t r = fib(hpx::find_here(), n);

        char const* fmt = "fibonacci({1}) == {2}\nelapsed time: {3} [s]\n";
        hpx::util::format_to(std::cout, fmt, n, r, t.elapsed());
    }

    return hpx::finalize();      // Handles HPX shutdown
}
```

Upon a closer look we see that we've created a `std::uint64_t` to store the result of invoking our `fibonacci_action` `fib`. This *action* will launch synchronously (as the work done inside of the *action* will be asynchronous itself) and return the result of the Fibonacci sequence. But wait, what is an *action*? And what is this `fibonacci_action`? For starters, an *action* is a wrapper for a function. By wrapping functions, *HPX* can send packets of work to different processing units. These vehicles allow users to calculate work now, later, or on certain nodes. The first argument to our *action* is the location where the *action* should be run. In this case, we just want to run the *action* on the machine that we are currently on, so we use `hpx::find_here`. To further understand this we turn to the code to find where `fibonacci_action` was defined:

```
// forward declaration of the Fibonacci function
std::uint64_t fibonacci(std::uint64_t n);
```

(continues on next page)

(continued from previous page)

```
// This is to generate the required boilerplate we need for the remote
// invocation to work.
HPX_PLAIN_ACTION(fibonacci, fibonacci_action)
```

A plain *action* is the most basic form of *action*. Plain *actions* wrap simple global functions which are not associated with any particular object (we will discuss other types of *actions* in [Components and actions](#)). In this block of code the function `fibonacci()` is declared. After the declaration, the function is wrapped in an *action* in the declaration `HDX_PLAIN_ACTION`. This function takes two arguments: the name of the function that is to be wrapped and the name of the *action* that you are creating.

This picture should now start making sense. The function `fibonacci()` is wrapped in an *action* `fibonacci_action`, which was run synchronously but created asynchronous work, then returns a `std::uint64_t` representing the result of the function `fibonacci()`. Now, let's look at the function `fibonacci()`:

```
std::uint64_t fibonacci(std::uint64_t n)
{
    if (n < 2)
        return n;

    // We restrict ourselves to execute the Fibonacci function locally.
    hpx::id_type const locality_id = hpx::find_here();

    // Invoking the Fibonacci algorithm twice is inefficient.
    // However, we intentionally demonstrate it this way to create some
    // heavy workload.

    fibonacci_action fib;
    hpx::future<std::uint64_t> n1 = hpx::async(fib, locality_id, n - 1);
    hpx::future<std::uint64_t> n2 = hpx::async(fib, locality_id, n - 2);

    return n1.get() +
        n2.get();      // wait for the Futures to return their values
}
```

This block of code is much more straightforward and should look familiar from the [previous example](#). First, `if (n < 2)`, meaning `n` is 0 or 1, then we return 0 or 1 (recall the first element of the Fibonacci sequence is 0 and the second is 1). If `n` is larger than 1 we spawn two tasks using `hpx::async`. Each of these futures represents an asynchronous, recursive call to `fibonacci`. As previously we wait for both futures to finish computing, get the results, add them together, and return that value as our result. The recursive call tree will continue until `n` is equal to 0 or 1, at which point the value can be returned because it is implicitly known. When this termination condition is reached, the futures can then be added up, producing the `n`-th value of the Fibonacci sequence.

2.2.4 Remote execution with actions

This program will print out a hello world message on every OS-thread on every *locality*. The output will look something like this:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 1 on locality 1
hello world from OS-thread 0 on locality 0
hello world from OS-thread 0 on locality 1
```

Setup

The source code for this example can be found here: `hello_world_distributed.cpp`.

To compile this program, go to your *HPX* build directory (see `hpx_build_system` for information on configuring and building *HPX*) and enter:

```
$ make examples.quickstart.hello_world_distributed
```

To run the program type:

```
$ ./bin/hello_world_distributed
```

This should print:

```
hello world from OS-thread 0 on locality 0
```

To use more OS-threads use the command line option `--hpx:threads` and type the number of threads that you wish to use. For example, typing:

```
$ ./bin/hello_world_distributed --hpx:threads 2
```

will yield:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
```

Notice how the ordering of the two print statements will change with subsequent runs. To run this program on multiple localities please see the section [How to use HPX applications with PBS](#).

Walkthrough

Now that you have compiled and run the code, let's look at how the code works, beginning with `main()`:

```
// Here is the main entry point. By using the include 'hpx/hpx_main.hpp' HPX
// will invoke the plain old C-main() as its first HPX thread.
int main()
{
    // Get a list of all available localities.
    std::vector<hpx::id_type> localities = hpx::find_all_localities();

    // Reserve storage space for futures, one for each locality.
    std::vector<hpx::future<void>> futures;
    futures.reserve(localities.size());

    for (hpx::id_type const& node : localities)
    {
        // Asynchronously start a new task. The task is encapsulated in a
        // future, which we can query to determine if the task has
        // completed.
        typedef hello_world_foreman_action action_type;
        futures.push_back(hpx::async<action_type>(node));
    }

    // The non-callback version of hpx::wait_all takes a single parameter,
    // a vector of futures to wait on. hpx::wait_all only returns when
```

(continues on next page)

(continued from previous page)

```
// all of the futures have finished.
hpx::wait_all(futures);
return 0;
}
```

In this excerpt of the code we again see the use of futures. This time the futures are stored in a vector so that they can easily be accessed. `hpx::wait_all` is a family of functions that wait on for an `std::vector<>` of futures to become ready. In this piece of code, we are using the synchronous version of `hpx::wait_all`, which takes one argument (the `std::vector<>` of futures to wait on). This function will not return until all the futures in the vector have been executed.

In *Asynchronous execution with actions* we used `hpx::find_here` to specify the target of our actions. Here, we instead use `hpx::find_all_localities`, which returns an `std::vector<>` containing the identifiers of all the machines in the system, including the one that we are on.

As in *Asynchronous execution with actions* our futures are set using `hpx::async<>`. The `hello_world_foreman_action` is declared here:

```
// Define the boilerplate code necessary for the function 'hello_world_foreman'
// to be invoked as an HPX action.
HPX_PLAIN_ACTION(hello_world_foreman, hello_world_foreman_action)
```

Another way of thinking about this wrapping technique is as follows: functions (the work to be done) are wrapped in actions, and actions can be executed locally or remotely (e.g. on another machine participating in the computation).

Now it is time to look at the `hello_world_foreman()` function which was wrapped in the action above:

```
void hello_world_foreman()
{
    // Get the number of worker OS-threads in use by this locality.
    std::size_t const os_threads = hpx::get_os_thread_count();

    // Populate a set with the OS-thread numbers of all OS-threads on this
    // locality. When the hello world message has been printed on a particular
    // OS-thread, we will remove it from the set.
    std::set<std::size_t> attendance;
    for (std::size_t os_thread = 0; os_thread < os_threads; ++os_thread)
        attendance.insert(os_thread);

    // As long as there are still elements in the set, we must keep scheduling
    // HPX-threads. Because HPX features work-stealing task schedulers, we have
    // no way of enforcing which worker OS-thread will actually execute
    // each HPX-thread.
    while (!attendance.empty())
    {
        // Each iteration, we create a task for each element in the set of
        // OS-threads that have not said "Hello world". Each of these tasks
        // is encapsulated in a future.
        std::vector<hpx::future<std::size_t>> futures;
        futures.reserve(attendance.size());

        for (std::size_t worker : attendance)
        {
            // Asynchronously start a new task. The task is encapsulated in a
            // future, which we can query to determine if the task has
            // completed. We give the task a hint to run on a particular worker
            // thread, but no guarantees are given by the scheduler that the
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

// task will actually run on that worker thread.
hpx::execution::parallel_executor exec(
    hpx::threads::thread_schedule_hint(
        hpx::threads::thread_schedule_hint_mode::thread,
        static_cast<std::int16_t>(worker)));
futures.push_back(hpx::async(exec, hello_world_worker, worker));
}

// Wait for all of the futures to finish. The callback version of the
// hpx::wait_each function takes two arguments: a vector of futures,
// and a binary callback. The callback takes two arguments; the first
// is the index of the future in the vector, and the second is the
// return value of the future. hpx::wait_each doesn't return until
// all the futures in the vector have returned.
hpx::spinlock mtx;
hpx::wait_each(hpx::unwrapping([&](std::size_t t) {
    if (std::size_t(-1) != t)
    {
        std::lock_guard<hpx::spinlock> lk(mtx);
        attendance.erase(t);
    }
}), futures);
}
}
}

```

Now, before we discuss `hello_world_foreman()`, let's talk about the `hpx::wait_each` function. The version of `hpx::wait_each` invokes a callback function provided by the user, supplying the callback function with the result of the future.

In `hello_world_foreman()`, an `std::set` called `attendance` keeps track of which OS-threads have printed out the hello world message. When the OS-thread prints out the statement, the future is marked as ready, and `hpx::wait_each` in `hello_world_foreman()`. If it is not executing on the correct OS-thread, it returns a value of -1, which causes `hello_world_foreman()` to leave the OS-thread id in `attendance`.

```

std::size_t hello_world_worker(std::size_t desired)
{
    // Returns the OS-thread number of the worker that is running this
    // HPX-thread.
    std::size_t current = hpx::get_worker_thread_num();
    if (current == desired)
    {
        // The HPX-thread has been run on the desired OS-thread.
        char const* msg = "hello world from OS-thread {1} on locality {2}\n";

        hpx::util::format_to(hpx::cout, msg, desired, hpx::get_locality_id())
            << std::flush;

        return desired;
    }

    // This HPX-thread has been run by the wrong OS-thread, make the foreman
    // try again by rescheduling it.
    return std::size_t(-1);
}

```

Because HPX features work stealing task schedulers, there is no way to guarantee that an action will be scheduled on

a particular OS-thread. This is why we must use a guess-and-check approach.

2.2.5 Components and actions

The accumulator example demonstrates the use of components. Components are C++ classes that expose methods as a type of *HPX* action. These actions are called component actions.

Components are globally named, meaning that a component action can be called remotely (e.g., from another machine). There are two accumulator examples in *HPX*.

In the *Asynchronous execution with actions* and the *Remote execution with actions*, we introduced plain actions, which wrapped global functions. The target of a plain action is an identifier which refers to a particular machine involved in the computation. For plain actions, the target is the machine where the action will be executed.

Component actions, however, do not target machines. Instead, they target component instances. The instance may live on the machine that we've invoked the component action from, or it may live on another machine.

The component in this example exposes three different functions:

- `reset ()` - Resets the accumulator value to 0.
- `add (arg)` - Adds `arg` to the accumulators value.
- `query ()` - Queries the value of the accumulator.

This example creates an instance of the accumulator, and then allows the user to enter commands at a prompt, which subsequently invoke actions on the accumulator instance.

Setup

The source code for this example can be found here: `accumulator_client.cpp`.

To compile this program, go to your *HPX* build directory (see `hpx_build_system` for information on configuring and building *HPX*) and enter:

```
$ make examples.accumulators.accumulator
```

To run the program type:

```
$ ./bin/accumulator_client
```

Once the program starts running, it will print the following prompt and then wait for input. An example session is given below:

```
commands: reset, add [amount], query, help, quit
> add 5
> add 10
> query
15
> add 2
> query
17
> reset
> add 1
> query
1
> quit
```

Walkthrough

Now, let's take a look at the source code of the accumulator example. This example consists of two parts: an *HPX* component library (a library that exposes an *HPX* component) and a client application which uses the library. This walkthrough will cover the *HPX* component library. The code for the client application can be found here: `accumulator_client.cpp`.

An *HPX* component is represented by two C++ classes:

- **A server class** - The implementation of the component's functionality.
- **A client class** - A high-level interface that acts as a proxy for an instance of the component.

Typically, these two classes both have the same name, but the server class usually lives in different sub-namespaces (`server`). For example, the full names of the two classes in `accumulator` are:

- `examples::server::accumulator` (server class)
- `examples::accumulator` (client class)

The server class

The following code is from: `accumulator.hpp`.

All *HPX* component server classes must inherit publicly from the *HPX* component base class: `hpx::components::component_base`

The `accumulator` component inherits from `hpx::components::locking_hook`. This allows the runtime system to ensure that all action invocations are serialized. That means that the system ensures that no two actions are invoked at the same time on a given component instance. This makes the component thread safe and no additional locking has to be implemented by the user. Moreover, an `accumulator` component is a component because it also inherits from `hpx::components::component_base` (the template argument passed to `locking_hook` is used as its base class). The following snippet shows the corresponding code:

```
class accumulator
: public hpx::components::locking_hook<
    hpx::components::component_base<accumulator>>
```

Our `accumulator` class will need a data member to store its value in, so let's declare a data member:

```
argument_type value_;
```

The constructor for this class simply initializes `value_` to 0:

```
accumulator()
: value_(0)
{}
```

Next, let's look at the three methods of this component that we will be exposing as component actions:

Here are the action types. These types wrap the methods we're exposing. The wrapping technique is very similar to the one used in the [Asynchronous execution with actions](#) and the [Remote execution with actions](#):

```
HPX_DEFINE_COMPONENT_ACTION(accumulator, reset)
HPX_DEFINE_COMPONENT_ACTION(accumulator, add)
HPX_DEFINE_COMPONENT_ACTION(accumulator, query)
```

The last piece of code in the server class header is the declaration of the action type registration code:

```
HPX_REGISTER_ACTION_DECLARATION(
    examples::server::accumulator::reset_action, accumulator_reset_action)

HPX_REGISTER_ACTION_DECLARATION(
    examples::server::accumulator::add_action, accumulator_add_action)

HPX_REGISTER_ACTION_DECLARATION(
    examples::server::accumulator::query_action, accumulator_query_action)
```

Note: The code above must be placed in the global namespace.

The rest of the registration code is in `accumulator.cpp`

```
///////////////////////////////
// Add factory registration functionality.
HPX_REGISTER_COMPONENT_MODULE()

///////////////////////////////
typedef hpx::components::component<examples::server::accumulator>
    accumulator_type;

HPX_REGISTER_COMPONENT(accumulator_type, accumulator)

///////////////////////////////
// Serialization support for accumulator actions.
HPX_REGISTER_ACTION(
    accumulator_type::wrapped_type::reset_action, accumulator_reset_action)
HPX_REGISTER_ACTION(
    accumulator_type::wrapped_type::add_action, accumulator_add_action)
HPX_REGISTER_ACTION(
    accumulator_type::wrapped_type::query_action, accumulator_query_action)
```

Note: The code above must be placed in the global namespace.

The client class

The following code is from `accumulator.hpp`.

The client class is the primary interface to a component instance. Client classes are used to create components:

```
// Create a component on this locality.
examples::accumulator c = hpx::new<examples::accumulator>(hpx::find_here());
```

and to invoke component actions:

```
c.add(hpx::launch::apply, 4);
```

Clients, like servers, need to inherit from a base class, this time, `hpx::components::client_base`:

```
class accumulator
: public hpx::components::client_base<accumulator, server::accumulator>
```

For readability, we `typedef` the base class like so:

```
typedef hpx::components::client_base<accumulator, server::accumulator>
base_type;
```

Here are examples of how to expose actions through a client class:

There are a few different ways of invoking actions:

- **Non-blocking:** For actions that don't have return types, or when we do not care about the result of an action, we can invoke the action using fire-and-forget semantics. This means that once we have asked *HPX* to compute the action, we forget about it completely and continue with our computation. We use `hpx::apply` to invoke an action in a non-blocking fashion.

```
void reset(hpx::launch::apply_policy)
{
    HPX_ASSERT(this->get_id());

    typedef server::accumulator::reset_action action_type;
    hpx::apply<action_type>(this->get_id());
}
```

- **Asynchronous:** Futures, as demonstrated in [Asynchronous execution](#), [Asynchronous execution with actions](#), and the [Remote execution with actions](#), enable asynchronous action invocation. Here's an example from the accumulator client class:

```
hpx::future<argument_type> query(hpx::launch::async_policy)
{
    HPX_ASSERT(this->get_id());

    typedef server::accumulator::query_action action_type;
    return hpx::async<action_type>(hpx::launch::async, this->get_id());
}
```

- **Synchronous:** To invoke an action in a fully synchronous manner, we can simply call `hpx::async().get()` (i.e., create a future and immediately wait on it to be ready). Here's an example from the accumulator client class:

```
void add(argument_type arg)
{
    HPX_ASSERT(this->get_id());

    typedef server::accumulator::add_action action_type;
    action_type() (this->get_id(), arg);
}
```

Note that `this->get_id()` references a data member of the `hpx::components::client_base` base class which identifies the server accumulator instance.

`hpx::naming::id_type` is a type which represents a global identifier in *HPX*. This type specifies the target of an action. This is the type that is returned by `hpx::find_here` in which case it represents the *locality* the code is running on.

2.2.6 Dataflow

HPX provides its users with several different tools to simply express parallel concepts. One of these tools is a *local control object (LCO)* called dataflow. An *LCO* is a type of component that can spawn a new thread when triggered. They are also distinguished from other components by a standard interface that allow users to understand and use them easily. A Dataflow, being an *LCO*, is triggered when the values it depends on become available. For instance, if you have a calculation X that depends on the results of three other calculations, you could set up a dataflow that would begin the calculation X as soon as the other three calculations have returned their values. Dataflows are set up to depend on other dataflows. It is this property that makes dataflow a powerful parallelization tool. If you understand the dependencies of your calculation, you can devise a simple algorithm that sets up a dependency tree to be executed. In this example, we calculate compound interest. To calculate compound interest, one must calculate the interest made in each compound period, and then add that interest back to the principal before calculating the interest made in the next period. A practical person would, of course, use the formula for compound interest:

$$F = P(1 + i)^n$$

where F is the future value, P is the principal value, i is the interest rate, and n is the number of compound periods. However, for the sake of this example, we have chosen to manually calculate the future value by iterating:

$$I = Pi$$

and

$$P = P + I$$

Setup

The source code for this example can be found here: `interest_calculator.cpp`.

To compile this program, go to your HPX build directory (see `hpx_build_system` for information on configuring and building HPX) and enter:

```
$ make examples.quickstart.interest_calculator
```

To run the program type:

```
$ ./bin/interest_calculator --principal 100 --rate 5 --cp 6 --time 36
Final amount: 134.01
Amount made: 34.0096
```

Walkthrough

Let us begin with main. Here we can see that we again are using Boost.Program Options to set our command line variables (see *Asynchronous execution with actions* for more details). These options set the principal, rate, compound period, and time. It is important to note that the units of time for cp and time must be the same.

```
int main(int argc, char** argv)
{
    options_description cmdline("Usage: " HPX_APPLICATION_STRING " [options]");
    cmdline.add_options() ("principal", value<double>() -> default_value(1000),
        "The principal [$]") ("rate", value<double>() -> default_value(7),
        "The interest rate [%]") ("cp", value<int>() -> default_value(12),
```

(continues on next page)

(continued from previous page)

```

"The compound period [months]) ("time",
value<int>() -> default_value(12 * 30),
"The time money is invested [months]");

hpx::init_params init_args;
init_args.desc_cmdline = cmdline;

return hpx::init(argc, argv, init_args);
}

```

Next we look at `hpx_main`.

```

int hpx_main(variables_map& vm)
{
{
    using hpx::dataflow;
    using hpx::make_ready_future;
    using hpx::shared_future;
    using hpx::unwrapping;
    hpx::id_type here = hpx::find_here();

    double init_principal =
        vm["principal"].as<double>();           //Initial principal
    double init_rate = vm["rate"].as<double>();      //Interest rate
    int cp = vm["cp"].as<int>();       //Length of a compound period
    int t = vm["time"].as<int>();       //Length of time money is invested

    init_rate /= 100;      //Rate is a % and must be converted
    t /= cp;              //Determine how many times to iterate interest calculation:
                          //How many full compound periods can fit in the time invested

    // In non-dataflow terms the implemented algorithm would look like:
    //
    // int t = 5;      // number of time periods to use
    // double principal = init_principal;
    // double rate = init_rate;
    //
    // for (int i = 0; i < t; ++i)
    //{
    //     double interest = calc(principal, rate);
    //     principal = add(principal, interest);
    //}
    //
    // Please note the similarity with the code below!

    shared_future<double> principal = make_ready_future(init_principal);
    shared_future<double> rate = make_ready_future(init_rate);

    for (int i = 0; i < t; ++i)
    {
        shared_future<double> interest =
            dataflow(unwrapping(calc), principal, rate);
        principal = dataflow(unwrapping(add), principal, interest);
    }

    // wait for the dataflow execution graph to be finished calculating our
    // overall interest
}

```

(continues on next page)

(continued from previous page)

```

double result = principal.get();

std::cout << "Final amount: " << result << std::endl;
std::cout << "Amount made: " << result - init_principal << std::endl;
}

return hpx::finalize();
}

```

Here we find our command line variables read in, the rate is converted from a percent to a decimal, the number of calculation iterations is determined, and then our shared_futures are set up. Notice that we first place our principal and rate into shares futures by passing the variables `init_principal` and `init_rate` using `hpx::make_ready_future`.

In this way `hpx::shared_future<double> principal` and `rate` will be initialized to `init_principal` and `init_rate` when `hpx::make_ready_future<double>` returns a future containing those initial values. These shared futures then enter the for loop and are passed to `interest`. Next `principal` and `interest` are passed to the reassignment of `principal` using a `hpx::dataflow`. A dataflow will first wait for its arguments to be ready before launching any callbacks, so add in this case will not begin until both `principal` and `interest` are ready. This loop continues for each compound period that must be calculated. To see how `interest` and `principal` are calculated in the loop, let us look at `calc_action` and `add_action`:

```

// Calculate interest for one period
double calc(double principal, double rate)
{
    return principal * rate;
}

///////////////////////////////
// Add the amount made to the principal
double add(double principal, double interest)
{
    return principal + interest;
}

```

After the shared future dependencies have been defined in `hpx_main`, we see the following statement:

```
double result = principal.get();
```

This statement calls `hpx::future::get` on the shared future `principal` which had its value calculated by our for loop. The program will wait here until the entire dataflow tree has been calculated and the value assigned to `result`. The program then prints out the final value of the investment and the amount of interest made by subtracting the final value of the investment from the initial value of the investment.

2.2.7 Local to remote

When developers write code they typically begin with a simple serial code and build upon it until all of the required functionality is present. The following set of examples were developed to demonstrate this iterative process of evolving a simple serial program to an efficient, fully-distributed *HPX* application. For this demonstration, we implemented a 1D heat distribution problem. This calculation simulates the diffusion of heat across a ring from an initialized state to some user-defined point in the future. It does this by breaking each portion of the ring into discrete segments and using the current segment's temperature and the temperature of the surrounding segments to calculate the temperature of the current segment in the next timestep as shown by Fig. 2.2 below.

We parallelize this code over the following eight examples:

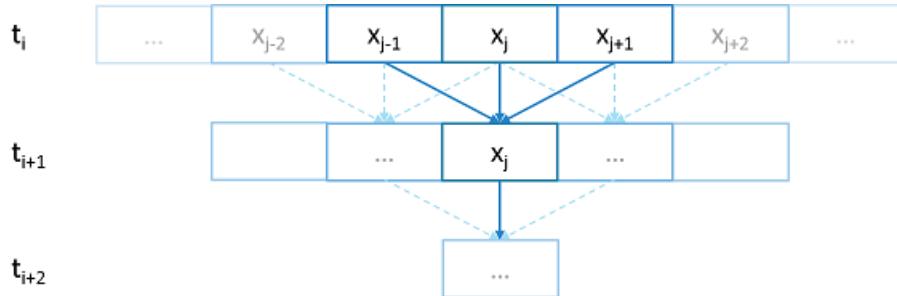


Fig. 2.2: Heat diffusion example program flow.

- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6
- Example 7
- Example 8

The first example is straight serial code. In this code we instantiate a vector U that contains two vectors of doubles as seen in the structure stepper.

```
struct stepper
{
    // Our partition type
    typedef double partition;

    // Our data for one time step
    typedef std::vector<partition> space;

    // Our operator
    static double heat(double left, double middle, double right)
    {
        return middle + (k * dt / (dx * dx)) * (left - 2 * middle + right);
    }

    // do all the work on 'nx' data points for 'nt' time steps
    space do_work(std::size_t nx, std::size_t nt)
    {
        // U[t][i] is the state of position i at time t.
        std::vector<space> U(2);
        for (space& s : U)
            s.resize(nx);

        // Initial conditions: f(0, i) = i
        for (std::size_t i = 0; i != nx; ++i)
            U[0][i] = double(i);

        // Actual time step loop
        for (std::size_t t = 0; t != nt; ++t)
    }
```

(continues on next page)

(continued from previous page)

```

{
    space const& current = U[t % 2];
    space& next = U[(t + 1) % 2];

    next[0] = heat(current[nx - 1], current[0], current[1]);

    for (std::size_t i = 1; i != nx - 1; ++i)
        next[i] = heat(current[i - 1], current[i], current[i + 1]);

    next[nx - 1] = heat(current[nx - 2], current[nx - 1], current[0]);
}

// Return the solution at time-step 'nt'.
return U[nt % 2];
}
};

```

Each element in the vector of doubles represents a single grid point. To calculate the change in heat distribution, the temperature of each grid point, along with its neighbors, is passed to the function `heat`. In order to improve readability, references named `current` and `next` are created which, depending on the time step, point to the first and second vector of doubles. The first vector of doubles is initialized with a simple heat ramp. After calling the `heat` function with the data in the `current` vector, the results are placed into the `next` vector.

In example 2 we employ a technique called futurization. Futurization is a method by which we can easily transform a code that is serially executed into a code that creates asynchronous threads. In the simplest case this involves replacing a variable with a future to a variable, a function with a future to a function, and adding a `.get()` at the point where a value is actually needed. The code below shows how this technique was applied to the `stepper`.

```

struct stepper
{
    // Our partition type
    typedef hpx::shared_future<double> partition;

    // Our data for one time step
    typedef std::vector<partition> space;

    // Our operator
    static double heat(double left, double middle, double right)
    {
        return middle + (k * dt / (dx * dx)) * (left - 2 * middle + right);
    }

    // do all the work on 'nx' data points for 'nt' time steps
    hpx::future<space> do_work(std::size_t nx, std::size_t nt)
    {
        using hpx::dataflow;
        using hpx::unwrapping;

        // U[t][i] is the state of position i at time t.
        std::vector<space> U(2);
        for (space& s : U)
            s.resize(nx);

        // Initial conditions: f(0, i) = i
        for (std::size_t i = 0; i != nx; ++i)
            U[0][i] = hpx::make_ready_future(double(i));
    }
};

```

(continues on next page)

(continued from previous page)

```

auto Op = unwrapping(&stepper::heat);

// Actual time step loop
for (std::size_t t = 0; t != nt; ++t)
{
    space const& current = U[t % 2];
    space& next = U[(t + 1) % 2];

    // WHEN U[t][i-1], U[t][i], and U[t][i+1] have been computed, THEN we
    // can compute U[t+1][i]
    for (std::size_t i = 0; i != nx; ++i)
    {
        next[i] =
            dataflow(hpx::launch::async, Op, current[idx(i, -1, nx)],
                     current[i], current[idx(i, +1, nx)]);
    }
}

// Now the asynchronous computation is running; the above for-loop does not
// wait on anything. There is no implicit waiting at the end of each timestep;
// the computation of each U[t][i] will begin as soon as its dependencies
// are ready and hardware is available.

// Return the solution at time-step 'nt'.
return hpx::when_all(U[nt % 2]);
}
};

```

In example 2, we redefine our partition type as a `shared_future` and, in `main`, create the object `result`, which is a future to a vector of partitions. We use `result` to represent the last vector in a string of vectors created for each timestep. In order to move to the next timestep, the values of a partition and its neighbors must be passed to `heat` once the futures that contain them are ready. In *HPX*, we have an LCO (Local Control Object) named `Dataflow` that assists the programmer in expressing this dependency. `Dataflow` allows us to pass the results of a set of futures to a specified function when the futures are ready. `Dataflow` takes three types of arguments, one which instructs the `Dataflow` on how to perform the function call (async or sync), the function to call (in this case `Op`), and futures to the arguments that will be passed to the function. When called, `Dataflow` immediately returns a future to the result of the specified function. This allows users to string `Dataflows` together and construct an execution tree.

After the values of the futures in `Dataflow` are ready, the values must be pulled out of the future container to be passed to the function `heat`. In order to do this, we use the *HPX* facility `unwrapping`, which underneath calls `.get()` on each of the futures so that the function `heat` will be passed doubles and not futures to doubles.

By setting up the algorithm this way, the program will be able to execute as quickly as the dependencies of each future are met. Unfortunately, this example runs terribly slow. This increase in execution time is caused by the overheads needed to create a future for each data point. Because the work done within each call to `heat` is very small, the overhead of creating and scheduling each of the three futures is greater than that of the actual useful work! In order to amortize the overheads of our synchronization techniques, we need to be able to control the amount of work that will be done with each future. We call this amount of work per overhead grain size.

In example 3, we return to our serial code to figure out how to control the grain size of our program. The strategy that we employ is to create “partitions” of data points. The user can define how many partitions are created and how many data points are contained in each partition. This is accomplished by creating the `struct partition`, which contains a member object `data_`, a vector of doubles that holds the data points assigned to a particular instance of `partition`.

In example 4, we take advantage of the partition setup by redefining `space` to be a vector of `shared_futures` with each

future representing a partition. In this manner, each future represents several data points. Because the user can define how many data points are in each partition, and, therefore, how many data points are represented by one future, a user can control the grainsize of the simulation. The rest of the code is then futurized in the same manner as example 2. It should be noted how strikingly similar example 4 is to example 2.

Example 4 finally shows good results. This code scales equivalently to the OpenMP version. While these results are promising, there are more opportunities to improve the application's scalability. Currently, this code only runs on one *locality*, but to get the full benefit of *HPX*, we need to be able to distribute the work to other machines in a cluster. We begin to add this functionality in example 5.

In order to run on a distributed system, a large amount of boilerplate code must be added. Fortunately, *HPX* provides us with the concept of a *component*, which saves us from having to write quite as much code. A component is an object that can be remotely accessed using its global address. Components are made of two parts: a server and a client class. While the client class is not required, abstracting the server behind a client allows us to ensure type safety instead of having to pass around pointers to global objects. Example 5 renames example 4's `struct partition` to `partition_data` and adds serialization support. Next, we add the server side representation of the data in the structure `partition_server`. `Partition_server` inherits from `hpx::components::component_base`, which contains a server-side component boilerplate. The boilerplate code allows a component's public members to be accessible anywhere on the machine via its Global Identifier (GID). To encapsulate the component, we create a client side helper class. This object allows us to create new instances of our component and access its members without having to know its GID. In addition, we are using the client class to assist us with managing our asynchrony. For example, our client class `partition`'s member function `get_data()` returns a future to `partition_data` `get_data()`. This struct inherits its boilerplate code from `hpx::components::client_base`.

In the structure `stepper`, we have also had to make some changes to accommodate a distributed environment. In order to get the data from a particular neighboring partition, which could be remote, we must retrieve the data from all of the neighboring partitions. These retrievals are asynchronous and the function `heat_part_data`, which, amongst other things, calls `heat`, should not be called unless the data from the neighboring partitions have arrived. Therefore, it should come as no surprise that we synchronize this operation with another instance of dataflow (found in `heat_part`). This dataflow receives futures to the data in the current and surrounding partitions by calling `get_data()` on each respective partition. When these futures are ready, dataflow passes them to the unwrapping function, which extracts the `shared_array` of doubles and passes them to the lambda. The lambda calls `heat_part_data` on the *locality*, which the middle partition is on.

Although this example could run distributed, it only runs on one *locality*, as it always uses `hpx::find_here()` as the target for the functions to run on.

In example 6, we begin to distribute the partition data on different nodes. This is accomplished in `stepper::do_work()` by passing the GID of the *locality* where we wish to create the partition to the partition constructor.

```
for (std::size_t i = 0; i != np; ++i)
    U[0][i] = partition(localities[locidx(i, np, nl)], nx, double(i));
```

We distribute the partitions evenly based on the number of localities used, which is described in the function `locidx`. Because some of the data needed to update the partition in `heat_part` could now be on a new *locality*, we must devise a way of moving data to the *locality* of the middle partition. We accomplished this by adding a switch in the function `get_data()` that returns the end element of the buffer `data_` if it is from the left partition or the first element of the buffer if the data is from the right partition. In this way only the necessary elements, not the whole buffer, are exchanged between nodes. The reader should be reminded that this exchange of end elements occurs in the function `get_data()` and, therefore, is executed asynchronously.

Now that we have the code running in distributed, it is time to make some optimizations. The function `heat_part` spends most of its time on two tasks: retrieving remote data and working on the data in the middle partition. Because we know that the data for the middle partition is local, we can overlap the work on the middle partition with that of the possibly remote call of `get_data()`. This algorithmic change, which was implemented in example 7, can be seen below:

```

// The partitioned operator, it invokes the heat operator above on all elements
// of a partition.
static partition heat_part(
    partition const& left, partition const& middle, partition const& right)
{
    using hpx::dataflow;
    using hpx::unwrapping;

    hpx::shared_future<partition_data> middle_data =
        middle.get_data(partition_server::middle_partition);

    hpx::future<partition_data> next_middle = middle_data.then(
        unwrapping([middle] (partition_data const& m) -> partition_data {
            HPX_UNUSED(middle);

            // All local operations are performed once the middle data of
            // the previous time step becomes available.
            std::size_t size = m.size();
            partition_data next(size);
            for (std::size_t i = 1; i != size - 1; ++i)
                next[i] = heat(m[i - 1], m[i], m[i + 1]);
            return next;
        }));
}

return dataflow(hpx::launch::async,
    unwrapping([left, middle, right] (partition_data next,
        partition_data const& l, partition_data const& m,
        partition_data const& r) -> partition {
        HPX_UNUSED(left);
        HPX_UNUSED(right);

        // Calculate the missing boundary elements once the
        // corresponding data has become available.
        std::size_t size = m.size();
        next[0] = heat(l[size - 1], m[0], m[1]);
        next[size - 1] = heat(m[size - 2], m[size - 1], r[0]);

        // The new partition_data will be allocated on the same locality
        // as 'middle'.
        return partition(middle.get_id(), std::move(next));
    }),
    std::move(next_middle),
    left.get_data(partition_server::left_partition), middle_data,
    right.get_data(partition_server::right_partition));
}

```

Example 8 completes the futurization process and utilizes the full potential of *HPX* by distributing the program flow to multiple localities, usually defined as nodes in a cluster. It accomplishes this task by running an instance of *HPX* main on each *locality*. In order to coordinate the execution of the program, the `struct stepper` is wrapped into a component. In this way, each *locality* contains an instance of stepper that executes its own instance of the function `do_work()`. This scheme does create an interesting synchronization problem that must be solved. When the program flow was being coordinated on the head node, the GID of each component was known. However, when we distribute the program flow, each partition has no notion of the GID of its neighbor if the next partition is on another *locality*. In order to make the GIDs of neighboring partitions visible to each other, we created two buffers to store the GIDs of the remote neighboring partitions on the left and right respectively. These buffers are filled by sending the GID of newly created edge partitions to the right and left buffers of the neighboring localities.

In order to finish the simulation, the solution vectors named `result` are then gathered together on *locality* 0 and

added into a vector of spaces `overall_result` using the *HPX* functions `gather_id` and `gather_here`.

Example 8 completes this example series, which takes the serial code of example 1 and incrementally morphs it into a fully distributed parallel code. This evolution was guided by the simple principles of futurization, the knowledge of grainsize, and utilization of components. Applying these techniques easily facilitates the scalable parallelization of most applications.

2.3 Manual

The manual is your comprehensive guide to *HPX*. It contains detailed information on how to build and use *HPX* in different scenarios.

2.3.1 Prerequisites

Supported platforms

At this time, *HPX* supports the following platforms. Other platforms may work, but we do not test *HPX* with other platforms, so please be warned.

Table 2.1: Supported Platforms for *HPX*

Name	Minimum Version	Architectures
Linux	2.6	x86-32, x86-64, k1om
BlueGeneQ	V1R2M0	PowerPC A2
Windows	Any Windows system	x86-32, x86-64
Mac OSX	Any OSX system	x86-64

Supported compilers

The table below shows the supported compilers for *HPX*.

Table 2.2: Supported Compilers for *HPX*

Name	Minimum Version
GNU Compiler Collection (g++) ¹³	8.0
clang: a C language family frontend for LLVM ¹⁴	8.0
Visual C++ ¹⁵ (x64)	2015

¹³ <https://gcc.gnu.org>

¹⁴ <https://clang.llvm.org/>

¹⁵ <https://msdn.microsoft.com/en-us/visualc/default.aspx>

Software and libraries

The table below presents all the necessary prerequisites for building *HPX*.

Table 2.3: Software prerequisites for *HPX*

	Name	Minimum Version
Build System	CMake ¹⁶	3.18
Required Libraries	Boost ¹⁷	1.71.0
	Portable Hardware Locality (HWLOC) ¹⁸	1.5
	Asio ¹⁹	1.12.0

The most important dependencies are Boost²⁰ and Portable Hardware Locality (HWLOC)²¹. The installation of Boost is described in detail in Boost's [Getting Started²²](#) document. A recent version of hwloc is required in order to support thread pinning and NUMA awareness and can be found in [Hwloc Downloads²³](#).

HPX is written in 99.99% Standard C++ (the remaining 0.01% is platform specific assembly code). As such, *HPX* is compilable with almost any standards compliant C++ compiler. The code base takes advantage of C++ language and standard library features when available.

Note: When building Boost using gcc, please note that it is required to specify a `cxxflags=-std=c++17` command line argument to `b2` (`bjam`).

Note: In most configurations, *HPX* depends only on header-only Boost. Boost.Filesystem is required if the standard library does not support filesystem. The following are not needed by default, but are required in certain configurations: Boost.Chrono, Boost.DateTime, Boost.Log, Boost.LogSetup, Boost.Regex, and Boost.Thread.

Depending on the options you chose while building and installing *HPX*, you will find that *HPX* may depend on several other libraries such as those listed below.

Note: In order to use a high speed parcelport, we currently recommend configuring *HPX* to use MPI so that MPI can be used for communication between different localities. Please set the CMake variable `MPI_CXX_COMPILER` to your MPI C++ compiler wrapper if not detected automatically.

Table 2.4: Optional software prerequisites for *HPX*

Name	Minimum version
google-perftools ²⁴	1.7.1
jemalloc ²⁵	2.1.0
mi-malloc ²⁶	1.0.0
Performance Application Programming Interface (PAPI)	

¹⁶ <https://www.cmake.org>

¹⁷ <https://www.boost.org/>

¹⁸ <https://www.open-mpi.org/projects/hwloc/>

¹⁹ <https://think-async.com/asio/>

²⁰ <https://www.boost.org/>

²¹ <https://www.open-mpi.org/projects/hwloc/>

²² https://www.boost.org/more/getting_started/index.html

²³ <https://www.open-mpi.org/software/hwloc/v1.11>

2.3.2 Getting HPX

Download a tarball of the latest release from [HPX Downloads²⁷](#) and unpack it or clone the repository directly using git:

```
$ git clone https://github.com/STELLAR-GROUP/hpx.git
```

It is also recommended that you check out the latest stable tag:

```
$ cd hpx
$ git checkout 1.7.1
```

2.3.3 Building HPX

Basic information

The build system for *HPX* is based on [CMake²⁸](#), a cross-platform build-generator tool which is not responsible for building the project but rather generates the files needed by your build tool (GNU make, Visual Studio, etc.) for building *HPX*. If CMake is not already installed in your system, you can download it and install it here: [CMake Downloads²⁹](#).

Once CMake has been run, the build process can be started. The *HPX* build process is highly configurable through CMake, and various CMake variables influence the build process. The build process consists of the following parts:

- The *HPX* core libraries (target `core`): This forms the basic set of *HPX* libraries.
- *HPX* Examples (target `examples`): This target is enabled by default and builds all *HPX* examples (disable by setting `HPX_WITH_EXAMPLES:BOOL=Off`). *HPX* examples are part of the `all` target and are included in the installation if enabled.
- *HPX* Tests (target `tests`): This target builds the *HPX* test suite and is enabled by default (disable by setting `HPX_WITH_TESTS:BOOL=Off`). They are not built by the `all` target and have to be built separately.
- *HPX* Documentation (target `docs`): This target builds the documentation, and is not enabled by default (enable by setting `HPX_WITH_DOCUMENTATION:BOOL=On`. For more information see [Documentation](#).

For a complete list of available CMake variables that influence the build of *HPX*, see [CMake variables used to configure HPX](#).

The variables can be used to refine the recipes that can be found at [Platform specific build recipes](#) which show some basic steps on how to build *HPX* for a specific platform.

In order to use *HPX*, only the core libraries are required. In order to use the optional libraries, you need to specify them as link dependencies in your build (See [Creating HPX projects](#)).

²⁴ <https://code.google.com/p/gperftools>

²⁵ <http://jemalloc.net>

²⁶ <http://microsoft.github.io/mimalloc/>

²⁷ <https://hpx.stellar-group.org/downloads/>

²⁸ <https://www.cmake.org>

²⁹ <https://www.cmake.org/cmake/resources/software.html>

Most important CMake options

While building *HPX*, you are provided with multiple CMake options which correspond to different configurations. Below, there is a set of the most important and frequently used CMake options.

HPX_WITH_MALLOC

Use a custom allocator. Using a custom allocator tuned for multithreaded applications is very important for the performance of *HPX* applications. When debugging applications, it's useful to set this to `system`, as custom allocators can hide some memory-related bugs. Note that setting this to something other than `system` requires an external dependency.

HPX_WITH_CUDA

Enable support for CUDA. Use `CMAKE_CUDA_COMPILER` to set the CUDA compiler. This is a standard CMake variable, like `CMAKE_CXX_COMPILER`.

HPX_WITH_PARCELPORT_MPI

Enable the MPI parcelport. This enables the use of MPI for the networking operations in the *HPX* runtime. The default value is `OFF` because it's not available on all systems and/or requires another dependency. However, it is the recommended parcelport.

HPX_WITH_PARCELPORT_TCP

Enable the TCP parcelport. Enables the use of TCP for networking in the runtime. The default value is `ON`. However, it's only recommended for debugging purposes, as it is slower than the MPI parcelport.

HPX_WITH_APEX

Enable APEX integration. APEX³⁰ can be used to profile *HPX* applications. In particular, it provides information about individual tasks in the *HPX* runtime.

HPX_WITH_GENERIC_CONTEXT_COROUTINES

Enable Boost. Context for task context switching. It must be enabled for non-x86 architectures such as ARM and Power.

HPX_WITH_MAX_CPU_COUNT

Set the maximum CPU count supported by *HPX*. The default value is 64, and should be set to a number at least as high as the number of cores on a system including virtual cores such as hyperthreads.

HPX_WITH_CXX_STANDARD

Set a specific C++ standard version e.g. `HPX_WITH_CXX_STANDARD=20`. The default and minimum value is 17.

HPX_WITH_EXAMPLES

Build examples.

HPX_WITH_TESTS

Build tests.

For a complete list of available CMake variables that influence the build of *HPX*, see [CMake variables used to configure *HPX*](#).

³⁰ <https://uo-oaciss.github.io/apex/quickstarthpx/>

Build types

CMake can be configured to generate project files suitable for builds that have enabled debugging support or for an optimized build (without debugging support). The CMake variable used to set the build type is `CMAKE_BUILD_TYPE` (for more information see the [CMake Documentation](#)³¹). Available build types are:

- **Debug:** Full debug symbols are available as well as additional assertions to help debugging. To enable the debug build type for the *HPX* API, the C++ Macro `HPX_DEBUG` is defined.
- **RelWithDebInfo:** Release build with debugging symbols. This is most useful for profiling applications
- **Release:** Release build. This disables assertions and enables default compiler optimizations.
- **RelMinSize:** Release build with optimizations for small binary sizes.

Important: We currently don't guarantee ABI compatibility between Debug and Release builds. Please make sure that applications built against *HPX* use the same build type as you used to build *HPX*. For CMake builds, this means that the `CMAKE_BUILD_TYPE` variables have to match and for projects not using [CMake](#)³², the `HPX_DEBUG` macro has to be set in debug mode.

Platform specific build recipes

Unix variants

Once you have the source code and the dependencies and assuming all your dependencies are in paths known to CMake, the following gets you started:

1. First, set up a separate build directory to configure the project:

```
$ mkdir build && cd build
```

2. To configure the project you have the following options:

- To build the core *HPX* libraries and examples, and install them to your chosen location (recommended):

```
$ cmake -DCMAKE_INSTALL_PREFIX=/install/path ..
```

Tip: If you want to change CMake variables for your build, it is usually a good idea to start with a clean build directory to avoid configuration problems. It is especially important that you use a clean build directory when changing between `Release` and `Debug` modes.

- To install *HPX* to the default system folders, simply leave out the `CMAKE_INSTALL_PREFIX` option:

```
$ cmake ..
```

- If your dependencies are in custom locations, you may need to tell CMake where to find them by passing one or more options to CMake as shown below:

```
$ cmake -DBOOST_ROOT=/path/to/boost  
-DHWLOC_ROOT=/path/to/hwloc  
-DTCMALLOC_ROOT=/path/to/tcmalloc
```

(continues on next page)

³¹ https://cmake.org/cmake/help/latest/variable/CMAKE_BUILD_TYPE.html

³² <https://www.cmake.org>

(continued from previous page)

```
-DJEMALLOC_ROOT=/path/to/jemalloc
[other CMake variable definitions]
/path/to/source/tree
```

For instance:

```
$ cmake -DBOOST_ROOT=~/packages/boost -DHWLOC_ROOT=/packages/hwloc -
-DCMAKE_INSTALL_PREFIX=~/packages/hpx ~/downloads/hpx_1.5.1
```

- If you want to try *HPX* without using a custom allocator pass `-DHPX_WITH_MALLOC=system` to CMake:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/install/path -DHPX_WITH_MALLOC=system ..
```

Note: Please pay special attention to the section about `HPX_WITH_MALLOC:STRING` as this is crucial for getting decent performance.

Important: If you are building *HPX* for a system with more than 64 processing units, you must change the CMake variable `HPX_WITH_MAX_CPU_COUNT` (to a value at least as big as the number of (virtual) cores on your system). Note that the default value is 64.

Caution: Compiling and linking *HPX* needs a considerable amount of memory. It is advisable that at least 2 GB of memory per parallel process is available.

3. Once the configuration is complete, to build the project you run:

```
$ cmake --build . --target install
```

Windows

Note: The following build recipes are mostly user-contributed and may be outdated. We always welcome updated and new build recipes.

To build *HPX* under Windows 10 x64 with Visual Studio 2015:

- Download the CMake V3.18.1 installer (or latest version) from [here³³](#)
- Download the hwloc V1.11.0 (or the latest version) from [here³⁴](#) and unpack it.
- Download the latest Boost libraries from [here³⁵](#) and unpack them.
- Build the Boost DLLs and LIBs by using these commands from Command Line (or PowerShell). Open CMD/PowerShell inside the Boost dir and type in:

³³ <https://blog.kitware.com/cmake-3-18-1-available-for-download/>

³⁴ <http://www.open-mpi.org/software/hwloc/v1.11/downloads/hwloc-win64-build-1.11.0.zip>

³⁵ <https://www.boost.org/users/download/>

```
bootstrap.bat
```

This batch file will set up everything needed to create a successful build. Now execute:

```
b2.exe link=shared variant=release,debug architecture=x86 address-model=64
  ↵threading=multi --build-type=complete install
```

This command will start a (very long) build of all available Boost libraries. Please, be patient.

- Open CMake-GUI.exe and set up your source directory (input field ‘Where is the source code’) to the *base directory* of the source code you downloaded from HPX’s GitHub pages. Here’s an example of CMake path settings, which point to the Documents/GitHub/hpx folder:

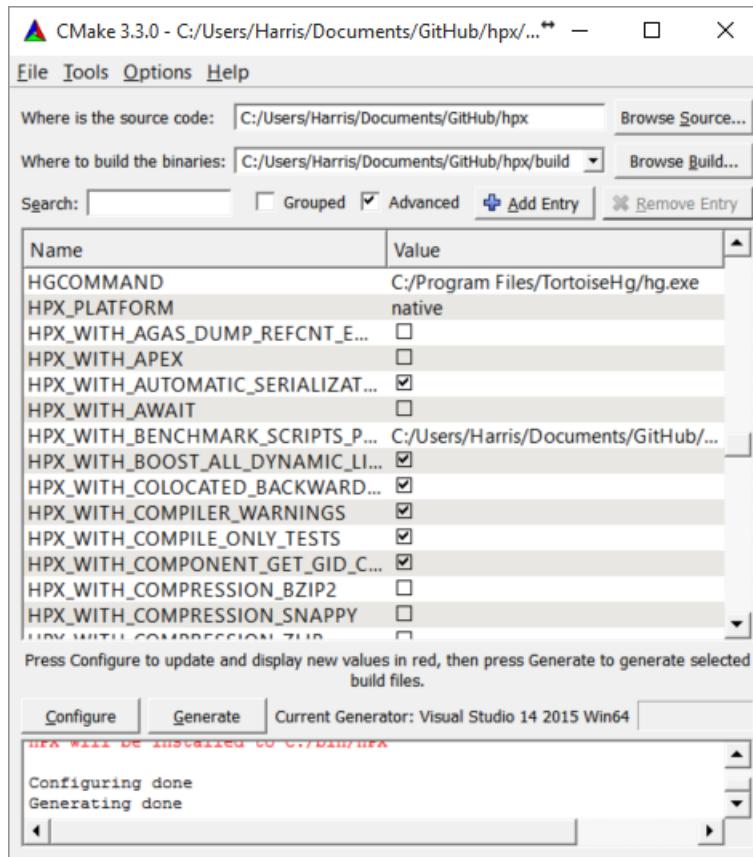


Fig. 2.3: Example CMake path settings.

Inside ‘Where is the source-code’ enter the base directory of your HPX source directory (do not enter the “src” sub-directory!). Inside ‘Where to build the binaries’ you should put in the path where all the building processes will happen. This is important because the building machinery will do an “out-of-tree” build. CMake will not touch or change the original source files in any way. Instead, it will generate Visual Studio Solution Files, which will build HPX packages out of the HPX source tree.

- Set three new environment variables (in CMake, not in Windows environment): BOOST_ROOT, HWLOC_ROOT, CMAKE_INSTALL_PREFIX. The meaning of these variables is as follows:
 - BOOST_ROOT the HPX root directory of the unpacked Boost headers/cpp files.
 - HWLOC_ROOT the HPX root directory of the unpacked Portable Hardware Locality files.
 - CMAKE_INSTALL_PREFIX the HPX root directory where the future builds of HPX should be installed.

Note: HPX is a very large software collection, so it is not recommended to use the default C:\Program Files\hpx. Many users may prefer to use simpler paths *without* whitespace, like C:\bin\hpx or D:\bin\hpx etc.

To insert new env-vars click on “Add Entry” and then insert the name inside “Name”, select PATH as Type and put the path-name in the “Path” text field. Repeat this for the first three variables.

This is how variable insertion will look:

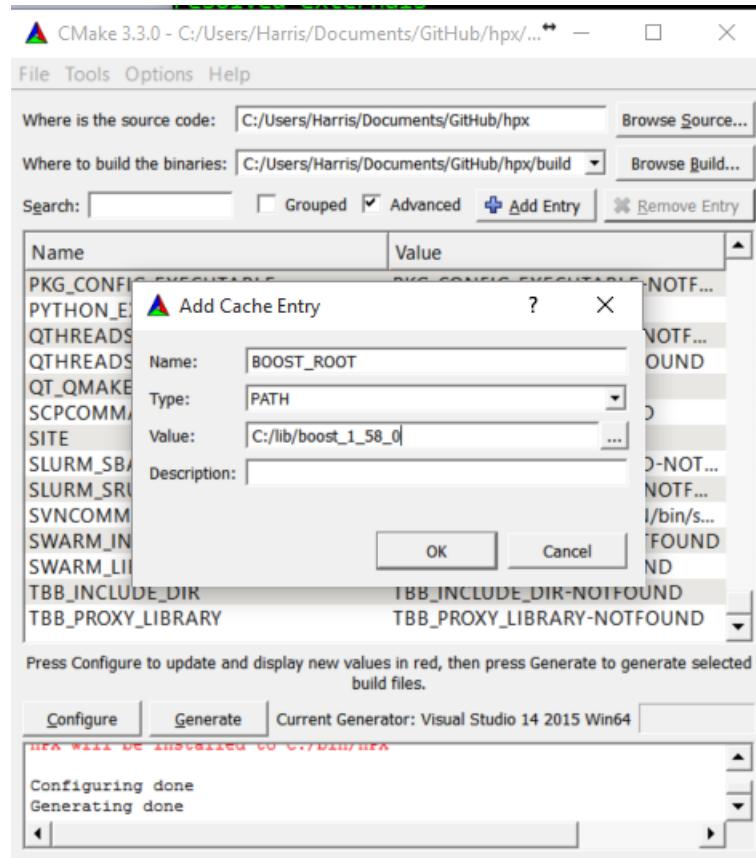


Fig. 2.4: Example CMake adding entry.

Alternatively, users could provide BOOST_LIBRARYDIR instead of BOOST_ROOT; the difference is that BOOST_LIBRARYDIR should point to the subdirectory inside Boost root where all the compiled DLLs/LIBs are. For example, BOOST_LIBRARYDIR may point to the bin.v2 subdirectory under the Boost rootdir. It is important to keep the meanings of these two variables separated from each other: BOOST_DIR points to the ROOT folder of the Boost library. BOOST_LIBRARYDIR points to the subdir inside the Boost root folder where the compiled binaries are.

- Click the ‘Configure’ button of CMake-GUI. You will be immediately presented with a small window where you can select the C++ compiler to be used within Visual Studio. This has been tested using the latest v14 (a.k.a C++ 2015) but older versions should be sufficient too. Make sure to select the 64Bit compiler.
- After the generate process has finished successfully, click the ‘Generate’ button. Now, CMake will put new VS Solution files into the BUILD folder you selected at the beginning.
- Open Visual Studio and load the HPX.sln from your build folder.

- Go to CMakePredefinedTargets and build the INSTALL project:

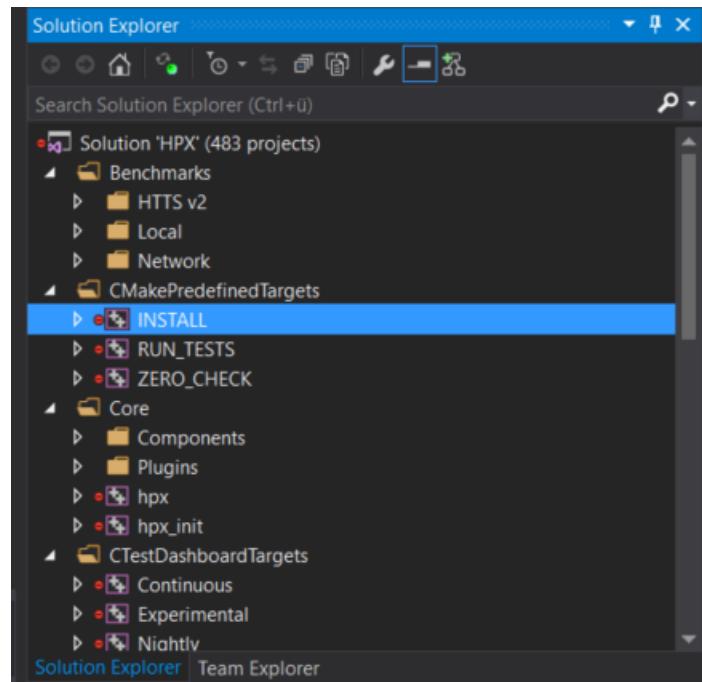


Fig. 2.5: Visual Studio INSTALL target.

It will take some time to compile everything, and in the end you should see an output similar to this one:

Tests and examples

Running tests

To build the tests:

```
$ cmake --build . --target tests
```

To control which tests to run use `ctest`:

- To run single tests, for example a test for `for_loop`:

```
$ ctest --output-on-failure -R tests.unit.modules.algorithms.for_loop
```

- To run a whole group of tests:

The screenshot shows the Visual Studio Output window with the title 'Output' at the top. The window displays a log of build commands and their results. The log includes numerous 'Installing:' entries for various executables like '1d_stencil_2.exe' through '1d_stencil_8.exe', '1d_stencil_1_omp.exe', '1d_stencil_3_omp.exe', and several transpose-related executables. It also lists library installations for 'hpx_simple_central_tuplespaced.lib' and 'hpx_simple_central_tuplespaced.dll'. The log concludes with a summary: 'Build: 116 succeeded, 0 failed, 0 up-to-date, 0 skipped ======='.

Output

```

116> -- Installing: C:/bin/HPX/bin/1d_stencil_2.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_3.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_4.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_4_parallel.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_5.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_6.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_7.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_8.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_1_omp.exe
116> -- Installing: C:/bin/HPX/bin/1d_stencil_3_omp.exe
116> -- Installing: C:/bin/HPX/bin/simple_central_tuplespace_client.exe
116> -- Installing: C:/bin/HPX/lib/hpx_simple_central_tuplespaced.lib
116> -- Installing: C:/bin/HPX/lib/hpx_simple_central_tuplespaced.dll
116> -- Installing: C:/bin/HPX/bintranspose_serial.exe
116> -- Installing: C:/bin/HPX/bintranspose_serial_block.exe
116> -- Installing: C:/bin/HPX/bintranspose_smp.exe
116> -- Installing: C:/bin/HPX/bintranspose_smp_block.exe
116> -- Installing: C:/bin/HPX/bintranspose_block.exe
116> -- Installing: C:/bin/HPX/bintranspose_serial_vector.exe
116> -- Installing: C:/bin/HPX/binhpx_runtime.exe
===== Build: 116 succeeded, 0 failed, 0 up-to-date, 0 skipped ======

```

Error List | Output | Find Symbol Results | Package Manager Console | Azure App Service Activity

Fig. 2.6: Visual Studio build output.

```
$ ctest --output-on-failure -R tests.unit
```

Running examples

- To build (and install) all examples invoke:

```
$ cmake -DHPX_WITH_EXAMPLES=On .
$ make examples
$ make install
```

- To build the `hello_world_1` example run:

```
$ make hello_world_1
```

HPX executables end up in the `bin` directory in your build directory. You can now run `hello_world_1` and should see the following output:

```
$ ./bin/hello_world_1
Hello World!
```

You've just run an example which prints `Hello World!` from the *HPX* runtime. The source for the example is in `examples/quickstart/hello_world_1.cpp`. The `hello_world_distributed` example (also available in the `examples/quickstart` directory) is a distributed hello world program, which is described in [Remote execution with actions](#). It provides a gentle introduction to the distributed aspects of *HPX*.

Tip: Most build targets in *HPX* have two names: a simple name and a hierarchical name corresponding to what type of example or test the target is. If you are developing *HPX* it is often helpful to run `make help` to get a list of 23 build targets. For example, `make help | grep hello_world` outputs the following:

```
... examples.quickstart.hello_world_2
... hello_world_2
... examples.quickstart.hello_world_1
```

Variables that influence how HPX is built

The options are split into these categories:

- *Generic options*
- *Build Targets options*
- *Thread Manager options*
- *AGAS options*
- *Parcelport options*
- *Profiling options*
- *Debugging options*
- *Modules options*

Generic options

- `HPX_WITH_AUTOMATIC_SERIALIZATION_REGISTRATION:BOOL`
- `HPX_WITH_BENCHMARK_SCRIPTS_PATH:PATH`
- `HPX_WITH_BUILD_BINARY_PACKAGE:BOOL`
- `HPX_WITH_CHECK_MODULE_DEPENDENCIES:BOOL`
- `HPX_WITH_COMPILER_WARNINGS:BOOL`
- `HPX_WITH_COMPILER_WARNINGS_AS_ERRORS:BOOL`
- `HPX_WITH_COMPRESSION_BZIP2:BOOL`
- `HPX_WITH_COMPRESSION_SNAPPY:BOOL`
- `HPX_WITH_COMPRESSION_ZLIB:BOOL`
- `HPX_WITH_CUDA:BOOL`
- `HPX_WITH_CXX_STANDARD:STRING`
- `HPX_WITH_DATAPAR_VC:BOOL`
- `HPX_WITH_DEPRECATED_WARNINGS:BOOL`
- `HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL`
- `HPX_WITH_DYNAMIC_HPX_MAIN:BOOL`
- `HPX_WITHFAULT_TOLERANCE:BOOL`
- `HPX_WITH_FULL_RPATH:BOOL`
- `HPX_WITH_GCC_VERSION_CHECK:BOOL`
- `HPX_WITH_GENERIC_CONTEXT_COROUTINES:BOOL`
- `HPX_WITH_HIDDEN_VISIBILITY:BOOL`
- `HPX_WITH_HIP:BOOL`
- `HPX_WITH_LOGGING:BOOL`
- `HPX_WITH_MALLOC:STRING`
- `HPX_WITH_MODULES_AS_STATIC_LIBRARIES:BOOL`

- `HPX_WITH_NICE_THREADLEVEL:BOOL`
- `HPX_WITH_PARCEL_COALESCING:BOOL`
- `HPX_WITH_PKGCONFIG:BOOL`
- `HPX_WITH_PRECOMPILED_HEADERS:BOOL`
- `HPX_WITH_RUN_MAIN_EVERYWHERE:BOOL`
- `HPX_WITH_STACKOVERFLOW_DETECTION:BOOL`
- `HPX_WITH_STATIC_LINKING:BOOL`
- `HPX_WITH_UNITY_BUILD:BOOL`
- `HPX_WITH_VIM_YCM:BOOL`
- `HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD:STRING`

HPX_WITH_AUTOMATIC_SERIALIZATION_REGISTRATION:BOOL

Use automatic serialization registration for actions and functions. This affects compatibility between HPX applications compiled with different compilers (default ON)

HPX_WITH_BENCHMARK_SCRIPTS_PATH:PATH

Directory to place batch scripts in

HPX_WITH_BUILD_BINARY_PACKAGE:BOOL

Build HPX on the build infrastructure on any LINUX distribution (default: OFF).

HPX_WITH_CHECK_MODULE_DEPENDENCIES:BOOL

Verify that no modules are cross-referenced from a different module category (default: OFF)

HPX_WITH_COMPILER_WARNINGS:BOOL

Enable compiler warnings (default: ON)

HPX_WITH_COMPILER_WARNINGS_AS_ERRORS:BOOL

Turn compiler warnings into errors (default: OFF)

HPX_WITH_COMPRESSION_BZIP2:BOOL

Enable bzip2 compression for parcel data (default: OFF).

HPX_WITH_COMPRESSION_SNAPPY:BOOL

Enable snappy compression for parcel data (default: OFF).

HPX_WITH_COMPRESSION_ZLIB:BOOL

Enable zlib compression for parcel data (default: OFF).

HPX_WITH_CUDA:BOOL

Enable support for CUDA (default: OFF)

HPX_WITH_CXX_STANDARD:STRING

Set the C++ standard to use when compiling HPX itself. (default: 17)

HPX_WITH_DATAPAR_VC:BOOL

Enable data parallel algorithm support using the external Vc library (default: OFF)

HPX_WITH_DEPRECATED_WARNINGS:BOOL

Enable warnings for deprecated facilities. (default: ON)

HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL

Disables the mechanism that produces debug output for caught signals and unhandled exceptions (default: OFF)

HPX_WITH_DYNAMIC_HPX_MAIN:BOOL

Enable dynamic overload of system `main()` (Linux and Apple only, default: ON)

HPX_WITH_FAULT_TOLERANCE:BOOL

Build HPX to tolerate failures of nodes, i.e. ignore errors in active communication channels (default: OFF)

HPX_WITH_FULL_RPATH:BOOL

Build and link HPX libraries and executables with full RPATHs (default: ON)

HPX_WITH_GCC_VERSION_CHECK:BOOL

Don't ignore version reported by gcc (default: ON)

HPX_WITH_GENERIC_CONTEXT_COROUTINES:BOOL

Use Boost.Context as the underlying coroutines context switch implementation.

HPX_WITH_HIDDEN_VISIBILITY:BOOL

Use -fvisibility=hidden for builds on platforms which support it (default OFF)

HPX_WITH_HIP:BOOL

Enable compilation with HIPCC (default: OFF)

HPX_WITH_LOGGING:BOOL

Build HPX with logging enabled (default: ON).

HPX_WITH_MALLOC:STRING

Define which allocator should be linked in. Options are: system, tcmalloc, jemalloc, mimalloc, tbbmalloc, and custom (default is: tcmalloc)

HPX_WITH_MODULES_AS_STATIC_LIBRARIES:BOOL

Compile HPX modules as STATIC (whole-archive) libraries instead of OBJECT libraries (Default: ON)

HPX_WITH_NICE_THREADLEVEL:BOOL

Set HPX worker threads to have high NICE level (may impact performance) (default: OFF)

HPX_WITH_PARCEL_COALESCING:BOOL

Enable the parcel coalescing plugin (default: ON).

HPX_WITH_PKGCONFIG:BOOL

Enable generation of pkgconfig files (default: ON on Linux without CUDA/HIP, otherwise OFF)

HPX_WITH_PRECOMPILED_HEADERS:BOOL

Enable precompiled headers for certain build targets (experimental) (default OFF)

HPX_WITH_RUN_MAIN_EVERYWHERE:BOOL

Run hpx_main by default on all localities (default: OFF).

HPX_WITH_STACKOVERFLOW_DETECTION:BOOL

Enable stackoverflow detection for HPX threads/coroutines. (default: OFF, debug: ON)

HPX_WITH_STATIC_LINKING:BOOL

Compile HPX statically linked libraries (Default: OFF)

HPX_WITH_UNITY_BUILD:BOOL

Enable unity build for certain build targets (default OFF)

HPX_WITH_VIM_YCM:BOOL

Generate HPX completion file for VIM YouCompleteMe plugin

HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD:STRING

The threshold in bytes to when perform zero copy optimizations (default: 128)

Build Targets options

- `HPX_WITH_ASIO_TAG:STRING`
- `HPX_WITH_COMPILE_ONLY_TESTS:BOOL`
- `HPX_WITH_DISTRIBUTED_RUNTIME:BOOL`
- `HPX_WITH_DOCUMENTATION:BOOL`
- `HPX_WITH_DOCUMENTATION_OUTPUT_FORMATS:STRING`
- `HPX_WITH_EXAMPLES:BOOL`
- `HPX_WITH_EXAMPLES_HDF5:BOOL`
- `HPX_WITH_EXAMPLES_OPENMP:BOOL`
- `HPX_WITH_EXAMPLES_QT4:BOOL`
- `HPX_WITH_EXAMPLES_QTHREADS:BOOL`
- `HPX_WITH_EXAMPLES_TBB:BOOL`
- `HPX_WITH_EXECUTABLE_PREFIX:STRING`
- `HPX_WITH_FAIL_COMPILE_TESTS:BOOL`
- `HPX_WITH_FETCH_ASIO:BOOL`
- `HPX_WITH_FETCH_LCI:BOOL`
- `HPX_WITH_IO_COUNTERS:BOOL`
- `HPX_WITH_LCI_TAG:STRING`
- `HPX_WITH_TESTS:BOOL`
- `HPX_WITH_TESTS_BENCHMARKS:BOOL`
- `HPX_WITH_TESTS_EXAMPLES:BOOL`
- `HPX_WITH_TESTS_EXTERNAL_BUILD:BOOL`
- `HPX_WITH_TESTS_HEADERS:BOOL`
- `HPX_WITH_TESTS_REGRESSIONS:BOOL`
- `HPX_WITH_TESTS_UNIT:BOOL`
- `HPX_WITH_TOOLS:BOOL`

`HPX_WITH_ASIO_TAG:STRING`

Asio repository tag or branch

`HPX_WITH_COMPILE_ONLY_TESTS:BOOL`

Create build system support for compile time only HPX tests (default ON)

`HPX_WITH_DISTRIBUTED_RUNTIME:BOOL`

Enable the distributed runtime (default: ON). Turning off the distributed runtime completely disallows the creation and use of components and actions. Turning this option off is experimental!

`HPX_WITH_DOCUMENTATION:BOOL`

Build the HPX documentation (default OFF).

`HPX_WITH_DOCUMENTATION_OUTPUT_FORMATS:STRING`

List of documentation output formats to generate. Valid options are html;singlehtml;latexpdf;man. Multiple values can be separated with semicolons. (default html).

HPX_WITH_EXAMPLES:BOOL

Build the HPX examples (default ON)

HPX_WITH_EXAMPLES_HDF5:BOOL

Enable examples requiring HDF5 support (default: OFF).

HPX_WITH_EXAMPLES_OPENMP:BOOL

Enable examples requiring OpenMP support (default: OFF).

HPX_WITH_EXAMPLES_QT4:BOOL

Enable examples requiring Qt4 support (default: OFF).

HPX_WITH_EXAMPLES_QTHREADS:BOOL

Enable examples requiring QThreads support (default: OFF).

HPX_WITH_EXAMPLES_TBB:BOOL

Enable examples requiring TBB support (default: OFF).

HPX_WITH_EXECUTABLE_PREFIX:STRING

Executable prefix (default none), ‘**hpx**’ useful for system install.

HPX_WITH_FAIL_COMPILE_TESTS:BOOL

Create build system support for fail compile HPX tests (default ON)

HPX_WITH_FETCH_ASIO:BOOL

Use FetchContent to fetch Asio. By default an installed Asio will be used. (default: OFF)

HPX_WITH_FETCH_LCI:BOOL

Use FetchContent to fetch LCI. By default an installed LCI will be used. (default: OFF)

HPX_WITH_IO_COUNTERS:BOOL

Enable IO counters (default: ON)

HPX_WITH_LCI_TAG:STRING

LCI repository tag or branch

HPX_WITH_TESTS:BOOL

Build the HPX tests (default ON)

HPX_WITH_TESTS_BENCHMARKS:BOOL

Build HPX benchmark tests (default: ON)

HPX_WITH_TESTS_EXAMPLES:BOOL

Add HPX examples as tests (default: ON)

HPX_WITH_TESTS_EXTERNAL_BUILD:BOOL

Build external cmake build tests (default: ON)

HPX_WITH_TESTS_HEADERS:BOOL

Build HPX header tests (default: OFF)

HPX_WITH_TESTS_REGRESSIONS:BOOL

Build HPX regression tests (default: ON)

HPX_WITH_TESTS_UNIT:BOOL

Build HPX unit tests (default: ON)

HPX_WITH_TOOLS:BOOL

Build HPX tools (default: OFF)

Thread Manager options

- `HPX_COROUTINES_WITH_SWAP_CONTEXT_EMULATION:BOOL`
- `HPX_WITH_COROUTINE_COUNTERS:BOOL`
- `HPX_WITH_IO_POOL:BOOL`
- `HPX_WITH_MAX_CPU_COUNT:STRING`
- `HPX_WITH_MAX_NUMA_DOMAIN_COUNT:STRING`
- `HPX_WITH_SCHEDULER_LOCAL_STORAGE:BOOL`
- `HPX_WITH_SPINLOCK_DEADLOCK_DETECTION:BOOL`
- `HPX_WITH_SPINLOCK_POOL_NUM:STRING`
- `HPX_WITH_STACKTRACES:BOOL`
- `HPX_WITH_STACKTRACES_DEMANGLE_SYMBOLS:BOOL`
- `HPX_WITH_STACKTRACES_STATIC_SYMBOLS:BOOL`
- `HPX_WITH_THREAD_BACKTRACE_DEPTH:STRING`
- `HPX_WITH_THREAD_BACKTRACE_ON_SUSPENSION:BOOL`
- `HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES:BOOL`
- `HPX_WITH_THREAD_CUMULATIVE_COUNTS:BOOL`
- `HPX_WITH_THREAD_IDLE_RATES:BOOL`
- `HPX_WITH_THREAD_LOCAL_STORAGE:BOOL`
- `HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF:BOOL`
- `HPX_WITH_THREAD_QUEUE_WAITTIME:BOOL`
- `HPX_WITH_THREAD_STACK_MMAP:BOOL`
- `HPX_WITH_THREAD_STEALING_COUNTS:BOOL`
- `HPX_WITH_THREAD_TARGET_ADDRESS:BOOL`
- `HPX_WITH_TIMER_POOL:BOOL`

`HPX_COROUTINES_WITH_SWAP_CONTEXT_EMULATION:BOOL`

Emulate SwapContext API for coroutines (Windows only, default: OFF)

`HPX_WITH_COROUTINE_COUNTERS:BOOL`

Enable keeping track of coroutine creation and rebinding counts (default: OFF)

`HPX_WITH_IO_POOL:BOOL`

Disable internal IO thread pool, do not change if not absolutely necessary (default: ON)

`HPX_WITH_MAX_CPU_COUNT:STRING`

HPX applications will not use more than this number of OS-Threads (empty string means dynamic) (default: "")

`HPX_WITH_MAX_NUMA_DOMAIN_COUNT:STRING`

HPX applications will not run on machines with more NUMA domains (default: 8)

`HPX_WITH_SCHEDULER_LOCAL_STORAGE:BOOL`

Enable scheduler local storage for all HPX schedulers (default: OFF)

`HPX_WITH_SPINLOCK_DEADLOCK_DETECTION:BOOL`

Enable spinlock deadlock detection (default: OFF)

HPX_WITH_SPINLOCK_POOL_NUM: STRING

Number of elements a spinlock pool manages (default: 128)

HPX_WITH_STACKTRACES: BOOL

Attach backtraces to HPX exceptions (default: ON)

HPX_WITH_STACKTRACES_DEMANGLE_SYMBOLS: BOOL

Thread stack back trace symbols will be demangled (default: ON)

HPX_WITH_STACKTRACES_STATIC_SYMBOLS: BOOL

Thread stack back trace will resolve static symbols (default: OFF)

HPX_WITH_THREAD_BACKTRACE_DEPTH: STRING

Thread stack back trace depth being captured (default: 20)

HPX_WITH_THREAD_BACKTRACE_ON_SUSPENSION: BOOL

Enable thread stack back trace being captured on suspension (default: OFF)

HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES: BOOL

Enable measuring thread creation and cleanup times (default: OFF)

HPX_WITH_THREAD_CUMULATIVE_COUNTS: BOOL

Enable keeping track of cumulative thread counts in the schedulers (default: ON)

HPX_WITH_THREAD_IDLE_RATES: BOOL

Enable measuring the percentage of overhead times spent in the scheduler (default: OFF)

HPX_WITH_THREAD_LOCAL_STORAGE: BOOL

Enable thread local storage for all HPX threads (default: OFF)

HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF: BOOL

HPX scheduler threads do exponential backoff on idle queues (default: ON)

HPX_WITH_THREAD_QUEUE_WAITTIME: BOOL

Enable collecting queue wait times for threads (default: OFF)

HPX_WITH_THREAD_STACK_MMAP: BOOL

Use mmap for stack allocation on appropriate platforms

HPX_WITH_THREAD_STEALING_COUNTS: BOOL

Enable keeping track of counts of thread stealing incidents in the schedulers (default: OFF)

HPX_WITH_THREAD_TARGET_ADDRESS: BOOL

Enable storing target address in thread for NUMA awareness (default: OFF)

HPX_WITH_TIMER_POOL: BOOL

Disable internal timer thread pool, do not change if not absolutely necessary (default: ON)

AGAS options

- **HPX_WITH_AGAS_DUMP_REFCNT_ENTRIES: BOOL**

HPX_WITH_AGAS_DUMP_REFCNT_ENTRIES: BOOL

Enable dumps of the AGAS refcnt tables to logs (default: OFF)

Parcelport options

- `HPX_WITH_NETWORKING:BOOL`
- `HPX_WITH_PARCELPORT_ACTION_COUNTERS:BOOL`
- `HPX_WITH_PARCELPORT_COUNTERS:BOOL`
- `HPX_WITH_PARCELPORT_LCI:BOOL`
- `HPX_WITH_PARCELPORT_LIBFABRIC:BOOL`
- `HPX_WITH_PARCELPORT_MPI:BOOL`
- `HPX_WITH_PARCELPORT_TCP:BOOL`
- `HPX_WITH_PARCEL_PROFILING:BOOL`

`HPX_WITH_NETWORKING:BOOL`

Enable support for networking and multi-node runs (default: ON)

`HPX_WITH_PARCELPORT_ACTION_COUNTERS:BOOL`

Enable performance counters reporting parcelport statistics on a per-action basis.

`HPX_WITH_PARCELPORT_COUNTERS:BOOL`

Enable performance counters reporting parcelport statistics.

`HPX_WITH_PARCELPORT_LCI:BOOL`

Enable the LCI based parcelport.

`HPX_WITH_PARCELPORT_LIBFABRIC:BOOL`

Enable the libfabric based parcelport. This is currently an experimental feature

`HPX_WITH_PARCELPORT_MPI:BOOL`

Enable the MPI based parcelport.

`HPX_WITH_PARCELPORT_TCP:BOOL`

Enable the TCP based parcelport.

`HPX_WITH_PARCEL_PROFILING:BOOL`

Enable profiling data for parcels

Profiling options

- `HPX_WITH_APEX:BOOL`
- `HPX_WITH_ITTNOTIFY:BOOL`
- `HPX_WITH_PAPI:BOOL`

`HPX_WITH_APEX:BOOL`

Enable APEX instrumentation support.

`HPX_WITH_ITTNOTIFY:BOOL`

Enable Amplifier (ITT) instrumentation support.

`HPX_WITH_PAPI:BOOL`

Enable the PAPI based performance counter.

Debugging options

- `HPX_WITH_ATTACH_DEBUGGER_ON_TEST_FAILURE:BOOL`
- `HPX_WITH_PARALLEL_TESTS_BIND_NONE:BOOL`
- `HPX_WITH_SANITIZERS:BOOL`
- `HPX_WITH_TESTS_DEBUG_LOG:BOOL`
- `HPX_WITH_TESTS_DEBUG_LOG_DESTINATION:STRING`
- `HPX_WITH_TESTS_MAX_THREADS_PER_LOCALITY:STRING`
- `HPX_WITH_THREAD_DEBUG_INFO:BOOL`
- `HPX_WITH_THREAD_DESCRIPTION_FULL:BOOL`
- `HPX_WITH_THREAD_GUARD_PAGE:BOOL`
- `HPX_WITH_VALGRIND:BOOL`
- `HPX_WITH_VERIFY_LOCKS:BOOL`
- `HPX_WITH_VERIFY_LOCKS_BACKTRACE:BOOL`

`HPX_WITH_ATTACH_DEBUGGER_ON_TEST_FAILURE:BOOL`

Break the debugger if a test has failed (default: OFF)

`HPX_WITH_PARALLEL_TESTS_BIND_NONE:BOOL`

Pass `-hpx:bind=none` to tests that may run in parallel (`cmake -j` flag) (default: OFF)

`HPX_WITH_SANITIZERS:BOOL`

Configure with sanitizer instrumentation support.

`HPX_WITH_TESTS_DEBUG_LOG:BOOL`

Turn on debug logs (`-hpx:debug-hpx-log`) for tests (default: OFF)

`HPX_WITH_TESTS_DEBUG_LOG_DESTINATION:STRING`

Destination for test debug logs (default: cout)

`HPX_WITH_TESTS_MAX_THREADS_PER_LOCALITY:STRING`

Maximum number of threads to use for tests (default: 0, use the number of threads specified by the test)

`HPX_WITH_THREAD_DEBUG_INFO:BOOL`

Enable thread debugging information (default: OFF, implicitly enabled in debug builds)

`HPX_WITH_THREAD_DESCRIPTION_FULL:BOOL`

Use function address for thread description (default: OFF)

`HPX_WITH_THREAD_GUARD_PAGE:BOOL`

Enable thread guard page (default: ON)

`HPX_WITH_VALGRIND:BOOL`

Enable Valgrind instrumentation support.

`HPX_WITH_VERIFY_LOCKS:BOOL`

Enable lock verification code (default: OFF, enabled in debug builds)

`HPX_WITH_VERIFY_LOCKS_BACKTRACE:BOOL`

Enable thread stack back trace being captured on lock registration (to be used in combination with `HPX_WITH_VERIFY_LOCKS=ON`, default: OFF)

Modules options

- `HPX_DATASTRUCTURES_WITH_ADAPT_STD_TUPLE:BOOL`
- `HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY:BOOL`
- `HPX_ITERATOR_SUPPORT_WITH_BOOST_ITERATOR_TRAVERSAL_TAG_COMPATIBILITY:BOOL`
- `HPX_SERIALIZATION_WITH_ALLOW_CONST_TUPLE_MEMBERS:BOOL`
- `HPX_SERIALIZATION_WITH_ALLOW_RAW_POINTER_SERIALIZATION:BOOL`
- `HPX_SERIALIZATION_WITH_ALL_TYPES_ARE_BITWISE_SERIALIZABLE:BOOL`
- `HPX_SERIALIZATION_WITH_BOOST_TYPES:BOOL`
- `HPX_SERIALIZATION_WITH_SUPPORTS_ENDIANESS:BOOL`
- `HPX_TOPOLOGY_WITH_ADDITIONAL_HWLOC_TESTING:BOOL`

`HPX_DATASTRUCTURES_WITH_ADAPT_STD_TUPLE:BOOL`

Enable compatibility of hpx::tuple with std::tuple. (default: ON)

`HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY:BOOL`

Enable Boost.FileSystem compatibility. (default: OFF)

`HPX_ITERATOR_SUPPORT_WITH_BOOST_ITERATOR_TRAVERSAL_TAG_COMPATIBILITY:BOOL`

Enable Boost.Iterator traversal tag compatibility. (default: OFF)

`HPX_SERIALIZATION_WITH_ALLOW_CONST_TUPLE_MEMBERS:BOOL`

Enable serializing std::tuple with const members. (default: OFF)

`HPX_SERIALIZATION_WITH_ALLOW_RAW_POINTER_SERIALIZATION:BOOL`

Enable serializing raw pointers. (default: OFF)

`HPX_SERIALIZATION_WITH_ALL_TYPES_ARE_BITWISE_SERIALIZABLE:BOOL`

Assume all types are bitwise serializable. (default: OFF)

`HPX_SERIALIZATION_WITH_BOOST_TYPES:BOOL`

Enable serialization of certain Boost types. (default: OFF)

`HPX_SERIALIZATION_WITH_SUPPORTS_ENDIANESS:BOOL`

Support endian conversion on inout and output archives. (default: OFF)

`HPX_TOPOLOGY_WITH_ADDITIONAL_HWLOC_TESTING:BOOL`

Enable HWLOC filtering that makes it report no cores, this is purely an option supporting better testing - do not enable under normal circumstances. (default: OFF)

Additional tools and libraries used by HPX

Here is a list of additional libraries and tools that are either optionally supported by the build system or are optionally required for certain examples or tests. These libraries and tools can be detected by the *HPX* build system.

Each of the tools or libraries listed here will be automatically detected if they are installed in some standard location. If a tool or library is installed in a different location, you can specify its base directory by appending `_ROOT` to the variable name as listed below. For instance, to configure a custom directory for `BOOST`, specify `BOOST_ROOT=/custom/boost/root`.

`BOOST_ROOT:PATH`

Specifies where to look for the Boost installation to be used for compiling *HPX*. Set this if CMake is not able to locate a suitable version of Boost. The directory specified here can be either the root of an installed Boost distribution or the directory where you unpacked and built Boost without installing it (with staged libraries).

HWLOC_ROOT:PATH

Specifies where to look for the hwloc library. Set this if CMake is not able to locate a suitable version of hwloc. Hwloc provides platform- independent support for extracting information about the used hardware architecture (number of cores, number of NUMA domains, hyperthreading, etc.). *HPX* utilizes this information if available.

PAPI_ROOT:PATH

Specifies where to look for the PAPI library. The PAPI library is needed to compile a special component exposing PAPI hardware events and counters as *HPX* performance counters. This is not available on the Windows platform.

AMPLIFIER_ROOT:PATH

Specifies where to look for one of the tools of the Intel Parallel Studio product, either Intel Amplifier or Intel Inspector. This should be set if the CMake variable `HPX_USE_ITT_NOTIFY` is set to ON. Enabling ITT support in *HPX* will integrate any application with the mentioned Intel tools, which customizes the generated information for your application and improves the generated diagnostics.

In addition, some of the examples may need the following variables:

HDF5_ROOT:PATH

Specifies where to look for the Hierarchical Data Format V5 (HDF5) include files and libraries.

2.3.5 Creating *HPX* projects

Using *HPX* with pkg-config

How to build *HPX* applications with pkg-config

After you are done installing *HPX*, you should be able to build the following program. It prints Hello World! on the *locality* you run it on.

```
// Including 'hpx/hpx_main.hpp' instead of the usual 'hpx/hpx_init.hpp' enables
// to use the plain C-main below as the direct main HPX entry point.
#include <hpx/hpx_main.hpp>
#include <hpx/iostream.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << std::flush;
    return 0;
}
```

Copy the text of this program into a file called hello_world.cpp.

Now, in the directory where you put hello_world.cpp, issue the following commands (where `$HPX_LOCATION` is the build directory or `CMAKE_INSTALL_PREFIX` you used while building *HPX*):

```
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HPX_LOCATION/lib/pkgconfig
$ c++ -o hello_world hello_world.cpp \
`pkg-config --cflags --libs hpx_application` \
-lhpx_iostreams -DHPX_APPLICATION_NAME=hello_world
```

Important: When using pkg-config with *HPX*, the pkg-config flags must go after the `-o` flag.

Note: HPX libraries have different names in debug and release mode. If you want to link against a debug HPX library, you need to use the _debug suffix for the pkg-config name. That means instead of hpx_application or hpx_component, you will have to use hpx_application_debug or hpx_component_debug Moreover, all referenced HPX components need to have an appended d suffix. For example, instead of -lhpux_iostreams you will need to specify -lhpux_iostreamsd.

Important: If the HPX libraries are in a path that is not found by the dynamic linker, you will need to add the path \$HPX_LOCATION/lib to your linker search path (for example LD_LIBRARY_PATH on Linux).

To test the program, type:

```
$ ./hello_world
```

which should print Hello World! and exit.

How to build HPX components with pkg-config

Let's try a more complex example involving an HPX component. An HPX component is a class that exposes HPX actions. HPX components are compiled into dynamically loaded modules called component libraries. Here's the source code:

hello_world_component.cpp

```
#include <hpx/config.hpp>
#if !defined(HPX_COMPUTE_DEVICE_CODE)
#include <hpx/iostream.hpp>
#include "hello_world_component.hpp"

#include <iostream>

namespace examples { namespace server {
    void hello_world::invoke()
    {
        hpx::cout << "Hello HPX World!" << std::endl;
    }
}} // namespace examples::server

HDX_REGISTER_COMPONENT_MODULE()

typedef hpx::components::component<examples::server::hello_world>
    hello_world_type;

HDX_REGISTER_COMPONENT(hello_world_type, hello_world)

HDX_REGISTER_ACTION(
    examples::server::hello_world::invoke_action, hello_world_invoke_action)
#endif
```

hello_world_component.hpp

```
#pragma once

#include <hpx/config.hpp>
```

(continues on next page)

(continued from previous page)

```
#if !defined(HPX_COMPUTE_DEVICE_CODE)
#include <hpx/hpx.hpp>
#include <hpx/include/actions.hpp>
#include <hpx/include/components.hpp>
#include <hpx/include/lcos.hpp>
#include <hpx/serialization.hpp>

#include <utility>

namespace examples { namespace server {
    struct HPX_COMPONENT_EXPORT hello_world
        : hpx::components::component_base<hello_world>
    {
        void invoke();
        HPX_DEFINE_COMPONENT_ACTION(hello_world, invoke)
    };
}} // namespace examples::server

HPX_REGISTER_ACTION_DECLARATION(
    examples::server::hello_world::invoke_action, hello_world_invoke_action)

namespace examples {
    struct hello_world
        : hpx::components::client_base<hello_world, server::hello_world>
    {
        typedef hpx::components::client_base<hello_world, server::hello_world>
            base_type;

        hello_world(hpx::future<hpx::id_type>&& f)
            : base_type(std::move(f))
        {
        }

        hello_world(hpx::id_type&& f)
            : base_type(std::move(f))
        {
        }

        void invoke()
        {
            hpx::async<server::hello_world::invoke_action>(<this>->get_id())
                .get();
        }
    };
} // namespace examples

#endif
```

hello_world_client.cpp

```
#include <hpx/config.hpp>
#if defined(HPX_COMPUTE_HOST_CODE)
#include <hpx/wrap_main.hpp>

#include "hello_world_component.hpp"

int main()
```

(continues on next page)

(continued from previous page)

```
{
{
    // Create a single instance of the component on this locality.
    examples::hello_world client =
        hpx::new<examples::hello_world>(hpx::find_here());

    // Invoke the component's action, which will print "Hello World!".
    client.invoke();
}

return 0;
}
#endif
```

Copy the three source files above into three files (called `hello_world_component.cpp`, `hello_world_component.hpp` and `hello_world_client.cpp`, respectively).

Now, in the directory where you put the files, run the following command to build the component library. (where `$HPX_LOCATION` is the build directory or `CMAKE_INSTALL_PREFIX` you used while building `HPX`):

```
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HPX_LOCATION/lib/pkgconfig
$ c++ -o libhpx_hello_world.so hello_world_component.cpp \
`pkg-config --cflags --libs hpx_component` \
-lhpx_iostreams -DHPX_COMPONENT_NAME=hpx_hello_world
```

Now pick a directory in which to install your `HPX` component libraries. For this example, we'll choose a directory named `my_hpx_libs`:

```
$ mkdir ~/my_hpx_libs
$ mv libhpx_hello_world.so ~/my_hpx_libs
```

Note: `HPX` libraries have different names in debug and release mode. If you want to link against a debug `HPX` library, you need to use the `_debug` suffix for the `pkg-config` name. That means instead of `hpx_application` or `hpx_component` you will have to use `hpx_application_debug` or `hpx_component_debug`. Moreover, all referenced `HPX` components need to have a appended `d` suffix, e.g. instead of `-lhpx_iostreams` you will need to specify `-lhpx_iostreamsd`.

Important: If the `HPX` libraries are in a path that is not found by the dynamic linker. You need to add the path `$HPX_LOCATION/lib` to your linker search path (for example `LD_LIBRARY_PATH` on Linux).

Now, to build the application that uses this component (`hello_world_client.cpp`), we do:

```
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HPX_LOCATION/lib/pkgconfig
$ c++ -o hello_world_client hello_world_client.cpp \
`pkg-config --cflags --libs hpx_application` \
-L${HOME}/my_hpx_libs -lhpx_hello_world -lhpx_iostreams
```

Important: When using `pkg-config` with `HPX`, the `pkg-config` flags must go after the `-o` flag.

Finally, you'll need to set your `LD_LIBRARY_PATH` before you can run the program. To run the program, type:

```
$ export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$HOME/my_hpx_libs"
$ ./hello_world_client
```

which should print Hello HPX World! and exit.

Using HPX with CMake-based projects

In addition to the pkg-config support discussed on the previous pages, *HPX* comes with full CMake support. In order to integrate *HPX* into existing or new CMakeLists.txt, you can leverage the `find_package`³⁶ command integrated into CMake. Following, is a Hello World component example using CMake.

Let's revisit what we have. We have three files that compose our example application:

- `hello_world_component.hpp`
- `hello_world_component.cpp`
- `hello_world_client.hpp`

The basic structure to include *HPX* into your CMakeLists.txt is shown here:

```
# Require a recent version of cmake
cmake_minimum_required(VERSION 3.18 FATAL_ERROR)

# This project is C++ based.
project(your_app CXX)

# Instruct cmake to find the HPX settings
find_package(HPX)
```

In order to have CMake find *HPX*, it needs to be told where to look for the `HPXConfig.cmake` file that is generated when *HPX* is built or installed. It is used by `find_package(HPX)` to set up all the necessary macros needed to use *HPX* in your project. The ways to achieve this are:

- Set the `HPX_DIR` CMake variable to point to the directory containing the `HPXConfig.cmake` script on the command line when you invoke CMake:

```
$ cmake -DHPX_DIR=$HPX_LOCATION/lib/cmake/HPX ...
```

where `$HPX_LOCATION` is the build directory or `CMAKE_INSTALL_PREFIX` you used when building/configuring *HPX*.

- Set the `CMAKE_PREFIX_PATH` variable to the root directory of your *HPX* build or install location on the command line when you invoke CMake:

```
$ cmake -DCMAKE_PREFIX_PATH=$HPX_LOCATION ...
```

The difference between `CMAKE_PREFIX_PATH` and `HPX_DIR` is that CMake will add common postfixes, such as `lib/cmake/<project>`, to the `CMAKE_PREFIX_PATH` and search in these locations too. Note that if your project uses *HPX* as well as other CMake-managed projects, the paths to the locations of these multiple projects may be concatenated in the `CMAKE_PREFIX_PATH`.

- The variables above may be set in the CMake GUI or curses `ccmake` interface instead of the command line.

Additionally, if you wish to require *HPX* for your project, replace the `find_package(HPX)` line with `find_package(HPX REQUIRED)`.

You can check if *HPX* was successfully found with the `HPX_FOUND` CMake variable.

³⁶ https://www.cmake.org/cmake/help/latest/command/find_package.html

Using CMake targets

The recommended way of setting up your targets to use *HPX* is to link to the `HPX::hpx` CMake target:

```
target_link_libraries(hello_world_component PUBLIC HPX::hpx)
```

This requires that you have already created the target like this:

```
add_library(hello_world_component SHARED hello_world_component.cpp)
target_include_directories(hello_world_component PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})
```

When you link your library to the `HPX::hpx` CMake target, you will be able use *HPX* functionality in your library. To use `main()` as the implicit entry point in your application you must additionally link your application to the CMake target `HPX::wrap_main`. This target is automatically linked to executables if you are using the macros described below ([Using macros to create new targets](#)). See [Re-use the `main\(\)` function as the main *HPX* entry point](#) for more information on implicitly using `main()` as the entry point.

Creating a component requires setting two additional compile definitions:

```
target_compile_options(hello_world_component
    HPX_COMPONENT_NAME=hello_world
    HPX_COMPONENT_EXPORTS)
```

Instead of setting these definitions manually you may link to the `HPX::component` target, which sets `HPX_COMPONENT_NAME` to `hpx_<target_name>`, where `<target_name>` is the target name of your library. Note that these definitions should be `PRIVATE` to make sure these definitions are not propagated transitively to dependent targets.

In addition to making your library a component you can make it a plugin. To do so link to the `HPX::plugin` target. Similarly to `HPX::component` this will set `HPX_PLUGIN_NAME` to `hpx_<target_name>`. This definition should also be `PRIVATE`. Unlike regular shared libraries, plugins are loaded at runtime from certain directories and will not be found without additional configuration. Plugins should be installed into a directory containing only plugins. For example, the plugins created by *HPX* itself are installed into the `hpx` subdirectory in the library install directory (typically `lib` or `lib64`). When using the `HPX::plugin` target you need to install your plugins into an appropriate directory. You may also want to set the location of your plugin in the build directory with the `*_OUTPUT_DIRECTORY*` CMake target properties to be able to load the plugins in the build directory. Once you've set the install or output directory of your plugin you need to tell your executable where to find it at runtime. You can do this either by setting the environment variable `HPX_COMPONENT_PATHS` or the ini setting `hpx.component_paths` (see `--hpx:ini`) to the directory containing your plugin.

Using macros to create new targets

In addition to the targets described above, *HPX* provides convenience macros to hide optional boilerplate code that may be useful for your project. The link to the targets described above. We recommend that you use the targets directly whenever possible as they tend to compose better with other targets.

The macro for adding an *HPX* component is `add_hpx_component`. It can be used in your `CMakeLists.txt` file like this:

```
# build your application using HPX
add_hpx_component(hello_world
    SOURCES hello_world_component.cpp
    HEADERS hello_world_component.hpp
    COMPONENT_DEPENDENCIES iostreams)
```

Note: `add_hpx_component` adds a `_component` suffix to the target name. In the example above, a `hello_world_component` target will be created.

The available options to `add_hpx_component` are:

- `SOURCES`: The source files for that component
- `HEADERS`: The header files for that component
- `DEPENDENCIES`: Other libraries or targets this component depends on
- `COMPONENT_DEPENDENCIES`: The components this component depends on
- `PLUGIN`: Treats this component as a plugin-able library
- `COMPILE_FLAGS`: Additional compiler flags
- `LINK_FLAGS`: Additional linker flags
- `FOLDER`: Adds the headers and source files to this Source Group folder
- `EXCLUDE_FROM_ALL`: Do not build this component as part of the `all` target

After adding the component, the way you add the executable is as follows:

```
# build your application using HPX
add_hpx_executable(hello_world
    SOURCES hello_world_client.cpp
    COMPONENT_DEPENDENCIES hello_world)
```

Note: `add_hpx_executable` automatically adds a `_component` suffix to dependencies specified in `COMPONENT_DEPENDENCIES`, meaning you can directly use the name given when adding a component using `add_hpx_component`.

When you configure your application, all you need to do is set the `HPX_DIR` variable to point to the installation of `HPX`.

Note: All library targets built with `HPX` are exported and readily available to be used as arguments to `target_link_libraries`³⁷ in your targets. The `HPX` include directories are available with the `HPX_INCLUDE_DIRS` CMake variable.

Using the `HPX` compiler wrapper `hpxcxx`

The `hpxcxx` compiler wrapper helps to compile a `HPX` component, application, or object file, based on the arguments passed to it.

```
$ hpxcxx [ --exe=<APPLICATION_NAME> | --comp=<COMPONENT_NAME> | -c] FLAGS FILES
```

The `hpxcxx` command **requires** that either an application or a component is built or `-c` flag is specified. If the build is against a debug build, the `-g` is to be specified while building.

³⁷ https://www.cmake.org/cmake/help/latest/command/target_link_libraries.html

Optional **FLAGS**

- `-l <LIBRARY> | -L<LIBRARY>`: Links `<LIBRARY>` to the build
- `-g`: Specifies that the application or component build is against a debug build
- `-rd`: Sets `release-with-debug-info` option
- `-mr`: Sets `minsize-release` option

All other flags (like `-o OUTPUT_FILE`) are directly passed to the underlying C++ compiler.

Using macros to set up existing targets to use **HPX**

In addition to the `add_hpx_component` and `add_hpx_executable`, you can use the `hpx_setup_target` macro to have an already existing target to be used with the *HPX* libraries:

```
hpx_setup_target(target)
```

Optional parameters are:

- `EXPORT`: Adds it to the CMake export list `HPXTargets`
- `INSTALL`: Generates an install rule for the target
- `PLUGIN`: Treats this component as a plugin-able library
- `TYPE`: The type can be: `EXECUTABLE`, `LIBRARY` or `COMPONENT`
- `DEPENDENCIES`: Other libraries or targets this component depends on
- `COMPONENT_DEPENDENCIES`: The components this component depends on
- `COMPILE_FLAGS`: Additional compiler flags
- `LINK_FLAGS`: Additional linker flags

If you do not use CMake, you can still build against *HPX*, but you should refer to the section on *How to build HPX components with pkg-config*.

Note: Since *HPX* relies on dynamic libraries, the dynamic linker needs to know where to look for them. If *HPX* isn't installed into a path that is configured as a linker search path, external projects need to either set `RPATH` or adapt `LD_LIBRARY_PATH` to point to where the *HPX* libraries reside. In order to set `RPATH`s, you can include `HPX_SetFullRPATH` in your project after all libraries you want to link against have been added. Please also consult the CMake documentation [here](#)³⁸.

Using **HPX** with Makefile

A basic project building with *HPX* is through creating makefiles. The process of creating one can get complex depending upon the use of `cmake` parameter `HPX_WITH_HPX_MAIN` (which defaults to `ON`).

³⁸ <https://gitlab.kitware.com/cmake/community/wikis/doc/cmake/RPATH-handling>

How to build HPX applications with makefile

If *HPX* is installed correctly, you should be able to build and run a simple Hello World program. It prints Hello World! on the *locality* you run it on.

```
// Including 'hpx/hpx_main.hpp' instead of the usual 'hpx/hpx_init.hpp' enables
// to use the plain C-main below as the direct main HPX entry point.
#include <hpx/hpx_main.hpp>
#include <iostream.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << std::flush;
    return 0;
}
```

Copy the content of this program into a file called `hello_world.cpp`.

Now, in the directory where you put `hello_world.cpp`, create a Makefile. Add the following code:

```
CXX=(CXX) # Add your favourite compiler here or let makefile choose default.

CXXFLAGS=-O3 -std=c++17

BOOST_ROOT=/path/to/boost
HWLOC_ROOT=/path/to/hwloc
TCMALLOC_ROOT=/path/to/tcmalloc
HPX_ROOT=/path/to/hpx

INCLUDE_DIRECTIVES=$(HPX_ROOT)/include $(BOOST_ROOT)/include $(HWLOC_ROOT)/include

LIBRARY_DIRECTIVES=-L$(HPX_ROOT)/lib $(HPX_ROOT)/lib/libhpx_init.a $(HPX_ROOT)/lib/
    libhpx.so $(BOOST_ROOT)/lib/libboost_atomic-mt.so $(BOOST_ROOT)/lib/libboost_
    filesystem-mt.so $(BOOST_ROOT)/lib/libboost_program_options-mt.so $(BOOST_ROOT)/lib/
    libboost_regex-mt.so $(BOOST_ROOT)/lib/libboost_system-mt.so -lpthread $(TCMALLOC_
    ROOT)/libtcmalloc_minimal.so $(HWLOC_ROOT)/libhwloc.so -ldl -lrt

LINK_FLAGS=$(HPX_ROOT)/lib/libhpx_wrap.a -Wl,-wrap=main # should be left empty for_
    HPX_WITH_HPX_MAIN=OFF

hello_world: hello_world.o
    $(CXX) $(CXXFLAGS) -o hello_world hello_world.o $(LIBRARY_DIRECTIVES) $(LINK_FLAGS)

hello_world.o:
    $(CXX) $(CXXFLAGS) -c -o hello_world.o hello_world.cpp $(INCLUDE_DIRECTIVES)
```

Important: `LINK_FLAGS` should be left empty if `HPX_WITH_HPX_MAIN` is set to OFF. Boost in the above example is build with `--layout=tagged`. Actual Boost flags may vary on your build of Boost.

To build the program, type:

```
$ make
```

A successful build should result in `hello_world` binary. To test, type:

```
$ ./hello_world
```

How to build HPX components with makefile

Let's try a more complex example involving an *HPX* component. An *HPX* component is a class that exposes *HPX* actions. *HPX* components are compiled into dynamically-loaded modules called component libraries. Here's the source code:

hello_world_component.cpp

```
#include <hpx/config.hpp>
#ifndef HPX_COMPUTE_DEVICE_CODE
#include <hpx/iostream.hpp>
#include "hello_world_component.hpp"

#include <iostream>

namespace examples { namespace server {
    void hello_world::invoke()
    {
        hpx::cout << "Hello HPX World!" << std::endl;
    }
}} // namespace examples::server

Hpx_REGISTER_COMPONENT_MODULE()

typedef hpx::components::component<examples::server::hello_world>
    hello_world_type;

Hpx_REGISTER_COMPONENT(hello_world_type, hello_world)

Hpx_REGISTER_ACTION(
    examples::server::hello_world::invoke_action, hello_world_invoke_action)
#endif
```

hello_world_component.hpp

```
#pragma once

#include <hpx/config.hpp>
#ifndef HPX_COMPUTE_DEVICE_CODE
#include <hpx/hpx.hpp>
#include <hpx/include/actions.hpp>
#include <hpx/include/components.hpp>
#include <hpx/include/lcos.hpp>
#include <hpx/serialization.hpp>

#include <utility>

namespace examples { namespace server {
    struct HPX_COMPONENT_EXPORT hello_world
        : hpx::components::component_base<hello_world>
    {
        void invoke();
        HPX_DEFINE_COMPONENT_ACTION(hello_world, invoke)
    };
}}
```

(continues on next page)

(continued from previous page)

```

} } // namespace examples::server

HPX_REGISTER_ACTION_DECLARATION(
    examples::server::hello_world::invoke_action, hello_world_invoke_action)

namespace examples {
    struct hello_world
        : hpx::components::client_base<hello_world, server::hello_world>
    {
        typedef hpx::components::client_base<hello_world, server::hello_world>
            base_type;

        hello_world(hpx::future<hpx::id_type>&& f)
            : base_type(std::move(f))
        {
        }

        hello_world(hpx::id_type&& f)
            : base_type(std::move(f))
        {
        }

        void invoke()
        {
            hpx::async<server::hello_world::invoke_action>(<b>this>->get_id())
                .get();
        }
    };
} // namespace examples

#endif

```

hello_world_client.cpp

```

#include <hpx/config.hpp>
#if defined(HPX_COMPUTE_HOST_CODE)
#include <hpx/wrap_main.hpp>

#include "hello_world_component.hpp"

int main()
{
{
    // Create a single instance of the component on this locality.
    examples::hello_world client =
        hpx::new_<examples::hello_world>(hpx::find_here());

    // Invoke the component's action, which will print "Hello World!".
    client.invoke();
}

return 0;
}
#endif

```

Now, in the directory, create a Makefile. Add the following code:

```
CXX=(CXX) # Add your favourite compiler here or let makefile choose default.

CXXFLAGS=-O3 -std=c++17

BOOST_ROOT=/path/to/boost
HWLOC_ROOT=/path/to/hwloc
TCMALLOC_ROOT=/path/to/tcmalloc
HPX_ROOT=/path/to/hpx

INCLUDE_DIRECTIVES=$(HPX_ROOT)/include $(BOOST_ROOT)/include $(HWLOC_ROOT)/include

LIBRARY_DIRECTIVES=-L$(HPX_ROOT)/lib $(HPX_ROOT)/lib/libhpx_init.a $(HPX_ROOT)/lib/
→ libhpx.so $(BOOST_ROOT)/lib/libboost_atomic-mt.so $(BOOST_ROOT)/lib/libboost_
→ filesystem-mt.so $(BOOST_ROOT)/lib/libboost_program_options-mt.so $(BOOST_ROOT)/lib/
→ libboost_regex-mt.so $(BOOST_ROOT)/lib/libboost_system-mt.so -lpthread $(TCMALLOC_
→ ROOT)/libtcmalloc_minimal.so $(HWLOC_ROOT)/libhwloc.so -ldl -lrt

LINK_FLAGS=$(HPX_ROOT)/lib/libhpx_wrap.a -Wl,-wrap=main # should be left empty for_
→ HPX_WITH_HPX_MAIN=OFF

hello_world_client: libhpx_hello_world hello_world_client.o
$(CXX) $(CXXFLAGS) -o hello_world_client $(LIBRARY_DIRECTIVES) libhpx_hello_world
→ $(LINK_FLAGS)

hello_world_client.o: hello_world_client.cpp
$(CXX) $(CXXFLAGS) -o hello_world_client.o hello_world_client.cpp $(INCLUDE_
→ DIRECTIVES)

libhpx_hello_world: hello_world_component.o
$(CXX) $(CXXFLAGS) -o libhpx_hello_world hello_world_component.o $(LIBRARY_
→ DIRECTIVES)

hello_world_component.o: hello_world_component.cpp
$(CXX) $(CXXFLAGS) -c -o hello_world_component.o hello_world_component.cpp
→ $(INCLUDE_DIRECTIVES)
```

To build the program, type:

```
$ make
```

A successful build should result in `hello_world` binary. To test, type:

```
$ ./hello_world
```

Note: Due to high variations in CMake flags and library dependencies, it is recommended to build *HPX* applications and components with `pkg-config` or `CMakeLists.txt`. Writing `Makefile` may result in broken builds if due care is not taken. `pkg-config` files and CMake systems are configured with CMake build of *HPX*. Hence, they are stable when used together and provide better support overall.

2.3.6 Starting the HPX runtime

In order to write an application that uses services from the *HPX* runtime system, you need to initialize the *HPX* library by inserting certain calls into the code of your application. Depending on your use case, this can be done in 3 different ways:

- *Minimally invasive*: Re-use the `main()` function as the main *HPX* entry point.
- *Balanced use case*: Supply your own main *HPX* entry point while blocking the main thread.
- *Most flexibility*: Supply your own main *HPX* entry point while avoiding blocking the main thread.
- *Suspend and resume*: As above but suspend and resume the *HPX* runtime to allow for other runtimes to be used.

Re-use the `main()` function as the main *HPX* entry point

This method is the least intrusive to your code. However, it provides you with the smallest flexibility in terms of initializing the *HPX* runtime system. The following code snippet shows what a minimal *HPX* application using this technique looks like:

```
#include <hpx/hpx_main.hpp>

int main(int argc, char* argv[])
{
    return 0;
}
```

The only change to your code you have to make is to include the file `hpx/hpx_main.hpp`. In this case the function `main()` will be invoked as the first *HPX* thread of the application. The runtime system will be initialized behind the scenes before the function `main()` is executed and will automatically stop after `main()` has returned. For this method to work you must link your application to the CMake target `HPX::wrap_main`. This is done automatically if you are using the provided macros ([Using macros to create new targets](#)) to set up your application, but must be done explicitly if you are using targets directly ([Using CMake targets](#)). All *HPX* API functions can be used from within the `main()` function now.

Note: The function `main()` does not need to expect receiving `argc` and `argv` as shown above, but could expose the signature `int main()`. This is consistent with the usually allowed prototypes for the function `main()` in C++ applications.

All command line arguments specific to *HPX* will still be processed by the *HPX* runtime system as usual. However, those command line options will be removed from the list of values passed to `argc/argv` of the function `main()`. The list of values passed to `main()` will hold only the commandline options that are not recognized by the *HPX* runtime system (see the section [HPX Command Line Options](#) for more details on what options are recognized by *HPX*).

Note: In this mode all one-letter shortcuts that are normally available on the *HPX* command line are disabled (such as `-t` or `-l` see [HPX Command Line Options](#)). This is done to minimize any possible interaction between the command line options recognized by the *HPX* runtime system and any command line options defined by the application.

The value returned from the function `main()` as shown above will be returned to the operating system as usual.

Important: To achieve this seamless integration, the header file `hpx/hpx_main.hpp` defines a macro:

```
#define main hpx_startup::user_main
```

which could result in unexpected behavior.

Important: To achieve this seamless integration, we use different implementations for different operating systems. In case of Linux or macOS, the code present in `hpx_wrap.cpp` is put into action. We hook into the system function in case of Linux and provide alternate entry point in case of macOS. For other operating systems we rely on a macro:

```
#define main hpx_startup::user_main
```

provided in the header file `hpx/hpx_main.hpp`. This implementation can result in unexpected behavior.

Caution: We make use of an *override* variable `include_libhpx_wrap` in the header file `hpx/hpx_main.hpp` to swiftly choose the function call stack at runtime. Therefore, the header file should *only* be included in the main executable. Including it in the components will result in multiple definition of the variable.

Supply your own main *HPX* entry point while blocking the main thread

With this method you need to provide an explicit main-thread function named `hpx_main` at global scope. This function will be invoked as the main entry point of your *HPX* application on the console *locality* only (this function will be invoked as the first *HPX* thread of your application). All *HPX* API functions can be used from within this function.

The thread executing the function `hpx::init` will block waiting for the runtime system to exit. The value returned from `hpx_main` will be returned from `hpx::init` after the runtime system has stopped.

The function `hpx::finalize` has to be called on one of the *HPX* localities in order to signal that all work has been scheduled and the runtime system should be stopped after the scheduled work has been executed.

This method of invoking *HPX* has the advantage of the user being able to decide which version of `hpx::init` to call. This allows to pass additional configuration parameters while initializing the *HPX* runtime system.

```
#include <hpx/hpx_init.hpp>

int hpx_main(int argc, char* argv[])
{
    // Any HPX application logic goes here...
    return hpx::finalize();
}

int main(int argc, char* argv[])
{
    // Initialize HPX, run hpx_main as the first HPX thread, and
    // wait for hpx::finalize being called.
    return hpx::init(argc, argv);
}
```

Note: The function `hpx_main` does not need to expect receiving `argc/argv` as shown above, but could expose one of the following signatures:

```
int hpx_main();
int hpx_main(int argc, char* argv[]);
int hpx_main(hpx::program_options::variables_map& vm);
```

This is consistent with (and extends) the usually allowed prototypes for the function `main()` in C++ applications.

The header file to include for this method of using *HPX* is `hpx/hpx_init.hpp`.

There are many additional overloads of `hpx::init` available, such as the ability to provide your own entry-point function instead of `hpx_main`. Please refer to the function documentation for more details (see: `hpx/hpx_init.hpp`).

Supply your own main *HPX* entry point while avoiding blocking the main thread

With this method you need to provide an explicit main thread function named `hpx_main` at global scope. This function will be invoked as the main entry point of your *HPX* application on the console `locality` only (this function will be invoked as the first *HPX* thread of your application). All *HPX* API functions can be used from within this function.

The thread executing the function `hpx::start` will *not* block waiting for the runtime system to exit, but will return immediately. The function `hpx::finalize` has to be called on one of the *HPX* localities in order to signal that all work has been scheduled and the runtime system should be stopped after the scheduled work has been executed.

This method of invoking *HPX* is useful for applications where the main thread is used for special operations, such as GUIs. The function `hpx::stop` can be used to wait for the *HPX* runtime system to exit and should at least be used as the last function called in `main()`. The value returned from `hpx_main` will be returned from `hpx::stop` after the runtime system has stopped.

```
#include <hpx/hpx_start.hpp>

int hpx_main(int argc, char* argv[])
{
    // Any HPX application logic goes here...
    return hpx::finalize();
}

int main(int argc, char* argv[])
{
    // Initialize HPX, run hpx_main.
    hpx::start(argc, argv);

    // ...Execute other code here...

    // Wait for hpx::finalize being called.
    return hpx::stop();
}
```

Note: The function `hpx_main` does not need to expect receiving `argc/argv` as shown above, but could expose one of the following signatures:

```
int hpx_main();
int hpx_main(int argc, char* argv[]);
int hpx_main(hpx::program_options::variables_map& vm);
```

This is consistent with (and extends) the usually allowed prototypes for the function `main()` in C++ applications.

The header file to include for this method of using *HPX* is `hpx/hpx_start.hpp`.

There are many additional overloads of `hpx::start` available, such as the option for users to provide their own entry point function instead of `hpx_main`. Please refer to the function documentation for more details (see: `hpx/hpx_start.hpp`).

Suspending and resuming the *HPX* runtime

In some applications it is required to combine *HPX* with other runtimes. To support this use case, *HPX* provides two functions: `hpx::suspend` and `hpx::resume`. `hpx::suspend` is a blocking call which will wait for all scheduled tasks to finish executing and then put the thread pool OS threads to sleep. `hpx::resume` simply wakes up the sleeping threads so that they are ready to accept new work. `hpx::suspend` and `hpx::resume` can be found in the header `hpx/hpx_suspend.hpp`.

```
#include <hpx/hpx_start.hpp>
#include <hpx/hpx_suspend.hpp>

int main(int argc, char* argv[])
{
    // Initialize HPX, don't run hpx_main
    hpx::start(nullptr, argc, argv);

    // Schedule a function on the HPX runtime
    hpx::apply(&my_function, ...);

    // Wait for all tasks to finish, and suspend the HPX runtime
    hpx::suspend();

    // Execute non-HPX code here

    // Resume the HPX runtime
    hpx::resume();

    // Schedule more work on the HPX runtime

    // hpx::finalize has to be called from the HPX runtime before hpx::stop
    hpx::apply([]() { hpx::finalize(); });
    return hpx::stop();
}
```

Note: `hpx::suspend` does not wait for `hpx::finalize` to be called. Only call `hpx::finalize` when you wish to fully stop the *HPX* runtime.

Warning:

`hpx::suspend` only waits for local tasks, i.e. tasks on the current locality, to finish executing. When using `hpx::suspend` in a multi-locality scenario the user is responsible for ensuring that any work required from other localities has also finished.

HPX also supports suspending individual thread pools and threads. For details on how to do that, see the documentation for `hpx::threads::thread_pool_base`.

Automatically suspending worker threads

The previous method guarantees that the worker threads are suspended when you ask for it and that they stay suspended. An alternative way to achieve the same effect is to tweak how quickly HPX suspends its worker threads when they run out of work. The following configuration values make sure that HPX idles very quickly:

```
hpx.max_idle_backoff_time = 1000  
hpx.max_idle_loop_count = 0
```

They can be set on the command line using `--hpx:ini=hpx.max_idle_backoff_time=1000` and `--hpx:ini=hpx.max_idle_loop_count=0`. See [Launching and configuring HPX applications](#) for more details on how to set configuration parameters.

After setting idling parameters the previous example could now be written like this instead:

```
#include <hpx/hpx_start.hpp>  
  
int main(int argc, char* argv[])
{
    // Initialize HPX, don't run hpx_main
    hpx::start(nullptr, argc, argv);

    // Schedule some functions on the HPX runtime
    // NOTE: run_as_hpx_thread blocks until completion.
    hpx::run_as_hpx_thread(&my_function, ...);
    hpx::run_as_hpx_thread(&my_other_function, ...);

    // hpx::finalize has to be called from the HPX runtime before hpx::stop
    hpx::apply([]() { hpx::finalize(); });
    return hpx::stop();
}
```

In this example each call to `hpx::run_as_hpx_thread` acts as a “parallel region”.

Working of `hpx_main.hpp`

In order to initialize HPX from `main()`, we make use of linker tricks.

It is implemented differently for different operating systems. The method of implementation is as follows:

- *Linux*: Using linker `--wrap` option.
- *Mac OSX*: Using the linker `-e` option.
- *Windows*: Using `#define main hpx_startup::user_main`

Linux implementation

We make use of the Linux linker `ld`'s `--wrap` option to wrap the `main()` function. This way any calls to `main()` are redirected to our own implementation of `main`. It is here that we check for the existence of `hpx_main.hpp` by making use of a shadow variable `include_libhpx_wrap`. The value of this variable determines the function stack at runtime.

The implementation can be found in `libhpx_wrap.a`.

Important: It is necessary that `hpx_main.hpp` be not included more than once. Multiple inclusions can result in multiple definition of `include_libhpx_wrap`.

Mac OSX implementation

Here we make use of yet another linker option `-e` to change the entry point to our custom entry function `initialize_main`. We initialize the *HPX* runtime system from this function and call `main` from the initialized system. We determine the function stack at runtime by making use of the shadow variable `include_libhpx_wrap`.

The implementation can be found in `libhpx_wrap.a`.

Important: It is necessary that `hpx_main.hpp` be not included more than once. Multiple inclusions can result in multiple definition of `include_libhpx_wrap`.

Windows implementation

We make use of a macro `#define main hpx_startup::user_main` to take care of the initializations.

This implementation could result in unexpected behaviors.

2.3.7 Launching and configuring *HPX* applications

Configuring *HPX* applications

All *HPX* applications can be configured using special command line options and/or using special configuration files. This section describes the available options, the configuration file format, and the algorithm used to locate possible predefined configuration files. Additionally, this section describes the defaults assumed if no external configuration information is supplied.

During startup any *HPX* application applies a predefined search pattern to locate one or more configuration files. All found files will be read and merged in the sequence they are found into one single internal database holding all configuration properties. This database is used during the execution of the application to configure different aspects of the runtime system.

In addition to the ini files, any application can supply its own configuration files, which will be merged with the configuration database as well. Moreover, the user can specify additional configuration parameters on the command line when executing an application. The *HPX* runtime system will merge all command line configuration options (see the description of the `--hpx:ini`, `--hpx:config`, and `--hpx:app-config` command line options).

The HPX ini file format

All *HPX* applications can be configured using a special file format that is similar to the well-known [Windows INI file format](#)³⁹. This is a structured text format that allows users to group key/value pairs (properties) into sections. The basic element contained in an ini file is the property. Every property has a name and a value, delimited by an equal sign '='. The name appears to the left of the equal sign:

```
name=value
```

The value may contain equal signs as only the first '=' character is interpreted as the delimiter between name and value. Whitespace before the name, after the value and immediately before and after the delimiting equal sign is ignored. Whitespace inside the value is retained.

Properties may be grouped into arbitrarily named sections. The section name appears on a line by itself, in square brackets. All properties after the section declaration are associated with that section. There is no explicit “end of section” delimiter; sections end at the next section declaration or the end of the file:

```
[section]
```

In *HPX* sections can be nested. A nested section has a name composed of all section names it is embedded in. The section names are concatenated using a dot '.'. :

```
[outer_section.inner_section]
```

Here, `inner_section` is logically nested within `outer_section`.

It is possible to use the full section name concatenated with the property name to refer to a particular property. For example, in:

```
[a.b.c]
d = e
```

the property value of `d` can be referred to as `a.b.c.d=e`.

In *HPX* ini files can contain comments. Hash signs '#' at the beginning of a line indicate a comment. All characters starting with '#' until the end of the line are ignored.

If a property with the same name is reused inside a section, the second occurrence of this property name will override the first occurrence (discard the first value). Duplicate sections simply merge their properties together, as if they occurred contiguously.

In *HPX* ini files a property value `${FOO:default}` will use the environmental variable `FOO` to extract the actual value if it is set and `default` otherwise. No default has to be specified. Therefore, `${FOO}` refers to the environmental variable `FOO`. If `FOO` is not set or empty, the overall expression will evaluate to an empty string. A property value `${[section.key]:default}` refers to the value held by the property `section.key` if it exists and `default` otherwise. No default has to be specified. Therefore `${[section.key]}` refers to the property `section.key`. If the property `section.key` is not set or empty, the overall expression will evaluate to an empty string.

Note: Any property `${[section.key]:default}` is evaluated whenever it is queried and not when the configuration data is initialized. This allows for lazy evaluation and relaxes initialization order of different sections. The only exception are recursive property values, e.g., values referring to the very key they are associated with. Those property values are evaluated at initialization time to avoid infinite recursion.

³⁹ https://en.wikipedia.org/wiki/INI_file

Built-in default configuration settings

During startup any *HPX* application applies a predefined search pattern to locate one or more configuration files. All found files will be read and merged in the sequence they are found into one single internal data structure holding all configuration properties.

As a first step the internal configuration database is filled with a set of default configuration properties. Those settings are described on a section by section basis below.

Note: You can print the default configuration settings used for an executable by specifying the command line option `--hpx:dump-config`.

The system configuration section

```
[system]
pid = <process-id>
prefix = <current prefix path of core HPX library>
executable = <current prefix path of executable>
```

Property	Description
system.pid	This is initialized to store the current OS-process id of the application instance.
system.prefix	This is initialized to the base directory <i>HPX</i> has been loaded from.
system.executable_prefix	This is initialized to the base directory the current executable has been loaded from.

The `'hpx'` configuration section

```
[hpx]
location = ${HPX_LOCATION:${system.prefix}}
component_path = ${hpx.location}/lib/hpx:${system.executable_prefix}/lib/hpx:${system.
↳ executable_prefix}/../lib/hpx
master_ini_path = ${hpx.location}/share/hpx-<version>:${system.executable_prefix}/
↳ share/hpx-<version>:${system.executable_prefix}/../share/hpx-<version>
ini_path = ${hpx.master_ini_path}/ini
os_threads = 1
localities = 1
program_name =
cmd_line =
lock_detection = ${HPX_LOCK_DETECTION:0}
throw_on_held_lock = ${HPX_THROW_ON_HELD_LOCK:1}
minimal_deadlock_detection = <debug>
spinlock_deadlock_detection = <debug>
spinlock_deadlock_detection_limit = ${HPX_SPINLOCK_DEADLOCK_DETECTION_LIMIT:1000000}
max_background_threads = ${HPX_MAX_BACKGROUND_THREADS:${hpx.os_threads}}
max_idle_loop_count = ${HPX_MAX_IDLE_LOOP_COUNT:<hpx_idle_loop_count_max>}
max_busy_loop_count = ${HPX_MAX_BUSY_LOOP_COUNT:<hpx_busy_loop_count_max>}
max_idle_backoff_time = ${HPX_MAX_IDLE_BACKOFF_TIME:<hpx_idle_backoff_time_max>}
exception_verbosity = ${HPX_EXCEPTION_VERBOSITY:2}
```

[hpx.stacks]

(continues on next page)

(continued from previous page)

```
small_size = ${HPX_SMALL_STACK_SIZE:<hpx_small_stack_size>}
medium_size = ${HPX_MEDIUM_STACK_SIZE:<hpx_medium_stack_size>}
large_size = ${HPX_LARGE_STACK_SIZE:<hpx_large_stack_size>}
huge_size = ${HPX_HUGE_STACK_SIZE:<hpx_huge_stack_size>}
use_guard_pages = ${HPX_THREAD_GUARD_PAGE:1}
```


Property	Description
hpx.location	This is initialized to the id of the <i>locality</i> this application instance is running on.
hpx.component	Duplicates are discarded. This property can refer to a list of directories separated by ':' (Linux, Android, and MacOS) or by ';' (Windows).
hpx.master_ini	This is initialized to the list of default paths of the main hpx.ini configuration files. This property can refer to a list of directories separated by ':' (Linux, Android, and MacOS) or using ';' (Windows).
hpx.ini_path	This is initialized to the default path where <i>HPX</i> will look for more ini configuration files. This property can refer to a list of directories separated by ':' (Linux, Android, and MacOS) or using ';' (Windows).
hpx.os_threads	This setting reflects the number of OS threads used for running <i>HPX</i> threads. Defaults to number of detected cores (not hyperthreads/PUs).
hpx.localities	This setting reflects the number of localities the application is running on. Defaults to 1.
hpx.program_name	This setting reflects the program name of the application instance. Initialized from the command line argv[0].
hpx.cmd_line	This setting reflects the actual command line used to launch this application instance.
hpx.lock_detection	This setting verifies that no locks are being held while a <i>HPX</i> thread is suspended. This setting is applicable only if HPX_WITH_VERIFY_LOCKS is set during configuration in CMake.
hpx.throw_on_error	This setting causes an exception if during lock detection at least one lock is being held while a <i>HPX</i> thread is suspended. This setting is applicable only if HPX_WITH_VERIFY_LOCKS is set during configuration in CMake. This setting has no effect if hpx.lock_detection=0.
hpx.minimal_deadlock_detection	This setting enables support for minimal deadlock detection for <i>HPX</i> threads. By default this is set to 1 (for Debug builds) or to 0 (for Release, RelWithDebInfo, RelMinSize builds). This setting is effective only if HPX_WITH_THREAD_DEADLOCK_DETECTION is set during configuration in CMake.
hpx.spinlock_detection_limit	This setting verifies that spinlocks don't spin longer than specified using the hpx.spinlock_detection_limit. This setting is applicable only if HPX_WITH_SPINLOCK_DEADLOCK_DETECTION is set during configuration in CMake. By default this is set to 1 (for Debug builds) or to 0 (for Release, RelWithDebInfo, RelMinSize builds).
hpx.spinlock_perform	This setting specifies the upper limit of the allowed number of spins that spinlocks are allowed to perform. This setting is applicable only if HPX_WITH_SPINLOCK_DEADLOCK_DETECTION is set during configuration in CMake. By default this is set to 1000000.
hpx.max_background_work	This setting defines the number of threads in the scheduler, which are used to execute background work. By default this is the same as the number of cores used for the scheduler.
hpx.max_idle_loops	By default this is defined by the preprocessor constant HPX_IDLE_LOOP_COUNT_MAX. This is an internal setting that you should change only if you know exactly what you are doing.
hpx.max_busy_loops	This setting defines the maximum value of the busy-loop counter in the scheduler. By default this is defined by the preprocessor constant HPX_BUSY_LOOP_COUNT_MAX. This is an internal setting that you should change only if you know exactly what you are doing.
hpx.max_idle_time_for_hpx_max_idle_loop_count	This setting defines the maximum time (in milliseconds) for the scheduler to sleep after being idle for hpx.max_idle_loop_count iterations. This setting is applicable only if HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF is set during configuration in CMake. By default this is defined by the preprocessor constant HPX_IDLE_BACKOFF_TIME_MAX. This is an internal setting that you should change only if you know exactly what you are doing.
hpx.exception_level	This setting defines the verbosity of exceptions. Valid values are integers. A setting of 2 or higher prints all available information. A setting of 1 leaves out the build configuration and environment variables. A setting of 0 or lower prints only the description of the thrown exception and the file name, function, and line number where the exception was thrown. The default value is 2 or the value of the environment variable HPX_EXCEPTION_VERTOSITY.
hpx.stack_size	This is initialized to the small stack size to be used by <i>HPX</i> threads. Set by default to the value of the compile time preprocessor constant HPX_SMALL_STACK_SIZE (defaults to 0x8000). This value is used for all <i>HPX</i> threads by default, except for the thread running hpx_main (which runs on a large stack).
hpx.stack_size_medium	This is initialized to the medium stack size to be used by <i>HPX</i> threads. Set by default to the value of the compile time preprocessor constant HPX_MEDIUM_STACK_SIZE (defaults to 0x20000).

The `hpx.threadpools` configuration section

```
[hpx.threadpools]
io_pool_size = ${HPX_NUM_IO_POOL_SIZE:2}
parcel_pool_size = ${HPX_NUM_PARCEL_POOL_SIZE:2}
timer_pool_size = ${HPX_NUM_TIMER_POOL_SIZE:2}
```

Property	Description
<code>hpx.threadpools.io_pool_size</code>	The value of this property defines the number of OS threads created for the internal I/O thread pool.
<code>hpx.threadpools.parcel_pool_size</code>	The value of this property defines the number of OS threads created for the internal parcel thread pool.
<code>hpx.threadpools.timer_pool_size</code>	The value of this property defines the number of OS threads created for the internal timer thread pool.

The `hpx.thread_queue` configuration section

Important: These are the setting control internal values used by the thread scheduling queues in the *HPX* scheduler. You should not modify these settings unless you know exactly what you are doing.

```
[hpx.thread_queue]
min_tasks_to_steal_pending = ${HPX_THREAD_QUEUE_MIN_TASKS_TO_STEAL_PENDING:0}
min_tasks_to_steal_staged = ${HPX_THREAD_QUEUE_MIN_TASKS_TO_STEAL_STAGE:0}
min_add_new_count = ${HPX_THREAD_QUEUE_MIN_ADD_NEW_COUNT:10}
max_add_new_count = ${HPX_THREAD_QUEUE_MAX_ADD_NEW_COUNT:10}
max_delete_count = ${HPX_THREAD_QUEUE_MAX_DELETE_COUNT:1000}
```

Property	Description
<code>hpx.thread_queue.min_tasks_to_steal_pending</code>	The value of this property defines the number of pending <i>HPX</i> threads that have to be available before neighboring cores are allowed to steal work. The default is to allow stealing always.
<code>hpx.thread_queue.min_tasks_to_steal_staged</code>	The value of this property defines the number of staged <i>HPX</i> tasks that need to be available before neighboring cores are allowed to steal work. The default is to allow stealing always.
<code>hpx.thread_queue.min_add_new_count</code>	The value of this property defines the minimal number of tasks to be converted into <i>HPX</i> threads whenever the thread queues for a core have run empty.
<code>hpx.thread_queue.max_add_new_count</code>	The value of this property defines the maximal number of tasks to be converted into <i>HPX</i> threads whenever the thread queues for a core have run empty.
<code>hpx.thread_queue.max_delete_count</code>	The value of this property defines the number of terminated <i>HPX</i> threads to discard during each invocation of the corresponding function.

The `hpx.components` configuration section

```
[hpx.components]
load_external = ${HPX_LOAD_EXTERNAL_COMPONENTS:1}
```

Property	Description
<code>hpx.components.load_external</code>	This entry defines whether external components will be loaded on this <i>locality</i> . This entry is normally set to 1, and usually there is no need to directly change this value. It is automatically set to 0 for a dedicated <i>AGAS</i> server <i>locality</i> .

Additionally, the section `hpx.components` will be populated with the information gathered from all found components. The information loaded for each of the components will contain at least the following properties:

```
[hpx.components.<component_instance_name>]
name = <component_name>
path = <full_path_of_the_component_module>
enabled = ${hpx.components.load_external}
```

Property	Description
<code>hpx.components.<component_instance_name>.name</code>	This is the name of a component, usually the same as the second argument to the macro used while registering the component with <code>HPX_REGISTER_COMPONENT</code> . Set by the component factory.
<code>hpx.components.<component_instance_name>.path</code>	This is either the full path file name of the component module or the directory the component module is located in. In this case, the component module name will be derived from the property <code>hpx.components.<component_instance_name>.name</code> . Set by the component factory.
<code>hpx.components.<component_instance_name>.enabled</code>	This setting explicitly enables or disables the component. This is an optional property. HPX assumes that the component is enabled if it is not defined.

The value for `<component_instance_name>` is usually the same as for the corresponding `name` property. However, generally it can be defined to any arbitrary instance name. It is used to distinguish between different ini sections, one for each component.

The `hpx.parcel` configuration section

```
[hpx.parcel]
address = ${HPX_PARCEL_SERVER_ADDRESS:<hpx_initial_ip_address>}
port = ${HPX_PARCEL_SERVER_PORT:<hpx_initial_ip_port>}
bootstrap = ${HPX_PARCEL_BOOTSTRAP:<hpx_parcel_bootstrap>}
max_connections = ${HPX_PARCEL_MAX_CONNECTIONS:<hpx_parcel_max_connections>}
max_connections_per_locality = ${HPX_PARCEL_MAX_CONNECTIONS_PER_LOCALITY:<hpx_parcel_max_connections_per_locality>}
max_message_size = ${HPX_PARCEL_MAX_MESSAGE_SIZE:<hpx_parcel_max_message_size>}
max_outbound_message_size = ${HPX_PARCEL_MAX_OUTBOUND_MESSAGE_SIZE:<hpx_parcel_max_outbound_message_size>}
array_optimization = ${HPX_PARCEL_ARRAY_OPTIMIZATION:1}
zero_copy_optimization = ${HPX_PARCEL_ZERO_COPY_OPTIMIZATION:[hpx.parcel.array_optimization]}
```

(continues on next page)

(continued from previous page)

```
async_serialization = ${HPX_PARCEL_ASYNC_SERIALIZATION:1}
message_handlers = ${HPX_PARCEL_MESSAGE_HANDLERS:0}
```

Property	Description
hpx.parcel.address	This property defines the default IP address to be used for the <i>parcel</i> layer to listen to. This IP address will be used as long as no other values are specified (for instance, using the <code>--hpx:hpx</code> command line option). The expected format is any valid IP address or domain name format that can be resolved into an IP address. The default depends on the compile time preprocessor constant <code>HPX_INITIAL_IP_ADDRESS ("127.0.0.1")</code> .
hpx.parcel.port	This property defines the default IP port to be used for the <i>parcel</i> layer to listen to. This IP port will be used as long as no other values are specified (for instance using the <code>--hpx:hpx</code> command line option). The default depends on the compile time preprocessor constant <code>HPX_INITIAL_IP_PORT (7910)</code> .
hpx.parcel.bootstrap	This property defines which parcelport type should be used during application bootstrap. The default depends on the compile time preprocessor constant <code>HPX_PARCEL_BOOTSTRAP ("tcp")</code> .
hpx.parcel.max_connections	This property defines how many network connections between different localities are overall kept alive by each <i>locality</i> . The default depends on the compile time preprocessor constant <code>HPX_PARCEL_MAX_CONNECTIONS (512)</code> .
hpx.parcel.max_connections_per_locality	This property defines the maximum number of network connections that one <i>locality</i> will open to another <i>locality</i> . The default depends on the compile time preprocessor constant <code>HPX_PARCEL_MAX_CONNECTIONS_PER_LOCALITY (4)</code> .
hpx.parcel.max_message_size	This property defines the maximum allowed message size that will be transferrable through the <i>parcel</i> layer. The default depends on the compile time preprocessor constant <code>HPX_PARCEL_MAX_MESSAGE_SIZE (1000000000 bytes)</code> .
hpx.parcel.max_outbound_message_size	This property defines the maximum allowed outbound coalesced message size that will be transferrable through the <i>parcel</i> layer. The default depends on the compile time preprocessor constant <code>HPX_PARCEL_MAX_OUTBOUND_MESSAGE_SIZE (1000000 bytes)</code> .
hpx.parcel.array_optimization	This property defines whether this <i>locality</i> is allowed to utilize array optimizations during serialization of <i>parcel</i> data. The default is 1.
hpx.parcel.zero_copy_optimization	This property defines whether this <i>locality</i> is allowed to utilize zero copy optimizations during serialization of <i>parcel</i> data. The default is the same value as set for <code>hpx.parcel.array_optimization</code> .
hpx.parcel.zero_copy_serialization_threshold	This property defines the threshold value (in bytes) starting at which the serialization layer will apply zero-copy optimizations for serialized entities. The default value is defined by the preprocessor constant <code>HPX_ZERO_COPY_SERIALIZATION_THRESHOLD</code> .
hpx.parcel.async_serialization	This property defines whether this <i>locality</i> is allowed to spawn a new thread for serialization (this is both for encoding and decoding parcels). The default is 1.
hpx.parcel.message_handlers	This property defines whether message handlers are loaded. The default is 0.
hpx.parcel.max_background_threads	This property defines how many cores should be used to perform background operations. The default is -1 (all cores).

The following settings relate to the TCP/IP parcelport.

```
[hpx.parcel.tcp]
enable = ${HPX_HAVE_PARCELPORT_TCP:$[hpx.parcel.enabled]}
```

(continues on next page)

(continued from previous page)

```

array_optimization = ${HPX_PARCEL_TCP_ARRAY_OPTIMIZATION:[hpx.parcel.array_
    ↪optimization]}
zero_copy_optimization = ${HPX_PARCEL_TCP_ZERO_COPY_OPTIMIZATION:[hpx.parcel.zero_
    ↪copy_optimization]}
zero_copy_serialization_threshold = ${HPX_PARCEL_TCP_ZERO_COPY_SERIALIZATION_
    ↪THRESHOLD:[hpx.parcel.zero_copy_serialization_threshold]}
async_serialization = ${HPX_PARCEL_TCP_ASYNC_SERIALIZATION:[hpx.parcel.async_
    ↪serialization]}
parcel_pool_size = ${HPX_PARCEL_TCP_PARCEL_POOL_SIZE:[hpx.threadpools.parcel_pool_
    ↪size]}
max_connections = ${HPX_PARCEL_TCP_MAX_CONNECTIONS:[hpx.parcel.max_connections]}
max_connections_per_locality = ${HPX_PARCEL_TCP_MAX_CONNECTIONS_PER_LOCALITY:[hpx.
    ↪parcel.max_connections_per_locality]}
max_message_size = ${HPX_PARCEL_TCP_MAX_MESSAGE_SIZE:[hpx.parcel.max_message_size]}
max_outbound_message_size = ${HPX_PARCEL_TCP_MAX_OUTBOUND_MESSAGE_SIZE:[hpx.parcel.
    ↪max_outbound_message_size]}
max_background_threads = ${HPX_PARCEL_TCP_MAX_BACKGROUND_THREADS:[hpx.parcel.max_
    ↪background_threads]}

```

Property	Description
hpx.parcel.tcp.enable	Enables the use of the default TCP parcelport. Note that the initial bootstrap of the overall <i>HPX</i> application will be performed using the default TCP connections. This parcelport is enabled by default. This will be disabled only if MPI is enabled (see below).
hpx.parcel.tcp.array_optimization	This property defines whether this <i>locality</i> is allowed to utilize array optimizations in the TCP/IP parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.array_optimization.
hpx.parcel.tcp.zero_copy_optimization	This property defines whether this <i>locality</i> is allowed to utilize zero copy optimizations in the TCP/IP parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.zero_copy_optimization.
hpx.parcel.tcp.zero_copy_serialization_threshold	This property defines the threshold value (in bytes) starting at which the serialization layer will apply zero copy optimizations for serialized entities. The default is the same value as set for hpx.parcel.zero_copy_serialization_threshold.
hpx.parcel.tcp.async_serialization	This property defines whether this <i>locality</i> is allowed to spawn a new thread for serialization in the TCP/IP parcelport (this is both for encoding and decoding parcels). The default is the same value as set for hpx.parcel.async_serialization.
hpx.parcel.tcp.parcel_pool_size	The value of this property defines the number of OS threads created for the internal parcel thread pool of the TCP <i>parcel</i> port. The default is taken from hpx.threadpools.parcel_pool_size.
hpx.parcel.tcp.max_connections	This property defines how many network connections between different localities are overall kept alive by each <i>locality</i> . The default is taken from hpx.parcel.max_connections.
hpx.parcel.tcp.max_connections_per_locality	This property defines the maximum number of network connections that one <i>locality</i> will open to another <i>locality</i> . The default is taken from hpx.parcel.max_connections_per_locality.
hpx.parcel.tcp.max_message_size	This property defines the maximum allowed message size that will be transferrable through the <i>parcel</i> layer. The default is taken from hpx.parcel.max_message_size.
hpx.parcel.tcp.max_outbound_message_size	This property defines the maximum allowed outbound coalesced message size that will be transferrable through the <i>parcel</i> layer. The default is taken from hpx.parcel.max_outbound_connections.
hpx.parcel.tcp.max_background_threads	This property defines how many cores should be used to perform background operations. The default is taken from hpx.parcel.max_background_threads.

The following settings relate to the MPI parcelport. These settings take effect only if the compile time constant `HPX_HAVE_PARCELPORT_MPI` is set (the equivalent CMake variable is `HPX_WITH_PARCELPORT_MPI` and has to be set to ON).

```
[hpx.parcel.mpi]
enable = ${HPX_HAVE_PARCELPORT_MPI:$[hpx.parcel.enabled]}
env = ${HPX_HAVE_PARCELPORT_MPI_ENV:MV2_COMM_WORLD_RANK,PMI_RANK,OMPI_COMM_WORLD_SIZE,
    ↪ALPS_APP_PE,PALS_NODEID}
multithreaded = ${HPX_HAVE_PARCELPORT_MPI_MULTITHREADED:1}
rank = <MPI_rank>
processor_name = <MPI_processor_name>
array_optimization = ${HPX_HAVE_PARCEL_MPI_ARRAY_OPTIMIZATION:$[hpx.parcel.array_
    ↪optimization]}
zero_copy_optimization = ${HPX_HAVE_PARCEL_MPI_ZERO_COPY_OPTIMIZATION:$[hpx.parcel.
    ↪zero_copy_optimization]}
zero_copy_serialization_threshold = ${HPX_PARCEL_MPI_ZERO_COPY_SERIALIZATION_
    ↪THRESHOLD:$[hpx.parcel.zero_copy_serialization_threshold]}
use_io_pool = ${HPX_HAVE_PARCEL_MPI_USE_IO_POOL:$1}
async_serialization = ${HPX_HAVE_PARCEL_MPI_ASYNC_SERIALIZATION:$[hpx.parcel.async_
    ↪serialization]}
parcel_pool_size = ${HPX_HAVE_PARCEL_MPI_PARCEL_POOL_SIZE:$[hpx.threadpools.parcel_
    ↪pool_size]}
max_connections = ${HPX_HAVE_PARCEL_MPI_MAX_CONNECTIONS:$[hpx.parcel.max_
    ↪connections]}
max_connections_per_locality = ${HPX_HAVE_PARCEL_MPI_MAX_CONNECTIONS_PER_LOCALITY:
    ↪$[hpx.parcel.max_connections_per_locality]}
max_message_size = ${HPX_HAVE_PARCEL_MPI_MAX_MESSAGE_SIZE:$[hpx.parcel.max_message_
    ↪size]}
max_outbound_message_size = ${HPX_HAVE_PARCEL_MPI_MAX_OUTBOUND_MESSAGE_SIZE:$[hpx.
    ↪parcel.max_outbound_message_size]}
max_background_threads = ${HPX_PARCEL_MPI_MAX_BACKGROUND_THREADS:$[hpx.parcel.max_
    ↪background_threads]}
```

Property	Description
hpx.parcel.mpi.enable	Enables the use of the MPI parcelport. <i>HPX</i> tries to detect if the application was started within a parallel MPI environment. If the detection was successful, the MPI parcelport is enabled by default. To explicitly disable the MPI parcelport, set to 0. Note that the initial bootstrap of the overall <i>HPX</i> application will be performed using MPI as well.
hpx.parcel.mpi.env	This property influences which environment variables (separated by commas) will be analyzed to find out whether the application was invoked by MPI.
hpx.parcel.mpi.multithreaded	This property is used to determine what threading mode to use when initializing MPI. If this setting is 0, <i>HPX</i> will initialize MPI with MPI_THREAD_SINGLE. If the value is not equal to 0, <i>HPX</i> will initialize MPI with MPI_THREAD_MULTI.
hpx.parcel.mpi.rank	This property will be initialized to the MPI rank of the <i>locality</i> .
hpx.parcel.mpi.processor_name	This property will be initialized to the MPI processor name of the <i>locality</i> .
hpx.parcel.mpi.array_optimization	This property defines whether this <i>locality</i> is allowed to utilize array optimizations in the MPI parcelport during serialization of <i>parcel</i> data. The default is the same value as set for hpx.parcel.array_optimization.
hpx.parcel.mpi.zero_copy_optimization	This property defines whether this <i>locality</i> is allowed to utilize zero copy optimizations in the MPI parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.zero_copy_optimization.
hpx.parcel.mpi.zero_copy_serialization_threshold	This property defines the threshold value (in bytes) starting at which the serialization layer will apply zero-copy optimizations for serialized entities. The default is the same value as set for hpx.parcel.zero_copy_serialization_threshold.
hpx.parcel.mpi.use_io_pool	This property can be set to run the progress thread inside of <i>HPX</i> threads instead of a separate thread pool. The default is 1.
hpx.parcel.mpi.async_serialization	This property defines whether this <i>locality</i> is allowed to spawn a new thread for serialization in the MPI parcelport (this is both for encoding and decoding parcels). The default is the same value as set for hpx.parcel.async_serialization.
hpx.parcel.mpi.parcel_pool_size	The value of this property defines the number of OS threads created for the internal parcel thread pool of the MPI <i>parcel</i> port. The default is taken from hpx.threadpools.parcel_pool_size.
hpx.parcel.mpi.max_connections	This property defines how many network connections between different localities are overall kept alive by each <i>locality</i> . The default is taken from hpx.parcel.max_connections.
hpx.parcel.mpi.max_connections_per_locality	This property defines the maximum number of network connections that one <i>locality</i> will open to another <i>locality</i> . The default is taken from hpx.parcel.max_connections_per_locality.
hpx.parcel.mpi.max_message_size	This property defines the maximum allowed message size that will be transferrable through the <i>parcel</i> layer. The default is taken from hpx.parcel.max_message_size.
hpx.parcel.mpi.max_outbound_message_size	This property defines the maximum allowed outbound coalesced message size that will be transferrable through the <i>parcel</i> layer. The default is taken from hpx.parcel.max_outbound_message_size.
hpx.parcel.tcp.max_background_threads	This property defines how many cores should be used to perform background operations. The default is taken from hpx.parcel.max_background_threads.

The `hpx.agas` configuration section

```
[hpx.agas]
address = ${HPX_AGAS_SERVER_ADDRESS:<hpx_initial_ip_address>}
port = ${HPX_AGAS_SERVER_PORT:<hpx_initial_ip_port>}
service_mode = hosted
dedicated_server = 0
max_pending_refcnt_requests = ${HPX_AGAS_MAX_PENDING_REFCNT_REQUESTS:<hpx_initial_
    ↴agас_max_pending_refcnt_requests>}
use_caching = ${HPX_AGAS_USE_CACHING:1}
use_range_caching = ${HPX_AGAS_USE_RANGE_CACHING:1}
local_cache_size = ${HPX_AGAS_LOCAL_CACHE_SIZE:<hpx_agas_local_cache_size>}
```

Property	Description
<code>hpx.agas.address</code>	This property defines the default IP address to be used for the <code>AGAS</code> root server. This IP address will be used as long as no other values are specified (for instance, using the <code>--hpx:agас</code> command line option). The expected format is any valid IP address or domain name format that can be resolved into an IP address. The default depends on the compile time preprocessor constant <code>HPX_INITIAL_IP_ADDRESS</code> ("127.0.0.1").
<code>hpx.agas.port</code>	This property defines the default IP port to be used for the <code>AGAS</code> root server. This IP port will be used as long as no other values are specified (for instance, using the <code>--hpx:агас</code> command line option). The default depends on the compile time preprocessor constant <code>HPX_INITIAL_IP_PORT</code> (7009).
<code>hpx.agas.service_mode</code>	This property specifies what type of <code>AGAS</code> service is running on this <code>locality</code> . Currently, two modes exist. The <code>locality</code> that acts as the <code>AGAS</code> server runs in <code>bootstrap</code> mode. All other localities are in <code>hosted</code> mode.
<code>hpx.agas.dedicated</code>	This property specifies whether the <code>AGAS</code> server is exclusively running <code>AGAS</code> services and not hosting any application components. It is a boolean value. Set to 1 if <code>separate-run-agas-server-only</code> is present.
<code>hpx.agas.max_pending_refcnt_requests</code>	This property defines the number of reference counting requests (increments or decrements) to buffer. The default depends on the compile time preprocessor constant <code>HPX_DEFAULT_MAX_AGAS_MAX_PENDING_REFCNT_REQUESTS</code> (4096).
<code>hpx.agas.use_caching</code>	This property specifies whether a software address translation cache is used. It is a boolean value. Defaults to 1.
<code>hpx.agas.use_range_caching</code>	This property specifies whether range-based caching is used by the software address translation cache. This property is ignored if <code>hpx.agas.use_caching</code> is false. It is a boolean value. Defaults to 1.
<code>hpx.agas.local_cache_size</code>	This property defines the size of the software address translation cache for <code>AGAS</code> services. This property is ignored if <code>hpx.agas.use_caching</code> is false. Note that if <code>hpx.agas.use_range_caching</code> is true, this size will refer to the maximum number of ranges stored in the cache, not the number of entries spanned by the cache. The default depends on the compile time preprocessor constant <code>HPX_AGAS_LOCAL_CACHE_SIZE</code> (4096).

The `hpx.commandline` configuration section

The following table lists the definition of all pre-defined command line option shortcuts. For more information about commandline options, see the section [HPX Command Line Options](#).

```
[hpx.commandline]
aliasing = ${HPX_COMMANDLINE_ALIASING:1}
allow_unknown = ${HPX_COMMANDLINE_ALLOW_UNKNOWN:0}

[hpx.commandline.aliases]
-a = --hpx:agas
-c = --hpx:console
-h = --hpx:help
-I = --hpx:ini
-l = --hpx:localities
-p = --hpx:app-config
-q = --hpx:queuing
-r = --hpx:run-agas-server
-t = --hpx:threads
-v = --hpx:version
-w = --hpx:worker
-x = --hpx:hpx
-0 = --hpx:node=0
-1 = --hpx:node=1
-2 = --hpx:node=2
-3 = --hpx:node=3
-4 = --hpx:node=4
-5 = --hpx:node=5
-6 = --hpx:node=6
-7 = --hpx:node=7
-8 = --hpx:node=8
-9 = --hpx:node=9
```


Property	Description
hpx.commandline. aliasing	Enable command line aliases as defined in the section <code>hpx.commandline.aliases</code> (see below). Defaults to 1.
hpx.commandline. allow_unknown	Allow for unknown command line options to be passed through to <code>hpx_main()</code> . Defaults to 0.
hpx.commandline. aliases.-a	On the commandline <code>-a</code> expands to: <code>--hpx:agas</code> .
hpx.commandline. aliases.-c	On the commandline <code>-c</code> expands to: <code>--hpx:console</code> .
hpx.commandline. aliases.-h	On the commandline <code>-h</code> expands to: <code>--hpx:help</code> .
hpx.commandline. aliases.--help	On the commandline <code>--help</code> expands to: <code>--hpx:help</code> .
hpx.commandline. aliases.-I	On the commandline <code>-I</code> expands to: <code>--hpx:ini</code> .
hpx.commandline. aliases.-l	On the commandline <code>-l</code> expands to: <code>--hpx:localities</code> .
hpx.commandline. aliases.-p	On the commandline <code>-p</code> expands to: <code>--hpx:app-config</code> .
hpx.commandline. aliases.-q	On the commandline <code>-q</code> expands to: <code>--hpx:queuing</code> .
hpx.commandline. aliases.-r	On the commandline <code>-r</code> expands to: <code>--hpx:run-agas-server</code> .
hpx.commandline. aliases.-t	On the commandline <code>-t</code> expands to: <code>--hpx:threads</code> .
hpx.commandline. aliases.-v	On the commandline <code>-v</code> expands to: <code>--hpx:version</code> .
hpx.commandline. aliases.--version	On the commandline <code>--version</code> expands to: <code>--hpx:version</code> .
hpx.commandline. aliases.-w	On the commandline <code>-w</code> expands to: <code>--hpx:worker</code> .
hpx.commandline. aliases.-x	On the commandline <code>-x</code> expands to: <code>--hpx:hpx</code> .
hpx.commandline. aliases.-0	On the commandline <code>-0</code> expands to: <code>--hpx:node=0</code> .
hpx.commandline. aliases.-1	On the commandline <code>-1</code> expands to: <code>--hpx:node=1</code> .
hpx.commandline. aliases.-2	On the commandline <code>-2</code> expands to: <code>--hpx:node=2</code> .
hpx.commandline. aliases.-3	On the commandline <code>-3</code> expands to: <code>--hpx:node=3</code> .
hpx.commandline. aliases.-4	On the commandline <code>-4</code> expands to: <code>--hpx:node=4</code> .
hpx.commandline. aliases.-5	On the commandline <code>-5</code> expands to: <code>--hpx:node=5</code> .
hpx.commandline. aliases.-6	On the commandline <code>-6</code> expands to: <code>--hpx:node=6</code> .
hpx.commandline. aliases.-7	On the commandline <code>-7</code> expands to: <code>--hpx:node=7</code> .
hpx.commandline. aliases.-8	On the commandline <code>-8</code> expands to: <code>--hpx:node=8</code> .
hpx.commandline. aliases.-9	On the commandline <code>-9</code> expands to: <code>--hpx:node=9</code> .

Loading INI files

During startup and after the internal database has been initialized as described in the section [Built-in default configuration settings](#), HPX will try to locate and load additional ini files to be used as a source for configuration properties. This allows for a wide spectrum of additional customization possibilities by the user and system administrators. The sequence of locations where HPX will try loading the ini files is well defined and documented in this section. All ini files found are merged into the internal configuration database. The merge operation itself conforms to the rules as described in the section [The HPX ini file format](#).

1. Load all component shared libraries found in the directories specified by the property `hpx.component_path` and retrieve their default configuration information (see section [Loading components](#) for more details). This property can refer to a list of directories separated by ':' (Linux, Android, and MacOS) or by ';' (Windows).
2. Load all files named `hpx.ini` in the directories referenced by the property `hpx.master_ini_path`. This property can refer to a list of directories separated by ':' (Linux, Android, and MacOS) or by ';' (Windows).
3. Load a file named `.hpx.ini` in the current working directory, e.g., the directory the application was invoked from.
4. Load a file referenced by the environment variable `HPX_INI`. This variable is expected to provide the full path name of the ini configuration file (if any).
5. Load a file named `/etc/hpx.ini`. This lookup is done on non-Windows systems only.
6. Load a file named `.hpx.ini` in the home directory of the current user, e.g., the directory referenced by the environment variable `HOME`.
7. Load a file named `.hpx.ini` in the directory referenced by the environment variable `PWD`.
8. Load the file specified on the command line using the option `--hpx:config`.
9. Load all properties specified on the command line using the option `--hpx:ini`. The properties will be added to the database in the same sequence as they are specified on the command line. The format for those options is, for instance, `--hpx:ini=hpx.default_stack_size=0x4000`. In addition to the explicit command line options, this will set the following properties as implied from other settings:
 - `hpx.parcel.address` and `hpx.parcel.port` as set by `--hpx:hpx`
 - `hpx.agas.address`, `hpx.agas.port` and `hpx.agas.service_mode` as set by `--hpx:agas`
 - `hpx.program_name` and `hpx.cmd_line` will be derived from the actual command line
 - **`hpx.os_threads` and `hpx.localities` as set by `--hpx:threads` and `--hpx:localities`**
 - `hpx.runtime_mode` will be derived from any explicit `--hpx:console`, `--hpx:worker`, or `--hpx:connect`, or it will be derived from other settings, such as `--hpx:node =0`, which implies `--hpx:console`.
10. Load files based on the pattern `*.ini` in all directories listed by the property `hpx.ini_path`. All files found during this search will be merged. The property `hpx.ini_path` can hold a list of directories separated by ':' (on Linux or Mac) or ';' (on Windows).
11. Load the file specified on the command line using the option `--hpx:app-config`. Note that this file will be merged as the content for a top level section `[application]`.

Note: Any changes made to the configuration database caused by one of the steps will influence the loading process for all subsequent steps. For instance, if one of the ini files loaded changes the property `hpx.ini_path`, this will influence the directories searched in step 9 as described above.

Important: The *HPX* core library will verify that all configuration settings specified on the command line (using the `--hpx:ini` option) will be checked for validity. That means that the library will accept only *known* configuration settings. This is to protect the user from unintentional typos while specifying those settings. This behavior can be overwritten by appending a '!' to the configuration key, thus forcing the setting to be entered into the configuration database. For instance: `--hpx:ini=hpx.foo! = 1`

If any of the environment variables or files listed above are not found, the corresponding loading step will be silently skipped.

Loading components

HPX relies on loading application specific components during the runtime of an application. Moreover, *HPX* comes with a set of preinstalled components supporting basic functionalities useful for almost every application. Any component in *HPX* is loaded from a shared library, where any of the shared libraries can contain more than one component type. During startup, *HPX* tries to locate all available components (e.g., their corresponding shared libraries) and creates an internal component registry for later use. This section describes the algorithm used by *HPX* to locate all relevant shared libraries on a system. As described, this algorithm is customizable by the configuration properties loaded from the ini files (see section [Loading INI files](#)).

Loading components is a two-stage process. First *HPX* tries to locate all component shared libraries, loads those, and generates a default configuration section in the internal configuration database for each component found. For each found component the following information is generated:

```
[hpx.components.<component_instance_name>]
name = <name_of_shared_library>
path = ${component_path}
enabled = ${hpx.components.load_external}
default = 1
```

The values in this section correspond to the expected configuration information for a component as described in the section [Built-in default configuration settings](#).

In order to locate component shared libraries, *HPX* will try loading all shared libraries (files with the platform specific extension of a shared library, Linux: *.so, Windows: *.dll, MacOS: *.dylib found in the directory referenced by the ini property `hpx.component_path`).

This first step corresponds to step 1) during the process of filling the internal configuration database with default information as described in section [Loading INI files](#).

After all of the configuration information has been loaded, *HPX* performs the second step in terms of loading components. During this step, *HPX* scans all existing configuration sections `[hpx.component.<some_component_instance_name>]` and instantiates a special factory object for each of the successfully located and loaded components. During the application's life time, these factory objects are responsible for creating new and discarding old instances of the component they are associated with. This step is performed after step 11) of the process of filling the internal configuration database with default information as described in section [Loading INI files](#).

Application specific component example

This section assumes there is a simple application component that exposes one member function as a component action. The header file `app_server.hpp` declares the C++ type to be exposed as a component. This type has a member function `print_greeting()`, which is exposed as an action `print_greeting_action`. We assume the source files for this example are located in a directory referenced by `$APP_ROOT`:

```
// file: $APP_ROOT/app_server.hpp
#include <hpx/hpx.hpp>
#include <hpx/include/iostreams.hpp>

namespace app
{
    // Define a simple component exposing one action 'print_greeting'
    class HPX_COMPONENT_EXPORT server
        : public hpx::components::component_base<server>
    {
        void print_greeting ()
        {
            hpx::cout << "Hey, how are you?\n" << std::flush;
        }

        // Component actions need to be declared, this also defines the
        // type 'print_greeting_action' representing the action.
        HPX_DEFINE_COMPONENT_ACTION(server, print_greeting, print_greeting_action);
    };
}

// Declare boilerplate code required for each of the component actions.
HPX_REGISTER_ACTION_DECLARATION(app::server::print_greeting_action);
```

The corresponding source file contains mainly macro invocations that define the boilerplate code needed for *HPX* to function properly:

```
// file: $APP_ROOT/app_server.cpp
#include "app_server.hpp"

// Define boilerplate required once per component module.
HPX_REGISTER_COMPONENT_MODULE();

// Define factory object associated with our component of type 'app::server'.
HPX_REGISTER_COMPONENT(app::server, app_server);

// Define boilerplate code required for each of the component actions. Use the
// same argument as used for HPX_REGISTER_ACTION_DECLARATION above.
HPX_REGISTER_ACTION(app::server::print_greeting_action);
```

The following gives an example of how the component can be used. Here, one instance of the `app::server` component is created on the current *locality* and the exposed action `print_greeting_action` is invoked using the global id of the newly created instance. Note that no special code is required to delete the component instance after it is not needed anymore. It will be deleted automatically when its last reference goes out of scope (shown in the example below at the closing brace of the block surrounding the code):

```
// file: $APP_ROOT/use_app_server_example.cpp
#include <hpx/hpx_init.hpp>
#include "app_server.hpp"
```

(continues on next page)

(continued from previous page)

```

int hpx_main()
{
{
    // Create an instance of the app_server component on the current locality.
    hpx::naming:id_type app_server_instance =
        hpx::create_component<app::server>(hpx::find_here());

    // Create an instance of the action 'print_greeting_action'.
    app::server::print_greeting_action print_greeting;

    // Invoke the action 'print_greeting' on the newly created component.
    print_greeting(app_server_instance);
}
return hpx::finalize();
}

int main(int argc, char* argv[])
{
    return hpx::init(argc, argv);
}

```

In order to make sure that the application will be able to use the component `app::server`, special configuration information must be passed to *HPX*. The simplest way to allow *HPX* to ‘find’ the component is to provide special ini configuration files that add the necessary information to the internal configuration database. The component should have a special ini file containing the information specific to the component `app_server`.

```

# file: $APP_ROOT/app_server.ini
[hpx.components.app_server]
name = app_server
path = $APP_LOCATION/

```

Here, `$APP_LOCATION` is the directory where the (binary) component shared library is located. *HPX* will attempt to load the shared library from there. The section name `hpx.components.app_server` reflects the instance name of the component (`app_server` is an arbitrary, but unique name). The property value for `hpx.components.app_server.name` should be the same as used for the second argument to the macro `HPX_REGISTER_COMPONENT` above.

Additionally, a file `.hpx.ini`, which could be located in the current working directory (see step 3 as described in the section [Loading INI files](#)), can be used to add to the ini search path for components:

```

# file: $PWD/.hpx.ini
[hpx]
ini_path = ${hpx.ini_path}:$APP_ROOT/

```

This assumes that the above ini file specific to the component is located in the directory `$APP_ROOT`.

Note: It is possible to reference the defined property from inside its value. *HPX* will gracefully use the previous value of `hpx.ini_path` for the reference on the right hand side and assign the overall (now expanded) value to the property.

Logging

HPX uses a sophisticated logging framework, allowing users to follow in detail what operations have been performed inside the *HPX* library in what sequence. This information proves to be very useful for diagnosing problems or just for improving the understanding of what is happening in *HPX* as a consequence of invoking *HPX* API functionality.

Default logging

Enabling default logging is a simple process. The detailed description in the remainder of this section explains different ways to customize the defaults. Default logging can be enabled by using one of the following:

- A command line switch `--hpx:debug-hpx-log`, which will enable logging to the console terminal.
- The command line switch `--hpx:debug-hpx-log=<filename>`, which enables logging to a given file `<filename>`.
- Setting an environment variable `HPX_LOGLEVEL=<loglevel>` while running the *HPX* application. In this case `<loglevel>` should be a number between (or equal to) 1 and 5 where 1 means minimal logging and 5 causes all available messages to be logged. When setting the environment variable, the logs will be written to a file named `hpx.<PID>.lo` in the current working directory, where `<PID>` is the process id of the console instance of the application.

Customizing logging

Generally, logging can be customized either using environment variable settings or using by an ini configuration file. Logging is generated in several categories, each of which can be customized independently. All customizable configuration parameters have reasonable defaults, allowing for the use of logging without any additional configuration effort. The following table lists the available categories.

Table 2.5: Logging categories

Category	Category shortcut	Information to be generated	Environment variable
General	None	Logging information generated by different subsystems of <i>HPX</i> , such as thread-manager, parcel layer, LCOs, etc.	<code>HPX_LOGLEVEL</code>
<i>AGAS</i>	<i>AGAS</i>	Logging output generated by the <i>AGAS</i> subsystem	<code>HPX_AGAS_LOGLEVEL</code>
Application	APP	Logging generated by applications.	<code>HPX_APP_LOGLEVEL</code>

By default, all logging output is redirected to the console instance of an application, where it is collected and written to a file, one file for each logging category.

Each logging category can be customized at two levels. The parameters for each are stored in the ini configuration sections `hpx.logging.CATEGORY` and `hpx.logging.console.CATEGORY` (where `CATEGORY` is the category shortcut as listed in the table above). The former influences logging at the source *locality* and the latter modifies the logging behaviour for each of the categories at the console instance of an application.

Levels

All *HPX* logging output has seven different logging levels. These levels can be set explicitly or through environment variables in the main *HPX* ini file as shown below. The logging levels and their associated integral values are shown in the table below, ordered from most verbose to least verbose. By default, all *HPX* logs are set to 0, e.g., all logging output is disabled by default.

Table 2.6: Logging levels

Logging level	Integral value
<debug>	5
<info>	4
<warning>	3
<error>	2
<fatal>	1
No logging	0

Tip: The easiest way to enable logging output is to set the environment variable corresponding to the logging category to an integral value as described in the table above. For instance, setting `HPX_LOGLEVEL=5` will enable full logging output for the general category. Please note that the syntax and means of setting environment variables varies between operating systems.

Configuration

Logs will be saved to destinations as configured by the user. By default, logging output is saved on the console instance of an application to `hpx.<CATEGORY>.<PID>.lo` (where `CATEGORY` and `PID` are placeholders for the category shortcut and the OS process id). The output for the general logging category is saved to `hpx.<PID>.log`. The default settings for the general logging category are shown here (the syntax is described in the section [The HPX ini file format](#)):

```
[hpx.logging]
level = ${HPX_LOGLEVEL:0}
destination = ${HPX_LOGDESTINATION:console}
format = ${HPX_LOGFORMAT:(T%locality%/%hpxthread%.%hpxphase%/%hpxcomponent%) P
    ↪%parentloc%/%hpxparent%.%hpxparentphase% %time%($hh:$mm:$ss.$milis) [%idx%] |\\n}
```

The logging level is taken from the environment variable `HPX_LOGLEVEL` and defaults to zero, e.g., no logging. The default logging destination is read from the environment variable `HPX_LOGDESTINATION`. On any of the localities it defaults to `console`, which redirects all generated logging output to the console instance of an application. The following table lists the possible destinations for any logging output. It is possible to specify more than one destination separated by whitespace.

Table 2.7: Logging destinations

Logging destination	Description
file(<filename>)	Directs all output to a file with the given <filename>.
cout	Directs all output to the local standard output of the application instance on this <i>locality</i> .
cerr	Directs all output to the local standard error output of the application instance on this <i>locality</i> .
console	Directs all output to the console instance of the application. The console instance has its logging destinations configured separately.
android_log	Directs all output to the (Android) system log (available on Android systems only).

The logging format is read from the environment variable `HPX_LOGFORMAT`, and it defaults to a complex format description. This format consists of several placeholder fields (for instance `%locality%`), which will be replaced by concrete values when the logging output is generated. All other information is transferred verbatim to the output. The table below describes the available field placeholders. The separator character `|` separates the logging message prefix formatted as shown and the actual log message which will replace the separator.

Table 2.8: Available field placeholders

Name	Description
<code>locality</code>	The id of the <code>locality</code> on which the logging message was generated.
<code>hpxthread</code>	The id of the <code>HPX</code> thread generating this logging output.
<code>hpxphase</code>	The phase ⁴¹ of the <code>HPX</code> thread generating this logging output.
<code>hpxcomponent</code>	The local virtual address of the component which the current <code>HPX</code> thread is accessing.
<code>parentloc</code>	The id of the <code>locality</code> where the <code>HPX</code> thread was running that initiated the current <code>HPX</code> thread. The current <code>HPX</code> thread is generating this logging output.
<code>hpxparent</code>	The id of the <code>HPX</code> thread that initiated the current <code>HPX</code> thread. The current <code>HPX</code> thread is generating this logging output.
<code>hpxparentphase</code>	The phase of the <code>HPX</code> thread when it initiated the current <code>HPX</code> thread. The current <code>HPX</code> thread is generating this logging output.
<code>time</code>	The time stamp for this logging output line as generated by the source <code>locality</code> .
<code>idx</code>	The sequence number of the logging output line as generated on the source <code>locality</code> .
<code>osthread</code>	The sequence number of the OS thread that executes the current <code>HPX</code> thread.

Note: Not all of the field placeholder may be expanded for all generated logging output. If no value is available for a particular field, it is replaced with a sequence of `'-'` characters.

Here is an example line from a logging output generated by one of the `HPX` examples (please note that this is generated on a single line, without a line break):

```
(T00000000/0000000002d46f90.01/00000000009ebc10) P-----/0000000002d46f80.02 17:49.
˓→37.320 [00000000000004d]
    <info> [RT] successfully created component {000000100ff0001, 0000000000030002}_
˓→of type: component_barrier[7(3)]
```

The default settings for the general logging category on the console is shown here:

```
[hpx.logging.console]
level = ${HPX_LOGLEVEL:${hpx.logging.level}}
destination = ${HPX_CONSOLE_LOGDESTINATION:file(hpx.${system.pid}.log)}
format = ${HPX_CONSOLE_LOGFORMAT:|}
```

These settings define how the logging is customized once the logging output is received by the console instance of an application. The logging level is read from the environment variable `HPX_LOGLEVEL` (as set for the console instance of the application). The level defaults to the same values as the corresponding settings in the general logging configuration shown before. The destination on the console instance is set to be a file that's name is generated based on its OS process id. Setting the environment variable `HPX_CONSOLE_LOGDESTINATION` allows customization of the naming scheme for the output file. The logging format is set to leave the original logging output unchanged, as received from one of the localities the application runs on.

⁴¹ The phase of a `HPX`-thread counts how often this thread has been activated.

HPX Command Line Options

The predefined command line options for any application using `hpx::init` are described in the following subsections.

HPX options (allowed on command line only)

`--hpx:help`

Print out program usage (default: this message). Possible values: `full` (additionally prints options from components).

`--hpx:version`

Print out HPX version and copyright information.

`--hpx:info`

Print out HPX configuration information.

`--hpx:options-file arg`

Specify a file containing command line options (alternatively: @filepath).

HPX options (additionally allowed in an options file)

`--hpx:worker`

Run this instance in worker mode.

`--hpx:console`

Run this instance in console mode.

`--hpx:connect`

Run this instance in worker mode, but connecting late.

`--hpx:run-agas-server`

Run `AGAS` server as part of this runtime instance.

`--hpx:run-hpx-main`

Run the `hpx_main` function, regardless of `locality` mode.

`--hpx:hpx arg`

The IP address the HPX parcelport is listening on, expected format: `address:port` (default: `127.0.0.1:7910`).

`--hpx:agas arg`

The IP address the `AGAS` root server is running on, expected format: `address:port` (default: `127.0.0.1:7910`).

`--hpx:run-agas-server-only`

Run only the `AGAS` server.

`--hpx:nodofile arg`

The file name of a node file to use (list of nodes, one node name per line and core).

`--hpx:nodes arg`

The (space separated) list of the nodes to use (usually this is extracted from a node file).

`--hpx:endnodes`

This can be used to end the list of nodes specified using the option `--hpx:nodes`.

`--hpx:ifsuffix arg`

Suffix to append to host names in order to resolve them to the proper network interconnect.

--hpx:ifprefix arg
Prefix to prepend to host names in order to resolve them to the proper network interconnect.

--hpx:iftransform arg
Sed-style search and replace (s/search/replace/) used to transform host names to the proper network interconnect.

--hpx:localities arg
The number of localities to wait for at application startup (default: 1).

--hpx:node arg
Number of the node this *locality* is run on (must be unique).

--hpx:ignore-batch-env
Ignore batch environment variables.

--hpx:expect-connecting-localities
This *locality* expects other localities to dynamically connect (this is implied if the number of initial localities is larger than 1).

--hpx:pu-offset
The first processing unit this instance of *HPX* should be run on (default: 0).

--hpx:pu-step
The step between used processing unit numbers for this instance of *HPX* (default: 1).

--hpx:threads arg
The number of operating system threads to spawn for this *HPX locality*. Possible values are: numeric values 1, 2, 3 and so on, *all* (which spawns one thread per processing unit, includes hyperthreads), or *cores* (which spawns one thread per core) (default: *cores*).

--hpx:cores arg
The number of cores to utilize for this *HPX locality* (default: *all*, i.e., the number of cores is based on the number of threads *--hpx:threads* assuming *--hpx:bind=compact*).

--hpx:affinity arg
The affinity domain the OS threads will be confined to, possible values: *pu*, *core*, *numa*, *machine* (default: *pu*).

--hpx:bind arg
he detailed affinity description for the OS threads, see *More details about HPX command line options* for a detailed description of possible values. Do not use with *--hpx:pu-step*, *--hpx:pu-offset* or *--hpx:affinity* options. Implies *--hpx:numa-sensitive* (*--hpx:bind=none*) disables defining thread affinities).

--hpx:use-process-mask
Use the process mask to restrict available hardware resources (implies *--hpx:ignore-batch-env*).

--hpx:print-bind
Print to the console the bit masks calculated from the arguments specified to all *--hpx:bind* options.

--hpx:queuing arg
The queue scheduling policy to use. Options are *local*, *local-priority-fifo*, *local-priority-lifo*, *static*, *static-priority*, *abp-priority-fifo* and *abp-priority-lifo* (default: *local-priority-fifo*).

--hpx:high-priority-threads arg
The number of operating system threads maintaining a high priority queue (default: number of OS threads), valid for *--hpx:queuing=abp-priority*, *--hpx:queuing=static-priority* and *--hpx:queuing=local-priority* only.

--hpx:numa-sensitive
Makes the scheduler NUMA sensitive.

HPX configuration options

--hpx:app-config arg
Load the specified application configuration (ini) file.
--hpx:config arg
Load the specified HPX configuration (ini) file.
--hpx:ini arg
Add a configuration definition to the default runtime configuration.
--hpx:exit
Exit after configuring the runtime.

HPX debugging options

--hpx:list-symbolic-names
List all registered symbolic names after startup.
--hpx:list-component-types
List all dynamic component types after startup.
--hpx:dump-config-initial
Print the initial runtime configuration.
--hpx:dump-config
Print the final runtime configuration.
--hpx:debug-hpx-log [arg]
Enable all messages on the HPX log channel and send all HPX logs to the target destination (default: cout).
--hpx:debug-agas-log [arg]
Enable all messages on the AGAS log channel and send all AGAS logs to the target destination (default: cout).
--hpx:debug-parcel-log [arg]
Enable all messages on the parcel transport log channel and send all parcel transport logs to the target destination (default: cout).
--hpx:debug-timing-log [arg]
Enable all messages on the timing log channel and send all timing logs to the target destination (default: cout).
--hpx:debug-app-log [arg]
Enable all messages on the application log channel and send all application logs to the target destination (default: cout).
--hpx:debug-clp
Debug command line processing.
--hpx:attach-debugger arg
Wait for a debugger to be attached, possible arg values: startup or exception (default: startup)

HPX options related to performance counters

--hpx:print-counter

Print the specified performance counter either repeatedly and/or at the times specified by `--hpx:print-counter-at` (see also option `--hpx:print-counter-interval`).

--hpx:print-counter-reset

Print the specified performance counter either repeatedly and/or at the times specified by `--hpx:print-counter-at`. Reset the counter after the value is queried (see also option `--hpx:print-counter-interval`).

--hpx:print-counter-interval

Print the performance counter(s) specified with `--hpx:print-counter` repeatedly after the time interval (specified in milliseconds), (default: 0, which means print once at shutdown).

--hpx:print-counter-destination

Print the performance counter(s) specified with `--hpx:print-counter` to the given file (default: console).

--hpx:list-counters

List the names of all registered performance counters, possible values: minimal (prints counter name skeletons), full (prints all available counter names).

--hpx:list-counter-infos

List the description of all registered performance counters, possible values: minimal (prints info for counter name skeletons), full (prints all available counter infos).

--hpx:print-counter-format

Print the performance counter(s) specified with `--hpx:print-counter`. Possible formats in CSV include a format with a header or without any header (see option `--hpx:no-csv-header`). Possible values: csv (prints counter values in CSV format with full names as header), csv-short (prints counter values in CSV format with short names provided with `--hpx:print-counter` as `--hpx:print-counter` shortname, full-countername

--hpx:no-csv-header

Print the performance counter(s) specified with `--hpx:print-counter` and csv or csv-short format specified with `--hpx:print-counter-format` without header.

--hpx:print-counter-at arg

Print the performance counter(s) specified with `--hpx:print-counter` (or `--hpx:print-counter-reset`) at the given point in time, possible argument values: startup, shutdown (default), noshutdown.

--hpx:reset-counters

Reset all performance counter(s) specified with `--hpx:print-counter` after they have been evaluated.

--hpx:print-counters-locally

Each `locality` prints only its own local counters. If this is used with `--hpx:print-counter-destination=<file>`, the code will append a "`.<locality_id>`" to the file name in order to avoid clashes between localities.

Command line argument shortcuts

Additionally, the following shortcuts are available from every *HPX* application.

Table 2.9: Predefined command line option shortcuts

Shortcut option	Equivalent long option
-a	--hpx:agas
-c	--hpx:console
-h	--hpx:help
-I	--hpx:ini
-l	--hpx:localities
-p	--hpx:app-config
-q	--hpx:queuing
-r	--hpx:run-agas-server
-t	--hpx:threads
-v	--hpx:version
-w	--hpx:worker
-x	--hpx:hpx
-0	--hpx:node=0
-1	--hpx:node=1
-2	--hpx:node=2
-3	--hpx:node=3
-4	--hpx:node=4
-5	--hpx:node=5
-6	--hpx:node=6
-7	--hpx:node=7
-8	--hpx:node=8
-9	--hpx:node=9

It is possible to define your own shortcut options. In fact, all of the shortcuts listed above are pre-defined using the technique described here. Also, it is possible to redefine any of the pre-defined shortcuts to expand differently as well.

Shortcut options are obtained from the internal configuration database. They are stored as key-value properties in a special properties section named `hpx.commandline`. You can define your own shortcuts by adding the corresponding definitions to one of the ini configuration files as described in the section [Configuring HPX applications](#). For instance, in order to define a command line shortcut `--p`, which should expand to `--hpx:print-counter`, the following configuration information needs to be added to one of the ini configuration files:

```
[hpx.commandline.aliases]
--pc = --hpx:print-counter
```

Note: Any arguments for shortcut options passed on the command line are retained and passed as arguments to the corresponding expanded option. For instance, given the definition above, the command line option:

```
--pc=/threads{locality#0/total}/count/cumulative
```

would be expanded to:

```
--hpx:print-counter=/threads{locality#0/total}/count/cumulative
```

Important: Any shortcut option should either start with a single '-' or with two '--' characters. Shortcuts

starting with a single '-' are interpreted as short options (i.e., everything after the first character following the '-' is treated as the argument). Shortcuts starting with '--' are interpreted as long options. No other shortcut formats are supported.

Specifying options for single localities only

For runs involving more than one *locality*, it is sometimes desirable to supply specific command line options to single localities only. When the *HPX* application is launched using a scheduler (like PBS; for more details see section [How to use HPX applications with PBS](#)), specifying dedicated command line options for single localities may be desirable. For this reason all of the command line options that have the general format `--hpx:<some_key>` can be used in a more general form: `--hpx:<N>:<some_key>`, where <N> is the number of the *locality* this command line option will be applied to; all other localities will simply ignore the option. For instance, the following PBS script passes the option `--hpx:pu-offset=4` to the *locality* '1' only.

```
#!/bin/bash
#
#PBS -l nodes=2:ppn=4

APP_PATH=~/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:1:pu-offset=4 --hpx:nodes=`cat $PBS_NODEFILE`
```

Caution: If the first application specific argument (inside `$APP_OPTIONS`) is a non-option (i.e., does not start with a - or a --), then it must be placed before the option `--hpx:nodes`, which, in this case, should be the last option on the command line.

Alternatively, use the option `--hpx:endnodes` to explicitly mark the end of the list of node names:

```
$ pbsdsh -u $APP_PATH --hpx:1:pu-offset=4 --hpx:nodes=`cat $PBS_NODEFILE` --
→hpx:endnodes $APP_OPTIONS
```

More details about *HPX* command line options

This section documents the following list of the command line options in more detail:

- *The command line option --hpx:bind*

The command line option `--hpx:bind`

This command line option allows one to specify the required affinity of the *HPX* worker threads to the underlying processing units. As a result the worker threads will run only on the processing units identified by the corresponding bind specification. The affinity settings are to be specified using `--hpx:bind=<BINDINGS>`, where <BINDINGS> have to be formatted as described below.

In addition to the syntax described below, one can use `--hpx:bind=none` to disable all binding of any threads to a particular core. This is mostly supported for debugging purposes.

The specified affinities refer to specific regions within a machine hardware topology. In order to understand the hardware topology of a particular machine, it may be useful to run the `lstopo` tool, which is part of Portable Hardware

Locality (HWLOC), to see the reported topology tree. Seeing and understanding a topology tree will definitely help in understanding the concepts that are discussed below.

Affinities can be specified using hwloc tuples. Tuples of hwloc *objects* and associated *indexes* can be specified in the form `object:index, object:index-index` or `object:index, ..., index`. Hwloc objects represent types of mapped items in a topology tree. Possible values for objects are socket, numanode, core and pu (processing unit). Indexes are non-negative integers that specify a unique physical object in a topology tree using its logical sequence number.

Chaining multiple tuples together in the more general form `object1:index1[.object2:index2[...]]` is permissible. While the first tuple's object may appear anywhere in the topology, the Nth tuple's object must have a shallower topology depth than the (N+1)th tuple's object. Put simply: as you move right in a tuple chain, objects must go deeper in the topology tree. Indexes specified in chained tuples are relative to the scope of the parent object. For example, `socket:0.core:1` refers to the second core in the first socket (all indices are zero based).

Multiple affinities can be specified using several `--hpx:bind` command line options or by appending several affinities separated by a '`;`'. By default, if multiple affinities are specified, they are added.

"`all`" is a special affinity consisting in the entire current topology.

Note: All "names" in an affinity specification, such as `thread`, `socket`, `numanode`, `pu` or `all`, can be abbreviated. Thus, the affinity specification `threads:0-3=socket:0.core:1.pu:1` is fully equivalent to its shortened form `t:0-3=s:0.c:1.p:1`.

Here is a full grammar describing the possible format of mappings:

```
mappings      ::= distribution | mapping (";" mapping)*
distribution   ::= "compact" | "scatter" | "balanced" | "numa-balanced"
mapping        ::= thread_spec "=" pu_specs
thread_spec    ::= "thread:" range_specs
pu_specs       ::= pu_spec (".") pu_spec)*
pu_spec        ::= type ":" range_specs | "~" pu_spec
range_specs    ::= range_spec ("," range_spec)*
range_spec     ::= int | int "-" int | "all"
type           ::= "socket" | "numanode" | "core" | "pu"
```

The following example assumes a system with at least 4 cores, where each core has more than 1 processing unit (hardware threads). Running `hello_world_distributed` with 4 OS threads (on 4 processing units), where each of those threads is bound to the first processing unit of each of the cores, can be achieved by invoking:

```
$ hello_world_distributed -t4 --hpx:bind=thread:0-3=core:0-3.pu:0
```

Here, `thread:0-3` specifies the OS threads used to define affinity bindings, and `core:0-3.pu:0` defines that for each of the cores (`core:0-3`) only their first processing unit `pu:0` should be used.

Note: The command line option `--hpx:print-bind` can be used to print the bitmasks generated from the affinity mappings as specified with `--hpx:bind`. For instance, on a system with hyperthreading enabled (i.e. 2 processing units per core), the command line:

```
$ hello_world_distributed -t4 --hpx:bind=thread:0-3=core:0-3.pu:0 --hpx:print-bind
```

will cause this output to be printed:

```

0: PU L#0(P#0), Core L#0, Socket L#0, Node L#0(P#0)
1: PU L#2(P#2), Core L#1, Socket L#0, Node L#0(P#0)
2: PU L#4(P#4), Core L#2, Socket L#0, Node L#0(P#0)
3: PU L#6(P#6), Core L#3, Socket L#0, Node L#0(P#0)

```

where each bit in the bitmasks corresponds to a processing unit the listed worker thread will be bound to run on.

The difference between the four possible predefined distribution schemes (compact, scatter, balanced and numa-balanced) is best explained with an example. Imagine that we have a system with 4 cores and 4 hardware threads per core on 2 sockets. If we place 8 threads the assignments produced by the compact, scatter, balanced and numa-balanced types are shown in the figure below. Notice that compact does not fully utilize all the cores in the system. For this reason it is recommended that applications are run using the scatter or balanced/numa-balanced options in most cases.

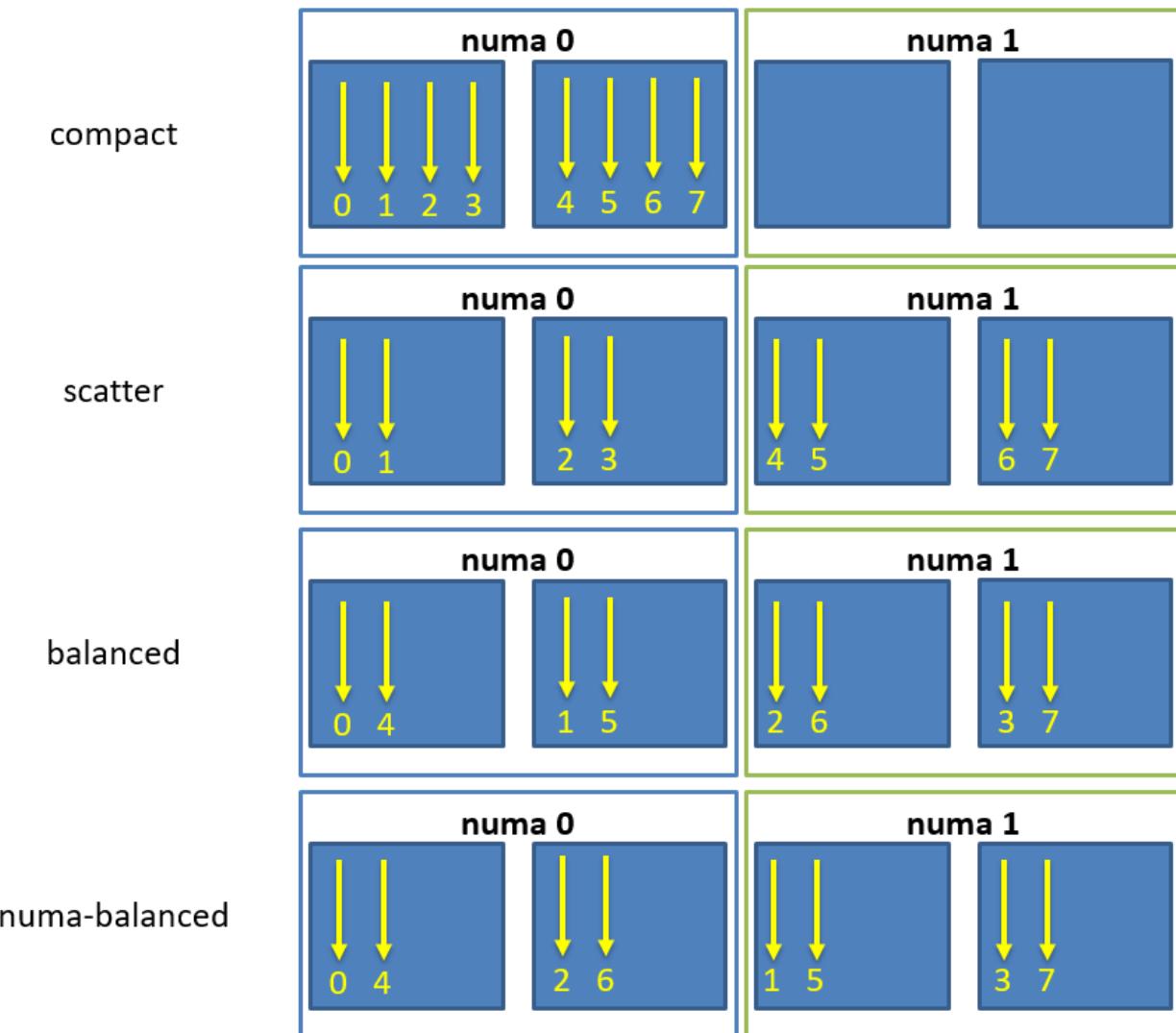


Fig. 2.7: Schematic of thread affinity type distributions.

In addition to the predefined distributions it is possible to restrict the resources used by *HPX* to the process CPU

mask. The CPU mask is typically set by e.g. MPI⁴⁰ and batch environments. Using the command line option `--hpx:use-process-mask` makes HPX act as if only the processing units in the CPU mask are available for use by HPX. The number of threads is automatically determined from the CPU mask. The number of threads can still be changed manually using this option, but only to a number less than or equal to the number of processing units in the CPU mask. The option `--hpx:print-bind` is useful in conjunction with `--hpx:use-process-mask` to make sure threads are placed as expected.

2.3.8 Writing single-node HPX applications

HPX is a C++ Standard Library for Concurrency and Parallelism. This means that it implements all of the corresponding facilities as defined by the C++ Standard. Additionally, HPX implements functionalities proposed as part of the ongoing C++ standardization process. This section focuses on the features available in HPX for parallel and concurrent computation on a single node, although many of the features presented here are also implemented to work in the distributed case.

Using LCOs

Lightweight Control Objects (LCOs) provide synchronization for HPX applications. Most of them are familiar from other frameworks, but a few of them work in slightly different ways adapted to HPX. The following synchronization objects are available in HPX:

1. future
2. queue
3. object_semaphore
4. barrier

Channels

Channels combine communication (the exchange of a value) with synchronization (guaranteeing that two calculations (tasks) are in a known state). A channel can transport any number of values of a given type from a sender to a receiver:

```
hpx::lcos::local::channel<int> c;
hpx::future<int> f = c.get();
HPX_ASSERT(!f.is_ready());
c.set(42);
HPX_ASSERT(f.is_ready());
std::cout << f.get() << std::endl;
```

Channels can be handed to another thread (or in case of channel components, to other localities), thus establishing a communication channel between two independent places in the program:

```
void do_something(hpx::lcos::local::receive_channel<int> c,
                  hpx::lcos::local::send_channel<> done)
{
    // prints 43
    std::cout << c.get(hpx::launch::sync) << std::endl;
    // signal back
    done.set();
}
```

(continues on next page)

⁴⁰ https://en.wikipedia.org/wiki/Message_Passing_Interface

(continued from previous page)

```
void send_receive_channel()
{
    hpx::lcos::local::channel<int> c;
    hpx::lcos::local::channel<> done;

    hpx::apply(&do_something, c, done);

    // send some value
    c.set(43);
    // wait for thread to be done
    done.get().wait();
}
```

Note how `hpx::lcos::local::channel::get` without any arguments returns a future which is ready when a value has been set on the channel. The launch policy `hpx::launch::sync` can be used to make `hpx::lcos::local::channel::get` block until a value is set and return the value directly.

A channel component is created on one *locality* and can be sent to another *locality* using an action. This example also demonstrates how a channel can be used as a range of values:

```
// channel components need to be registered for each used type (not needed
// for hpx::lcos::local::channel)
HPX_REGISTER_CHANNEL(double)

void channel_sender(hpx::lcos::channel<double> c)
{
    for (double d : c)
        hpx::cout << d << std::endl;
}
HPX_PLAIN_ACTION(channel_sender)

void channel()
{
    // create the channel on this locality
    hpx::lcos::channel<double> c(hpx::find_here());

    // pass the channel to a (possibly remote invoked) action
    hpx::apply(channel_sender_action(), hpx::find_here(), c);

    // send some values to the receiver
    std::vector<double> v = {1.2, 3.4, 5.0};
    for (double d : v)
        c.set(d);

    // explicitly close the communication channel (implicit at destruction)
    c.close();
}
```

Composable guards

Composable guards operate in a manner similar to locks, but are applied only to asynchronous functions. The guard (or guards) is automatically locked at the beginning of a specified task and automatically unlocked at the end. Because guards are never added to an existing task's execution context, the calling of guards is freely composable and can never deadlock.

To call an application with a single guard, simply declare the guard and call run_guarded() with a function (task):

```
hpx::lcos::local::guard gu;
run_guarded(gu,task);
```

If a single method needs to run with multiple guards, use a guard set:

```
boost::shared<hpx::lcos::local::guard> gu1(new hpx::lcos::local::guard());
boost::shared<hpx::lcos::local::guard> gu2(new hpx::lcos::local::guard());
gs.add(*gu1);
gs.add(*gu2);
run_guarded(gs,task);
```

Guards use two atomic operations (which are not called repeatedly) to manage what they do, so overhead should be extremely low. The following guards are available in *HPX*:

1. conditional_trigger
2. counting_semaphore
3. dataflow
4. event
5. mutex
6. once
7. recursive_mutex
8. spinlock
9. spinlock_no_backoff
10. trigger

Extended facilities for futures

Concurrency is about both decomposing and composing the program from the parts that work well individually and together. It is in the composition of connected and multicore components where today's C++ libraries are still lacking.

The functionality of `std::future` offers a partial solution. It allows for the separation of the initiation of an operation and the act of waiting for its result; however, the act of waiting is synchronous. In communication-intensive code this act of waiting can be unpredictable, inefficient and simply frustrating. The example below illustrates a possible synchronous wait using futures:

```
#include <future>
using namespace std;
int main()
{
    future<int> f = async([]() { return 123; });
    int result = f.get(); // might block
}
```

For this reason, *HPX* implements a set of extensions to `std::future` (as proposed by [__cpp11_n4107__](#)). This proposal introduces the following key asynchronous operations to `hpx::future`, `hpx::shared_future` and `hpx::async`, which enhance and enrich these facilities.

Table 2.11: Facilities extending `std::future`

Facility	Description
<code>hpx::future<T>::then</code>	In asynchronous programming, it is very common for one asynchronous operation, on completion, to invoke a second operation and pass data to it. The current C++ standard does not allow one to register a continuation to a future. With <code>then</code> , instead of waiting for the result, a continuation is “attached” to the asynchronous operation, which is invoked when the result is ready. Continuations registered using <code>then</code> function will help to avoid blocking waits or wasting threads on polling, greatly improving the responsiveness and scalability of an application.
<code>unwrapping constructor for hpx::future</code>	In some scenarios, you might want to create a future that returns another future, resulting in nested futures. Although it is possible to write code to unwrap the outer future and retrieve the nested future and its result, such code is not easy to write because users must handle exceptions and it may cause a blocking call. Unwrapping can allow users to mitigate this problem by doing an asynchronous call to unwrap the outermost future.
<code>hpx::future<T>::try_then</code>	There are some situations where a <code>get()</code> call on a future may not be a blocking call, or is only a blocking call under certain circumstances. This function gives the ability to test for early completion and allows us to avoid associating a continuation, which needs to be scheduled with some non-trivial overhead and near-certain loss of cache efficiency.
<code>hpx::make_ready_future<T></code>	Some functions may know the value at the point of construction. In these cases the value is immediately available, but needs to be returned as a future. By using <code>hpx::make_ready_future</code> a future can be created that holds a pre-computed result in its shared state. In the current standard it is non-trivial to create a future directly from a value. First a promise must be created, then the promise is set, and lastly the future is retrieved from the promise. This can now be done with one operation.

The standard also omits the ability to compose multiple futures. This is a common pattern that is ubiquitous in other asynchronous frameworks and is absolutely necessary in order to make C++ a powerful asynchronous programming language. Not including these functions is synonymous to Boolean algebra without AND/OR.

In addition to the extensions proposed by [N4313⁴²](#), *HPX* adds functions allowing users to compose several futures in a more flexible way.

⁴² <http://wg21.link/n4313>

Table 2.12: Facilities for composing `hpx::futures`

Facility	Description	Comment
<code>hpx::when_any</code> , <code>hpx::when_any_n</code>	Asynchronously wait for at least one of multiple future or shared_future objects to finish.	N4313⁴³ , ..._n versions are <i>HPX</i> only
<code>hpx::wait_any</code> , <code>hpx::wait_any_n</code>	Synchronously wait for at least one of multiple future or shared_future objects to finish.	<i>HPX</i> only
<code>hpx::when_all</code> , <code>hpx::when_all_n</code>	Asynchronously wait for all future and shared_future objects to finish.	N4313⁴⁴ , ..._n versions are <i>HPX</i> only
<code>hpx::wait_all</code> , <code>hpx::wait_all_n</code>	Synchronously wait for all future and shared_future objects to finish.	<i>HPX</i> only
<code>hpx::when_some</code> , <code>hpx::when_some_n</code>	Asynchronously wait for multiple future and shared_future objects to finish.	<i>HPX</i> only
<code>hpx::wait_some</code> , <code>hpx::wait_some_n</code>	Synchronously wait for multiple future and shared_future objects to finish.	<i>HPX</i> only
<code>hpx::when_each</code>	Asynchronously wait for multiple future and shared_future objects to finish and call a function for each of the future objects as soon as it becomes ready.	<i>HPX</i> only
<code>hpx::wait_each</code> , <code>hpx::wait_each_n</code>	Synchronously wait for multiple future and shared_future objects to finish and call a function for each of the future objects as soon as it becomes ready.	<i>HPX</i> only

High level parallel facilities

In preparation for the upcoming C++ Standards, there are currently several proposals targeting different facilities supporting parallel programming. *HPX* implements (and extends) some of those proposals. This is well aligned with our strategy to align the APIs exposed from *HPX* with current and future C++ Standards.

At this point, *HPX* implements several of the C++ Standardization working papers, most notably [N4409⁴⁵](#) (Working Draft, Technical Specification for C++ Extensions for Parallelism), [N4411⁴⁶](#) (Task Blocks), and [N4406⁴⁷](#) (Parallel Algorithms Need Executors).

Using parallel algorithms

A parallel algorithm is a function template described by this document which is declared in the (inline) namespace `hpx::parallel::v1`.

Note: For compilers that do not support inline namespaces, all of the namespace `v1` is imported into the namespace `hpx::parallel`. The effect is similar to what inline namespaces would do, namely all names defined in `hpx::parallel::v1` are accessible from the namespace `hpx::parallel` as well.

All parallel algorithms are very similar in semantics to their sequential counterparts (as defined in the namespace `std`) with an additional formal template parameter named `ExecutionPolicy`. The execution policy is generally

⁴³ <http://wg21.link/n4313>

⁴⁴ <http://wg21.link/n4313>

⁴⁵ <http://wg21.link/n4409>

⁴⁶ <http://wg21.link/n4411>

⁴⁷ <http://wg21.link/n4406>

passed as the first argument to any of the parallel algorithms and describes the manner in which the execution of these algorithms may be parallelized and the manner in which they apply user-provided function objects.

The applications of function objects in parallel algorithms invoked with an execution policy object of type `hpx::execution::sequenced_policy` or `hpx::execution::sequenced_task_policy` execute in sequential order. For `hpx::execution::sequenced_policy` the execution happens in the calling thread.

The applications of function objects in parallel algorithms invoked with an execution policy object of type `hpx::execution::parallel_policy` or `hpx::execution::parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and are indeterminately sequenced within each thread.

Important: It is the caller's responsibility to ensure correctness, such as making sure that the invocation does not introduce data races or deadlocks.

The applications of function objects in parallel algorithms invoked with an execution policy of type `hpx::execution::parallel_unsequenced_policy` is, in *HPX*, equivalent to the use of the execution policy `hpx::execution::parallel_policy`.

Algorithms invoked with an execution policy object of type `hpx::parallel::v1::execution_policy` execute internally as if invoked with the contained execution policy object. No exception is thrown when an `hpx::parallel::v1::execution_policy` contains an execution policy of type `hpx::execution::sequenced_task_policy` or `hpx::execution::parallel_task_policy` (which normally turn the algorithm into its asynchronous version). In this case the execution is semantically equivalent to the case of passing a `hpx::execution::sequenced_policy` or `hpx::execution::parallel_policy` contained in the `hpx::parallel::v1::execution_policy` object respectively.

Parallel exceptions

During the execution of a standard parallel algorithm, if temporary memory resources are required by any of the algorithms and no memory is available, the algorithm throws a `std::bad_alloc` exception.

During the execution of any of the parallel algorithms, if the application of a function object terminates with an uncaught exception, the behavior of the program is determined by the type of execution policy used to invoke the algorithm:

- If the execution policy object is of type `hpx::execution::parallel_unsequenced_policy`, `hpx::terminate` shall be called.
- If the execution policy object is of type `hpx::execution::sequenced_policy`, `hpx::execution::sequenced_task_policy`, `hpx::execution::parallel_policy`, or `hpx::execution::parallel_task_policy`, the execution of the algorithm terminates with an `hpx::exception_list` exception. All uncaught exceptions thrown during the application of user-provided function objects shall be contained in the `hpx::exception_list`.

For example, the number of invocations of the user-provided function object in `for_each` is unspecified. When `hpx::parallel::v1::for_each` is executed sequentially, only one exception will be contained in the `hpx::exception_list` object.

These guarantees imply that, unless the algorithm has failed to allocate memory and terminated with `std::bad_alloc`, all exceptions thrown during the execution of the algorithm are communicated to the caller. It is unspecified whether an algorithm implementation will “forge ahead” after encountering and capturing a user exception.

The algorithm may terminate with the `std::bad_alloc` exception even if one or more user-provided function objects have terminated with an exception. For example, this can happen when an algorithm fails to allocate memory

while creating or adding elements to the `hpx::exception_list` object.

Parallel algorithms

HPX provides implementations of the following parallel algorithms:

Table 2.13: Non-modifying parallel algorithms (in header: <hpx/algorithm.hpp>)

Name	Description	In header	Algorithm page at cppreference.com
hpx::adjacent_find	Computes the differences between adjacent elements in a range.	<hpx/algorithm.hpp>	adjacent_find ⁴⁸
hpx::all_of	Checks if a predicate is true for all of the elements in a range.	<hpx/algorithm.hpp>	all_any_none_of ⁴⁹
hpx::any_of	Checks if a predicate is true for any of the elements in a range.	<hpx/algorithm.hpp>	all_any_none_of ⁵⁰
hpx::count	Returns the number of elements equal to a given value.	<hpx/algorithm.hpp>	count ⁵¹
hpx::count_if	Returns the number of elements satisfying a specific criteria.	<hpx/algorithm.hpp>	count_if ⁵²
hpx::equal	Determines if two sets of elements are the same.	<hpx/algorithm.hpp>	equal ⁵³
hpx::find	Finds the first element equal to a given value.	<hpx/algorithm.hpp>	find ⁵⁴
hpx::find_end	Finds the last sequence of elements in a certain range.	<hpx/algorithm.hpp>	find_end ⁵⁵
hpx::find_first_of	Searches for any one of a set of elements.	<hpx/algorithm.hpp>	find_first_of ⁵⁶
hpx::find_if	Finds the first element satisfying a specific criteria.	<hpx/algorithm.hpp>	find_if ⁵⁷
hpx::find_if_not	Finds the first element not satisfying a specific criteria.	<hpx/algorithm.hpp>	find_if_not ⁵⁸
hpx::for_each	Applies a function to a range of elements.	<hpx/algorithm.hpp>	for_each ⁵⁹
hpx::for_each_n	Applies a function to a number of elements.	<hpx/algorithm.hpp>	for_each_n ⁶⁰
hpx::lexicographical_compare	Checks if a range of values is lexicographically less than another range of values.	<hpx/algorithm.hpp>	lexicographical_compare ⁶¹
hpx::parallel::v1::mismatch	Finds the first position where two ranges differ.	<hpx/algorithm.hpp>	mismatch ⁶²
hpx::none_of	Checks if a predicate is true for none of the elements in a range.	<hpx/algorithm.hpp>	all_any_none_of ⁶³
hpx::search	Searches for a range of elements.	<hpx/algorithm.hpp>	search ⁶⁴
2.3. Manual		hpp>	109
hpx::search_n	Searches for a number consecutive copies of an element in a range.	<hpx/algorithm.hpp>	search_n ⁶⁵

⁴⁸ http://en.cppreference.com/w/cpp/algorithm/adjacent_find
⁴⁹ http://en.cppreference.com/w/cpp/algorithm/all_any_none_of
⁵⁰ http://en.cppreference.com/w/cpp/algorithm/all_any_none_of
⁵¹ <http://en.cppreference.com/w/cpp/algorithm/count>
⁵² http://en.cppreference.com/w/cpp/algorithm/count_if
⁵³ <http://en.cppreference.com/w/cpp/algorithm/equal>
⁵⁴ <http://en.cppreference.com/w/cpp/algorithm/find>
⁵⁵ http://en.cppreference.com/w/cpp/algorithm/find_end
⁵⁶ http://en.cppreference.com/w/cpp/algorithm/find_first_of
⁵⁷ http://en.cppreference.com/w/cpp/algorithm/find_if
⁵⁸ http://en.cppreference.com/w/cpp/algorithm/find_if_not
⁵⁹ http://en.cppreference.com/w/cpp/algorithm/for_each
⁶⁰ http://en.cppreference.com/w/cpp/algorithm/for_each_n
⁶¹ http://en.cppreference.com/w/cpp/algorithm/lexicographical_compare
⁶² <http://en.cppreference.com/w/cpp/algorithm/mismatch>
⁶³ http://en.cppreference.com/w/cpp/algorithm/all_any_none_of
⁶⁴ <http://en.cppreference.com/w/cpp/algorithm/search>
⁶⁵ http://en.cppreference.com/w/cpp/algorithm/search_n

Table 2.14: Modifying parallel algorithms (In Header:
<hpx/algorithm.hpp>)

Name	Description	In header	Algorithm page at cppreference.com
<code>hpx::copy</code>	Copies a range of elements to a new location.	<code><hpx/algorithm.hpp></code>	<code>exclusive_scan</code> ⁶⁶
<code>hpx::copy_n</code>	Copies a number of elements to a new location.	<code><hpx/algorithm.hpp></code>	<code>copy_n</code> ⁶⁷
<code>hpx::copy_if</code>	Copies the elements from a range to a new location for which the given predicate is <code>true</code>	<code><hpx/algorithm.hpp></code>	<code>copy</code> ⁶⁸
<code>hpx::move</code>	Moves a range of elements to a new location.	<code><hpx/algorithm.hpp></code>	<code>move</code> ⁶⁹
<code>hpx::fill</code>	Assigns a range of elements a certain value.	<code><hpx/algorithm.hpp></code>	<code>fill</code> ⁷⁰
<code>hpx::fill_n</code>	Assigns a value to a number of elements.	<code><hpx/algorithm.hpp></code>	<code>fill_n</code> ⁷¹
<code>hpx::generate</code>	Saves the result of a function in a range.	<code><hpx/algorithm.hpp></code>	<code>generate</code> ⁷²
<code>hpx::generate_n</code>	Saves the result of N applications of a function.	<code><hpx/algorithm.hpp></code>	<code>generate_n</code> ⁷³
<code>hpx::remove</code>	Removes the elements from a range that are equal to the given value.	<code><hpx/algorithm.hpp></code>	<code>remove</code> ⁷⁴
<code>hpx::remove_if</code>	Removes the elements from a range that are equal to the given predicate is <code>false</code>	<code><hpx/algorithm.hpp></code>	<code>remove</code> ⁷⁵
<code>hpx::remove_copy</code>	Copies the elements from a range to a new location that are not equal to the given value.	<code><hpx/algorithm.hpp></code>	<code>remove_copy</code> ⁷⁶
<code>hpx::remove_copy_if</code>	Copies the elements from a range to a new location for which the given predicate is <code>false</code>	<code><hpx/algorithm.hpp></code>	<code>remove_copy</code> ⁷⁷
<code>hpx::replace</code>	Replaces all values satisfying specific criteria with another value.	<code><hpx/algorithm.hpp></code>	<code>replace</code> ⁷⁸
<code>hpx::replace_if</code>	Replaces all values satisfying specific criteria with another value.	<code><hpx/algorithm.hpp></code>	<code>replace</code> ⁷⁹
<code>hpx::replace_copy</code>	Copies a range, replacing elements satisfying specific criteria with another value.	<code><hpx/algorithm.hpp></code>	<code>replace_copy</code> ⁸⁰
<code>hpx::replace_copy_if</code>	Copies a range, replacing elements satisfying specific criteria with another value.	<code><hpx/algorithm.hpp></code>	<code>replace_copy</code> ⁸¹
<code>hpx::reverse</code>	Reverses the order elements in a range.	<code><hpx/algorithm.hpp></code>	<code>reverse</code> ⁸²
2.3. Manual		<code><hpx/algorithm.hpp></code>	111
<code>hpx::reverse_copy</code>	Creates a copy of a range that is reversed.	<code><hpx/algorithm.hpp></code>	<code>reverse_copy</code> ⁸³

Table 2.15: Set operations on sorted sequences (In Header:
<hpx/algorithm.hpp>)

Name	Description	In header	Algorithm page at cp-preference.com
<code>hpx::merge</code>	Merges two sorted ranges.	<code><hpx/algorithm.hpp></code>	merge⁹²
<code>hpx::inplace_merge</code>	Merges two ordered ranges in-place.	<code><hpx/algorithm.hpp></code>	inplace_merge⁹³
<code>hpx::includes</code>	Returns true if one set is a subset of another.	<code><hpx/algorithm.hpp></code>	includes⁹⁴
<code>hpx::set_difference</code>	Computes the difference between two sets.	<code><hpx/algorithm.hpp></code>	set_difference⁹⁵
<code>hpx::set_intersection</code>	Computes the intersection of two sets.	<code><hpx/algorithm.hpp></code>	set_intersection⁹⁶
<code>hpx::set_symmetric_difference</code>	Computes the symmetric difference between two sets.	<code><hpx/algorithm.hpp></code>	set_symmetric_difference⁹⁷
<code>hpx::set_union</code>	Computes the union of two sets.	<code><hpx/algorithm.hpp></code>	set_union⁹⁸

⁶⁶ http://en.cppreference.com/w/cpp/algorithm/exclusive_scan

⁶⁷ http://en.cppreference.com/w/cpp/algorithm/copy_n

⁶⁸ <http://en.cppreference.com/w/cpp/algorithm/copy>

⁶⁹ <http://en.cppreference.com/w/cpp/algorithm/move>

⁷⁰ <http://en.cppreference.com/w/cpp/algorithm/fill>

⁷¹ http://en.cppreference.com/w/cpp/algorithm/fill_n

⁷² <http://en.cppreference.com/w/cpp/algorithm/generate>

⁷³ http://en.cppreference.com/w/cpp/algorithm/generate_n

⁷⁴ <http://en.cppreference.com/w/cpp/algorithm/remove>

⁷⁵ <http://en.cppreference.com/w/cpp/algorithm/remove>

⁷⁶ http://en.cppreference.com/w/cpp/algorithm/remove_copy

⁷⁷ http://en.cppreference.com/w/cpp/algorithm/remove_copy

⁷⁸ <http://en.cppreference.com/w/cpp/algorithm/replace>

⁷⁹ <http://en.cppreference.com/w/cpp/algorithm/replace>

⁸⁰ http://en.cppreference.com/w/cpp/algorithm/replace_copy

⁸¹ http://en.cppreference.com/w/cpp/algorithm/replace_copy

⁸² <http://en.cppreference.com/w/cpp/algorithm/reverse>

⁸³ http://en.cppreference.com/w/cpp/algorithm/reverse_copy

⁸⁴ <http://en.cppreference.com/w/cpp/algorithm/rotate>

⁸⁵ http://en.cppreference.com/w/cpp/algorithm/rotate_copy

⁸⁶ http://en.cppreference.com/w/cpp/algorithm/shift_left

⁸⁷ http://en.cppreference.com/w/cpp/algorithm/shift_right

⁸⁸ http://en.cppreference.com/w/cpp/algorithm/swap_ranges

⁸⁹ <http://en.cppreference.com/w/cpp/algorithm/transform>

⁹⁰ <http://en.cppreference.com/w/cpp/algorithm/unique>

⁹¹ http://en.cppreference.com/w/cpp/algorithm/unique_copy

⁹² <http://en.cppreference.com/w/cpp/algorithm/merge>

⁹³ http://en.cppreference.com/w/cpp/algorithm/inplace_merge

⁹⁴ <http://en.cppreference.com/w/cpp/algorithm/includes>

⁹⁵ http://en.cppreference.com/w/cpp/algorithm/set_difference

⁹⁶ http://en.cppreference.com/w/cpp/algorithm/set_intersection

⁹⁷ http://en.cppreference.com/w/cpp/algorithm/set_symmetric_difference

⁹⁸ http://en.cppreference.com/w/cpp/algorithm/set_union

Table 2.16: Heap operations (In Header: <hpx/algorithm.hpp>)

Name	Description	In header	Algorithm page at cppreference.com
<code>hpx::is_heap</code>	Returns <code>true</code> if the range is max heap.	<hpx/algorithm.hpp>	<code>is_heap</code> ⁹⁹
<code>hpx::is_heap_until</code>	Returns the first element that breaks a max heap.	<hpx/algorithm.hpp>	<code>is_heap_until</code> ¹⁰⁰
<code>hpx::make_heap</code>	Constructs a max heap in the range [first, last).	<hpx/algorithm.hpp>	<code>make_heap</code> ¹⁰¹

Table 2.17: Minimum/maximum operations (In Header: <hpx/algorithm.hpp>)

Name	Description	In header	Algorithm page at cppreference.com
<code>hpx::max_element</code>	Returns the largest element in a range.	<hpx/algorithm.hpp>	<code>max_element</code> ¹⁰²
<code>hpx::min_element</code>	Returns the smallest element in a range.	<hpx/algorithm.hpp>	<code>min_element</code> ¹⁰³
<code>hpx::minmax_element</code>	Returns the smallest and the largest element in a range.	<hpx/algorithm.hpp>	<code>minmax_element</code> ¹⁰⁴

⁹⁹ http://en.cppreference.com/w/cpp/algorithm/is_heap¹⁰⁰ http://en.cppreference.com/w/cpp/algorithm/is_heap_until101 http://en.cppreference.com/w/cpp/algorithm/make_heap102 http://en.cppreference.com/w/cpp/algorithm/max_element103 http://en.cppreference.com/w/cpp/algorithm/min_element104 http://en.cppreference.com/w/cpp/algorithm/minmax_element

Table 2.18: Numeric Parallel Algorithms (In Header:
`<hpx/numeric.hpp>`)

Name	Description	In header	Algorithm page at cppreference.com
<code>hpx::adjacent_difference</code>	Calculates the difference between each element in an input range and the preceding element.	<code><hpx/numeric.hpp></code>	adjacent_difference ¹⁰⁵
<code>hpx::exclusive_scan</code>	Does an exclusive parallel scan over a range of elements.	<code><hpx/numeric.hpp></code>	exclusive_scan ¹⁰⁶
<code>hpx::reduce</code>	Sums up a range of elements.	<code><hpx/numeric.hpp></code>	reduce ¹⁰⁷
<code>hpx::inclusive_scan</code>	Does an inclusive parallel scan over a range of elements.	<code><hpx/algorithm.hpp></code>	inclusive_scan ¹⁰⁸
<code>hpx::parallel_reduce</code>	Performs an inclusive scan on consecutive elements with matching keys, with a reduction to output only the final sum for each key. The key sequence {1, 1, 1, 2, 3, 3, 3, 3, 1} and value sequence {2, 3, 4, 5, 6, 7, 8, 9, 10} would be reduced to <code>keys={1, 2, 3, 1}</code> , <code>values={9, 5, 30, 10}</code> .	<code><hpx/numeric.hpp></code>	
<code>hpx::transform_reduce</code>	Sums up a range of elements after applying a function. Also, accumulates the inner products of two input ranges.	<code><hpx/numeric.hpp></code>	transform_reduce ¹⁰⁹
<code>hpx::transform_inclusive_scan</code>	Does an inclusive parallel scan over a range of elements after applying a function.	<code><hpx/numeric.hpp></code>	transform_inclusive_scan ¹¹⁰
<code>hpx::parallel_exclusive_scan</code>	Does an exclusive parallel scan over a range of elements after applying a function.	<code><hpx/numeric.hpp></code>	transform_exclusive_scan ¹¹¹

¹⁰⁵ http://en.cppreference.com/w/cpp/algorithm/adjacent_difference

¹⁰⁶ http://en.cppreference.com/w/cpp/algorithm/exclusive_scan

¹⁰⁷ <http://en.cppreference.com/w/cpp/algorithm/reduce>

¹⁰⁸ http://en.cppreference.com/w/cpp/algorithm/inclusive_scan

¹⁰⁹ http://en.cppreference.com/w/cpp/algorithm/transform_reduce

¹¹⁰ http://en.cppreference.com/w/cpp/algorithm/transform_inclusive_scan

¹¹¹ http://en.cppreference.com/w/cpp/algorithm/transform_exclusive_scan

Table 2.19: Dynamic Memory Management (In Header:
<hpx/memory.hpp>)

Name	Description	In header	Algorithm page at cppreference.com
hpx::destroy	Destroys a range of objects.	<hpx/ memory. .hpp>	destroy ¹¹²
hpx::destroy_n	Destroys a range of objects.	<hpx/ memory. .hpp>	destroy_n ¹¹³
hpx::uninitialized_copy	Copies a range of objects to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_copy ¹¹⁴
hpx::uninitialized_copy_n	Copies a number of objects to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_copy_n ¹¹⁵
hpx::uninitialized_default_construct	Copies a range of objects to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_default_construct ¹¹⁶
hpx::uninitialized_default_construct_n	Copies a number of objects to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_default_construct_n ¹¹⁷
hpx::uninitialized_fill	Copies an object to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_fill ¹¹⁸
hpx::uninitialized_fill_n	Copies an object to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_fill_n ¹¹⁹
hpx::uninitialized_move	Moves a range of objects to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_move ¹²⁰
hpx::uninitialized_move_n	Moves a number of objects to an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_move_n ¹²¹
hpx::uninitialized_value_construct	Constructs objects in an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_value_construct ¹²²
hpx::uninitialized_value_construct_n	Constructs objects in an uninitialized area of memory.	<hpx/ memory. .hpp>	uninitialized_value_construct_n ¹²³

¹¹² <http://en.cppreference.com/w/cpp/memory/destroy>

¹¹³ http://en.cppreference.com/w/cpp/memory/destroy_n

¹¹⁴ http://en.cppreference.com/w/cpp/memory/uninitialized_copy

¹¹⁵ http://en.cppreference.com/w/cpp/memory/uninitialized_copy_n

¹¹⁶ http://en.cppreference.com/w/cpp/memory/uninitialized_default_construct

¹¹⁷ http://en.cppreference.com/w/cpp/memory/uninitialized_default_construct_n

¹¹⁸ http://en.cppreference.com/w/cpp/memory/uninitialized_fill

¹¹⁹ http://en.cppreference.com/w/cpp/memory/uninitialized_fill_n

¹²⁰ http://en.cppreference.com/w/cpp/memory/uninitialized_move

¹²¹ http://en.cppreference.com/w/cpp/memory/uninitialized_move_n

¹²² http://en.cppreference.com/w/cpp/memory/uninitialized_value_construct

¹²³ http://en.cppreference.com/w/cpp/memory/uninitialized_value_construct_n

Table 2.20: Index-based for-loops (In Header: *<hpx/algorithm.hpp>*)

Name	Description	In header
<code>hpx::experimental::for_loo</code>	Implements loop functionality over a range specified by integral or iterator bounds.	<code><hpx/algorithm.hpp></code>
<code>hpx::experimental::for_loo</code>	Implements loop functionality over a range specified by integral or iterator bounds.	<code><hpx/algorithm.hpp></code>
<code>hpx::experimental::for_loo</code>	Implements loop functionality over a range specified by integral or iterator bounds.	<code><hpx/algorithm.hpp></code>
<code>hpx::experimental::for_loo</code>	Implements loop functionality over a range specified by integral or iterator bounds.	<code><hpx/algorithm.hpp></code>

Executor parameters and executor parameter traits

HPX introduces the notion of execution parameters and execution parameter traits. At this point, the only parameter that can be customized is the size of the chunks of work executed on a single HPX thread (such as the number of loop iterations combined to run as a single task).

An executor parameter object is responsible for exposing the calculation of the size of the chunks scheduled. It abstracts the (potentially platform-specific) algorithms of determining those chunk sizes.

The way executor parameters are implemented is aligned with the way executors are implemented. All functionalities of concrete executor parameter types are exposed and accessible through a corresponding `hpx::parallel::executor_parameter_traits` type.

With `executor_parameter_traits`, clients access all types of executor parameters uniformly:

```
std::size_t chunk_size =
    executor_parameter_traits<my_parameter_t>::get_chunk_size(my_parameter,
        my_executor, [](){ return 0; }, num_tasks);
```

This call synchronously retrieves the size of a single chunk of loop iterations (or similar) to combine for execution on a single HPX thread if the overall number of tasks to schedule is given by `num_tasks`. The lambda function exposes a means of test-probing the execution of a single iteration for performance measurement purposes. The execution parameter type might dynamically determine the execution time of one or more tasks in order to calculate the chunk size; see `hpx::execution::auto_chunk_size` for an example of this executor parameter type.

Other functions in the interface exist to discover whether an executor parameter type should be invoked once (i.e., it returns a static chunk size; see `hpx::execution::static_chunk_size`) or whether it should be invoked for each scheduled chunk of work (i.e., it returns a variable chunk size; for an example, see `hpx::execution::guided_chunk_size`).

Although this interface appears to require executor parameter type authors to implement all different basic operations, none are required. In practice, all operations have sensible defaults. However, some executor parameter types will naturally specialize all operations for maximum efficiency.

HPX implements the following executor parameter types:

- `hpx::execution::auto_chunk_size`: Loop iterations are divided into pieces and then assigned to threads. The number of loop iterations combined is determined based on measurements of how long the execution of 1% of the overall number of iterations takes. This executor parameter type makes sure that as many loop iterations are combined as necessary to run for the amount of time specified.

- `hpx::execution::static_chunk_size`: Loop iterations are divided into pieces of a given size and then assigned to threads. If the size is not specified, the iterations are, if possible, evenly divided contiguously among the threads. This executor parameters type is equivalent to OpenMP's STATIC scheduling directive.
- `hpx::execution::dynamic_chunk_size`: Loop iterations are divided into pieces of a given size and then dynamically scheduled among the cores; when a core finishes one chunk, it is dynamically assigned another. If the size is not specified, the default chunk size is 1. This executor parameter type is equivalent to OpenMP's DYNAMIC scheduling directive.
- `hpx::execution::guided_chunk_size`: Iterations are dynamically assigned to cores in blocks as cores request them until no blocks remain to be assigned. This is similar to `dynamic_chunk_size` except that the block size decreases each time a number of loop iterations is given to a thread. The size of the initial block is proportional to `number_of_iterations / number_of_cores`. Subsequent blocks are proportional to `number_of_iterations_remaining / number_of_cores`. The optional chunk size parameter defines the minimum block size. The default minimal chunk size is 1. This executor parameter type is equivalent to OpenMP's GUIDED scheduling directive.

Using task blocks

The `define_task_block`, `run` and the `wait` functions implemented based on N4411¹²⁴ are based on the `task_block` concept that is a part of the common subset of the Microsoft Parallel Patterns Library (PPL)¹²⁵ and the Intel Threading Building Blocks (TBB)¹²⁶ libraries.

These implementations adopt a simpler syntax than exposed by those libraries—one that is influenced by language-based concepts, such as `spawn` and `sync` from Cilk++¹²⁷ and `async` and `finish` from X10¹²⁸. They improve on existing practice in the following ways:

- The exception handling model is simplified and more consistent with normal C++ exceptions.
- Most violations of strict fork-join parallelism can be enforced at compile time (with compiler assistance, in some cases).
- The syntax allows scheduling approaches other than child stealing.

Consider an example of a parallel traversal of a tree, where a user-provided function `compute` is applied to each node of the tree, returning the sum of the results:

```
template <typename Func>
int traverse(node& n, Func && compute)
{
    int left = 0, right = 0;
    define_task_block(
        [&] (task_block<>& tr) {
            if (n.left)
                tr.run([&] { left = traverse(*n.left, compute); });
            if (n.right)
                tr.run([&] { right = traverse(*n.right, compute); });
        });
    return compute(n) + left + right;
}
```

¹²⁴ <http://wg21.link/n4411>

¹²⁵ <https://msdn.microsoft.com/en-us/library/dd492418.aspx>

¹²⁶ <https://www.threadingbuildingblocks.org/>

¹²⁷ <https://software.intel.com/en-us/articles/intel-cilk-plus/>

¹²⁸ <https://x10-lang.org/>

The example above demonstrates the use of two of the functions, `hpx::parallel::define_task_block` and the `hpx::parallel::task_block::run` member function of a `hpx::parallel::task_block`.

The `task_block` function delineates a region in a program code potentially containing invocations of threads spawned by the `run` member function of the `task_block` class. The `run` function spawns an *HPX* thread, a unit of work that is allowed to execute in parallel with respect to the caller. Any parallel tasks spawned by `run` within the task block are joined back to a single thread of execution at the end of the `define_task_block`. `run` takes a user-provided function object `f` and starts it asynchronously—i.e., it may return before the execution of `f` completes. The *HPX* scheduler may choose to run `f` immediately or delay running `f` until compute resources become available.

A `task_block` can be constructed only by `define_task_block` because it has no public constructors. Thus, `run` can be invoked directly or indirectly only from a user-provided function passed to `define_task_block`:

```
void g();

void f(task_block<>& tr)
{
    tr.run(g);           // OK, invoked from within task_block in h
}

void h()
{
    define_task_block(f);
}

int main()
{
    task_block<> tr;    // Error: no public constructor
    tr.run(g);           // No way to call run outside of a define_task_block
    return 0;
}
```

Extensions for task blocks

Using execution policies with task blocks

HPX implements some extensions for `task_block` beyond the actual standards proposal N4411¹²⁹. The main addition is that a `task_block` can be invoked with an execution policy as its first argument, very similar to the parallel algorithms.

An execution policy is an object that expresses the requirements on the ordering of functions invoked as a consequence of the invocation of a task block. Enabling passing an execution policy to `define_task_block` gives the user control over the amount of parallelism employed by the created `task_block`. In the following example the use of an explicit `par` execution policy makes the user's intent explicit:

```
template <typename Func>
int traverse(node *n, Func&& compute)
{
    int left = 0, right = 0;

    define_task_block(
        execution::par,           // execution::parallel_policy
        [&](task_block<>& tb) {
            if (n->left)
```

(continues on next page)

¹²⁹ <http://wg21.link/n4411>

(continued from previous page)

```

        tb.run([&] { left = traverse(n->left, compute); });
    if (n->right)
        tb.run([&] { right = traverse(n->right, compute); });
    });

return compute(n) + left + right;
}

```

This also causes the `hpx::parallel::v2::task_block` object to be a template in our implementation. The template argument is the type of the execution policy used to create the task block. The template argument defaults to `hpx::execution::parallel_policy`.

HPX still supports calling `hpx::parallel::v2::define_task_block` without an explicit execution policy. In this case the task block will run using the `hpx::execution::parallel_policy`.

HPX also adds the ability to access the execution policy that was used to create a given `task_block`.

Using executors to run tasks

Often, users want to be able to not only define an execution policy to use by default for all spawned tasks inside the task block, but also to customize the execution context for one of the tasks executed by `task_block::run`. Adding an optionally passed executor instance to that function enables this use case:

```

template <typename Func>
int traverse(node *n, Func&& compute)
{
    int left = 0, right = 0;

    define_task_block(
        execution::par,                                // execution::parallel_policy
        [&] (auto& tb) {
            if (n->left)
            {
                // use explicitly specified executor to run this task
                tb.run(my_executor(), [&] { left = traverse(n->left, compute); });
            }
            if (n->right)
            {
                // use the executor associated with the par execution policy
                tb.run([&] { right = traverse(n->right, compute); });
            }
        });
    }

    return compute(n) + left + right;
}

```

HPX still supports calling `hpx::parallel::v2::task_block::run` without an explicit executor object. In this case the task will be run using the executor associated with the execution policy that was used to call `hpx::parallel::v2::define_task_block`.

2.3.9 Writing distributed HPX applications

This section focuses on the features of *HPX* needed to write distributed applications, namely the *Active Global Address Space (AGAS)*, remotely executable functions (i.e., *actions*), and distributed objects (i.e., *components*).

Global names

HPX implements an *Active Global Address Space (AGAS)* which exposes a single uniform address space spanning all localities an application runs on. AGAS is a fundamental component of the ParalleX execution model. Conceptually, there is no rigid demarcation of local or global memory in AGAS; all available memory is a part of the same address space. AGAS enables named objects to be moved (migrated) across localities without having to change the object's name; i.e., no references to migrated objects have to be ever updated. This feature has significance for dynamic load balancing and in applications where the workflow is highly dynamic, allowing work to be migrated from heavily loaded nodes to less loaded nodes. In addition, immutability of names ensures that AGAS does not have to keep extra indirections ("bread crumbs") when objects move, hence, minimizing complexity of code management for system developers as well as minimizing overheads in maintaining and managing aliases.

The AGAS implementation in *HPX* does not automatically expose every local address to the global address space. It is the responsibility of the programmer to explicitly define which of the objects have to be globally visible and which of the objects are purely local.

In *HPX* global addresses (global names) are represented using the `hpx::id_type` data type. This data type is conceptually very similar to `void*` pointers as it does not expose any type information of the object it is referring to.

The only predefined global addresses are assigned to all localities. The following *HPX* API functions allow one to retrieve the global addresses of localities:

- `hpx::find_here`: retrieves the global address of the *locality* this function is called on.
- `hpx::find_all_localities`: retrieves the global addresses of all localities available to this application (including the *locality* the function is being called on).
- `hpx::find_remote_localities`: retrieves the global addresses of all remote localities available to this application (not including the *locality* the function is being called on).
- `hpx::get_num_localities`: retrieves the number of localities available to this application.
- `hpx::find_locality`: retrieves the global address of any *locality* supporting the given component type.
- `hpx::get_colocation_id`: retrieves the global address of the *locality* currently hosting the object with the given global address.

Additionally, the global addresses of localities can be used to create new instances of components using the following *HPX* API function:

- `hpx::components::new_`: Creates a new instance of the given Component type on the specified *locality*.

Note: *HPX* does not expose any functionality to delete component instances. All global addresses (as represented using `hpx::id_type`) are automatically garbage collected. When the last (global) reference to a particular component instance goes out of scope, the corresponding component instance is automatically deleted.

Applying actions

Action type definition

Actions are special types used to describe possibly remote operations. For every global function and every member function which has to be invoked distantly, a special type must be defined. For any global function the special macro `HPX_PLAIN_ACTION` can be used to define the action type. Here is an example demonstrating this:

```
namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }
}

// This will define the action type 'some_global_action' which represents
// the function 'app::some_global_function'.
HPX_PLAIN_ACTION(app::some_global_function, some_global_action);
```

Important: The macro `HPX_PLAIN_ACTION` has to be placed in global namespace, even if the wrapped function is located in some other namespace. The newly defined action type is placed in the global namespace as well.

If the action type should be defined somewhere not in global namespace, the action type definition has to be split into two macro invocations (`HPX_DEFINE_PLAIN_ACTION` and `HPX_REGISTER_ACTION`) as shown in the next example:

```
namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }

    // On conforming compilers the following macro expands to:
    //
    //     typedef hpx::actions::make_action<
    //         decltype(&some_global_function), &some_global_function
    //     >::type some_global_action;
    //
    // This will define the action type 'some_global_action' which represents
    // the function 'some_global_function'.
    HPX_DEFINE_PLAIN_ACTION(some_global_function, some_global_action);
}

// The following macro expands to a series of definitions of global objects
// which are needed for proper serialization and initialization support
// enabling the remote invocation of the function ``some_global_function``.
HPX_REGISTER_ACTION(app::some_global_action, app_some_global_action);
```

The shown code defines an action type `some_global_action` inside the namespace `app`.

Important: If the action type definition is split between two macros as shown above, the name of the action type to create has to be the same for both macro invocations (here `some_global_action`).

Important: The second argument passed to `HPX_REGISTER_ACTION` (`app_some_global_action`) has to comprise a globally unique C++ identifier representing the action. This is used for serialization purposes.

For member functions of objects which have been registered with AGAS (e.g., ‘components’), a different registration macro `HPX_DEFINE_COMPONENT_ACTION` has to be utilized. Any component needs to be declared in a header file and have some special support macros defined in a source file. Here is an example demonstrating this. The first snippet has to go into the header file:

```
namespace app
{
    struct some_component
        : hpx::components::component_base<some_component>
    {
        int some_member_function(std::string s)
        {
            return boost::lexical_cast<int>(s);
        }

        // This will define the action type 'some_member_action' which
        // represents the member function 'some_member_function' of the
        // object type 'some_component'.
        HPX_DEFINE_COMPONENT_ACTION(some_component, some_member_function,
                                    some_member_action);
    };
}

// Note: The second argument to the macro below has to be systemwide-unique
//       C++ identifiers
HPX_REGISTER_ACTION_DECLARATION(app::some_component::some_member_action, some_
                                ↪component_some_action);
```

The next snippet belongs in a source file (e.g., the main application source file) in the simplest case:

```
typedef hpx::components::component<app::some_component> component_type;
typedef app::some_component some_component;

HPX_REGISTER_COMPONENT(component_type, some_component);

// The parameters for this macro have to be the same as used in the corresponding
// HPX_REGISTER_ACTION_DECLARATION() macro invocation above
typedef some_component::some_member_action some_component_some_action;
HPX_REGISTER_ACTION(some_component_some_action);
```

While these macro invocations are a bit more complex than those for simple global functions, they should still be manageable.

The most important macro invocation is the `HPX_DEFINE_COMPONENT_ACTION` in the header file as this defines the action type we need to invoke the member function. For a complete example of a simple component action see `component_in_executable.cpp`.

Action invocation

The process of invoking a global function (or a member function of an object) with the help of the associated action is called ‘applying the action’. Actions can have arguments, which will be supplied while the action is applied. At the minimum, one parameter is required to apply any action - the id of the *locality* the associated function should be invoked on (for global functions), or the id of the component instance (for member functions). Generally, *HPX* provides several ways to apply an action, all of which are described in the following sections.

Generally, *HPX* actions are very similar to ‘normal’ C++ functions except that actions can be invoked remotely. Fig. 2.8 below shows an overview of the main API exposed by *HPX*. This shows the function invocation syntax as defined by the C++ language (dark gray), the additional invocation syntax as provided through C++ Standard Library features (medium gray), and the extensions added by *HPX* (light gray) where:

- `f` function to invoke,
- `p...` (optional) arguments,
- `R`: return type of `f`,
- `action`: action type defined by, `HPX_DEFINE_PLAIN_ACTION` or `HPX_DEFINE_COMPONENT_ACTION` encapsulating `f`,
- `a`: an instance of the type `^action`,
- `id`: the global address the action is applied to.

<code>R f(p...)</code>	Synchronous Execution (returns <code>R</code>)	Asynchronous Execution (returns <code>future<R></code>)	Fire & Forget Execution (returns <code>void</code>)
Functions (direct invocation)	<code>f(p...)</code> <small>C++</small>	<code>async(f, p...)</code>	<code>apply(f, p...)</code>
Functions (lazy invocation)	<code>bind(f, p...)(...)</code>	<code>async(bind(f, p...), ...)</code> <small>C++ Standard Library</small>	<code>apply(bind(f, p...), ...)</code>
Actions (direct invocation)	<code>HPX_ACTION(f, action)</code> <code>a(id, p...)</code>	<code>HPX_ACTION(f, action)</code> <code>async(a, id, p...)</code>	<code>HPX_ACTION(f, action)</code> <code>apply(a, id, p...)</code>
Actions (lazy invocation)	<code>HPX_ACTION(f, action)</code> <code>bind(a, id, p...)</code> <code>(...)</code>	<code>HPX_ACTION(f, action)</code> <code>async(bind(a, id, p...), ...)</code>	<code>HPX_ACTION(f, action)</code> <code>apply(bind(a, id, p...), ...)</code> <small>HPX</small>

Fig. 2.8: Overview of the main API exposed by *HPX*.

This figure shows that *HPX* allows the user to apply actions with a syntax similar to the C++ standard. In fact, all action types have an overloaded function operator allowing to synchronously apply the action. Further, *HPX* implements `hpx::async` which semantically works similar to the way `std::async` works for plain C++ function.

Note: The similarity of applying an action to conventional function invocations extends even further. *HPX* implements `hpx::bind` and `hpx::function` two facilities which are semantically equivalent to the `std::bind` and `std::function` types as defined by the C++11 Standard. While `hpx::async` extends beyond the conventional semantics by supporting actions and conventional C++ functions, the *HPX* facilities `hpx::bind` and `hpx::function` extend beyond the conventional standard facilities too. The *HPX* facilities not only support conventional functions, but can be used for actions as well.

Additionally, *HPX* exposes `hpx::apply` and `hpx::async_continue` both of which refine and extend the standard C++ facilities.

The different ways to invoke a function in *HPX* will be explained in more detail in the following sections.

Applying an action asynchronously without any synchronization

This method ('fire and forget') will make sure the function associated with the action is scheduled to run on the target *locality*. Applying the action does not wait for the function to start running, instead it is a fully asynchronous operation. The following example shows how to apply the action as defined *in the previous section* on the local *locality* (the *locality* this code runs on):

```
some_global_action act;      // define an instance of some_global_action
hpx::apply(act, hpx::find_here(), 2.0);
```

(the function `hpx::find_here()` returns the id of the local *locality*, i.e. the *locality* this code executes on).

Any component member function can be invoked using the same syntactic construct. Given that `id` is the global address for a component instance created earlier, this invocation looks like:

```
some_component_action act;      // define an instance of some_component_action
hpx::apply(act, id, "42");
```

In this case any value returned from this action (e.g. in this case the integer 42 is ignored. Please look at *Action type definition* for the code defining the component action `some_component_action` used.

Applying an action asynchronously with synchronization

This method will make sure the action is scheduled to run on the target *locality*. Applying the action itself does not wait for the function to start running or to complete, instead this is a fully asynchronous operation similar to using `hpx::apply` as described above. The difference is that this method will return an instance of a `hpx::future<>` encapsulating the result of the (possibly remote) execution. The future can be used to synchronize with the asynchronous operation. The following example shows how to apply the action from above on the local *locality*:

```
some_global_action act;      // define an instance of some_global_action
hpx::future<void> f = hpx::async(act, hpx::find_here(), 2.0);
//
// ... other code can be executed here
//
f.get();      // this will possibly wait for the asynchronous operation to 'return'
```

(as before, the function `hpx::find_here()` returns the id of the local *locality* (the *locality* this code is executed on).

Note: The use of a `hpx::future<void>` allows the current thread to synchronize with any remote operation not returning any value.

Note: Any `std::future<>` returned from `std::async()` is required to block in its destructor if the value has not been set for this future yet. This is not true for `hpx::future<>` which will never block in its destructor, even if the value has not been returned to the future yet. We believe that consistency in the behavior of futures is more important than standards conformance in this case.

Any component member function can be invoked using the same syntactic construct. Given that `id` is the global address for a component instance created earlier, this invocation looks like:

```
some_component_action act;      // define an instance of some_component_action
hpx::future<int> f = hpx::async(act, id, "42");
//
// ... other code can be executed here
//
cout << f.get();    // this will possibly wait for the asynchronous operation to
// return' 42
```

Note: The invocation of `f.get()` will return the result immediately (without suspending the calling thread) if the result from the asynchronous operation has already been returned. Otherwise, the invocation of `f.get()` will suspend the execution of the calling thread until the asynchronous operation returns its result.

Applying an action synchronously

This method will schedule the function wrapped in the specified action on the target *locality*. While the invocation appears to be synchronous (as we will see), the calling thread will be suspended while waiting for the function to return. Invoking a plain action (e.g. a global function) synchronously is straightforward:

```
some_global_action act;      // define an instance of some_global_action
act(hpx::find_here(), 2.0);
```

While this call looks just like a normal synchronous function invocation, the function wrapped by the action will be scheduled to run on a new thread and the calling thread will be suspended. After the new thread has executed the wrapped global function, the waiting thread will resume and return from the synchronous call.

Equivalently, any action wrapping a component member function can be invoked synchronously as follows:

```
some_component_action act;      // define an instance of some_component_action
int result = act(id, "42");
```

The action invocation will either schedule a new thread locally to execute the wrapped member function (as before, `id` is the global address of the component instance the member function should be invoked on), or it will send a parcel to the remote *locality* of the component causing a new thread to be scheduled there. The calling thread will be suspended until the function returns its result. This result will be returned from the synchronous action invocation.

It is very important to understand that this ‘synchronous’ invocation syntax in fact conceals an asynchronous function call. This is beneficial as the calling thread is suspended while waiting for the outcome of a potentially remote operation. The *HPX* thread scheduler will schedule other work in the meantime, allowing the application to make further progress while the remote result is computed. This helps overlapping computation with communication and hiding communication latencies.

Note: The syntax of applying an action is always the same, regardless whether the target *locality* is remote to the invocation *locality* or not. This is a very important feature of *HPX* as it frees the user from the task of keeping track what actions have to be applied locally and which actions are remote. If the target for applying an action is local, a new thread is automatically created and scheduled. Once this thread is scheduled and run, it will execute the function encapsulated by that action. If the target is remote, *HPX* will send a parcel to the remote *locality* which encapsulates the action and its parameters. Once the parcel is received on the remote *locality* *HPX* will create and schedule a new thread there. Once this thread runs on the remote *locality*, it will execute the function encapsulated by the action.

Applying an action with a continuation but without any synchronization

This method is very similar to the method described in section [Applying an action asynchronously without any synchronization](#). The difference is that it allows the user to chain a sequence of asynchronous operations, while handing the (intermediate) results from one step to the next step in the chain. Where `hpx::apply` invokes a single function using ‘fire and forget’ semantics, `hpx::apply_continue` triggers a chain of functions without the need for the execution flow ‘to come back’ to the invocation site. Each of the asynchronous functions can be executed on a different *locality*.

Applying an action with a continuation and with synchronization

This method is very similar to the method described in section [Applying an action asynchronously with synchronization](#). In addition to what `hpx::async` can do, the functions `hpx::async_continue` takes an additional function argument. This function will be called as the continuation of the executed action. It is expected to perform additional operations and to make sure that a result is returned to the original invocation site. This method chains operations asynchronously by providing a continuation operation which is automatically executed once the first action has finished executing.

As an example we chain two actions, where the result of the first action is forwarded to the second action and the result of the second action is sent back to the original invocation site:

```
// first action
std::int32_t action1(std::int32_t i)
{
    return i+1;
}
HPX_PLAIN_ACTION(action1);      // defines action1_type

// second action
std::int32_t action2(std::int32_t i)
{
    return i*2;
}
HPX_PLAIN_ACTION(action2);      // defines action2_type

// this code invokes 'action1' above and passes along a continuation
// function which will forward the result returned from 'action1' to
// 'action2'.
action1_type act1;           // define an instance of 'action1_type'
action2_type act2;           // define an instance of 'action2_type'
hpx::future<int> f =
    hpx::async_continue(act1, hpx::make_continuation(act2),
        hpx::find_here(), 42);
hpx::cout << f.get() << "\n"; // will print: 86 ((42 + 1) * 2)
```

By default, the continuation is executed on the same *locality* as `hpx::async_continue` is invoked from. If you want to specify the *locality* where the continuation should be executed, the code above has to be written as:

```
// this code invokes 'action1' above and passes along a continuation
// function which will forward the result returned from 'action1' to
// 'action2'.
action1_type act1;           // define an instance of 'action1_type'
action2_type act2;           // define an instance of 'action2_type'
hpx::future<int> f =
    hpx::async_continue(act1, hpx::make_continuation(act2, hpx::find_here())),
```

(continues on next page)

(continued from previous page)

```
    hpx::find_here(), 42);
hpx::cout << f.get() << "\n"; // will print: 86 ((42 + 1) * 2)
```

Similarly, it is possible to chain more than 2 operations:

```
action1_type act1; // define an instance of 'action1_type'
action2_type act2; // define an instance of 'action2_type'
hpx::future<int> f =
    hpx::async_continue(act1,
        hpx::make_continuation(act2, hpx::make_continuation(act1)),
        hpx::find_here(), 42);
hpx::cout << f.get() << "\n"; // will print: 87 ((42 + 1) * 2 + 1)
```

The function `hpx::make_continuation` creates a special function object which exposes the following prototype:

```
struct continuation
{
    template <typename Result>
    void operator()(hpx::id_type id, Result&& result) const
    {
        ...
    }
};
```

where the parameters passed to the overloaded function `operator()` are:

- the `id` is the global id where the final result of the asynchronous chain of operations should be sent to (in most cases this is the id of the `hpx::future` returned from the initial call to `hpx::async_continue`. Any custom continuation function should make sure this `id` is forwarded to the last operation in the chain.
- the `result` is the result value of the current operation in the asynchronous execution chain. This value needs to be forwarded to the next operation.

Note: All of those operations are implemented by the predefined continuation function object which is returned from `hpx::make_continuation`. Any (custom) function object used as a continuation should conform to the same interface.

Action error handling

Like in any other asynchronous invocation scheme it is important to be able to handle error conditions occurring while the asynchronous (and possibly remote) operation is executed. In *HPX* all error handling is based on standard C++ exception handling. Any exception thrown during the execution of an asynchronous operation will be transferred back to the original invocation *locality*, where it is rethrown during synchronization with the calling thread.

Important: Exceptions thrown during asynchronous execution can be transferred back to the invoking thread only for the synchronous and the asynchronous case with synchronization. Like with any other unhandled exception, any exception thrown during the execution of an asynchronous action *without* synchronization will result in calling `hpx::terminate` causing the running application to exit immediately.

Note: Even if error handling internally relies on exceptions, most of the API functions exposed by *HPX* can be used

without throwing an exception. Please see [Working with exceptions](#) for more information.

As an example, we will assume that the following remote function will be executed:

```
namespace app
{
    void some_function_with_error(int arg)
    {
        if (arg < 0) {
            HPX_THROW_EXCEPTION(bad_parameter, "some_function_with_error",
                "some really bad error happened");
        }
        // do something else...
    }

// This will define the action type 'some_error_action' which represents
// the function 'app::some_function_with_error'.
HPX_PLAIN_ACTION(app::some_function_with_error, some_error_action);
```

The use of `HPX_THROW_EXCEPTION` to report the error encapsulates the creation of a `hpx::exception` which is initialized with the error code `hpx::bad_parameter`. Additionally it carries the passed strings, the information about the file name, line number, and call stack of the point the exception was thrown from.

We invoke this action using the synchronous syntax as described before:

```
// note: wrapped function will throw hpx::exception
some_error_action act;           // define an instance of some_error_action
try {
    act(hpx::find_here(), -3);   // exception will be rethrown from here
}
catch (hpx::exception const& e) {
    // prints: 'some really bad error happened: HPX(bad parameter)'
    cout << e.what();
}
```

If this action is invoked asynchronously with synchronization, the exception is propagated to the waiting thread as well and is re-thrown from the future's function `get()`:

```
// note: wrapped function will throw hpx::exception
some_error_action act;           // define an instance of some_error_action
hpx::future<void> f = hpx::async(act, hpx::find_here(), -3);
try {
    f.get();                  // exception will be rethrown from here
}
catch (hpx::exception const& e) {
    // prints: 'some really bad error happened: HPX(bad parameter)'
    cout << e.what();
}
```

For more information about error handling please refer to the section [Working with exceptions](#). There we also explain how to handle error conditions without having to rely on exception.

Writing components

A component in *HPX* is a C++ class which can be created remotely and for which its member functions can be invoked remotely as well. The following sections highlight how components can be defined, created, and used.

Defining components

In order for a C++ class type to be managed remotely in *HPX*, the type must be derived from the `hpx::components::component_base` template type. We call such C++ class types ‘components’.

Note that the component type itself is passed as a template argument to the base class:

```
// header file some_component.hpp

#include <hpx/include/components.hpp>

namespace app
{
    // Define a new component type 'some_component'
    struct some_component
        : hpx::components::component_base<some_component>
    {
        // This member function is has to be invoked remotely
        int some_member_function(std::string const& s)
        {
            return boost::lexical_cast<int>(s);
        }

        // This will define the action type 'some_member_action' which
        // represents the member function 'some_member_function' of the
        // object type 'some_component'.
        HPX_DEFINE_COMPONENT_ACTION(some_component, some_member_function, some_member_
        ↪action);
    };
}

// This will generate the necessary boiler-plate code for the action allowing
// it to be invoked remotely. This declaration macro has to be placed in the
// header file defining the component itself.
//
// Note: The second argument to the macro below has to be systemwide-unique
//       C++ identifiers
//
HPX_REGISTER_ACTION_DECLARATION(app::some_component::some_member_action, some_
    ↪component_some_action);
```

There is more boiler plate code which has to be placed into a source file in order for the component to be usable. Every component type is required to have macros placed into its source file, one for each component type and one macro for each of the actions defined by the component type.

For instance:

```
// source file some_component.cpp

#include "some_component.hpp"

// The following code generates all necessary boiler plate to enable the
```

(continues on next page)

(continued from previous page)

```
// remote creation of 'app::some_component' instances with 'hpx::new_<>()'
//
using some_component = app::some_component;
using some_component_type = hpx::components::component<some_component>;

// Please note that the second argument to this macro must be a
// (system-wide) unique C++-style identifier (without any namespaces)
//
HPX_REGISTER_COMPONENT(some_component_type, some_component);

// The parameters for this macro have to be the same as used in the corresponding
// HPX_REGISTER_ACTION_DECLARATION() macro invocation in the corresponding
// header file.
//
// Please note that the second argument to this macro must be a
// (system-wide) unique C++-style identifier (without any namespaces)
//
HPX_REGISTER_ACTION(app::some_component::some_member_action, some_component_some_
    ↪action);
```

Defining client side representation classes

Often it is very convenient to define a separate type for a component which can be used on the client side (from where the component is instantiated and used). This step might seem as unnecessary duplicating code, however it significantly increases the type safety of the code.

A possible implementation of such a client side representation for the component described in the previous section could look like:

```
#include <hpx/include/components.hpp>

namespace app
{
    // Define a client side representation type for the component type
    // 'some_component' defined in the previous section.
    //
    struct some_component_client
        : hpx::components::client_base<some_component_client, some_component>
    {
        using base_type = hpx::components::client_base<
            some_component_client, some_component>;
        some_component_client(hpx::future<hpx::id_type> && id)
            : base_type(std::move(id))
        {}

        hpx::future<int> some_member_function(std::string const& s)
        {
            some_component::some_member_action act;
            return hpx::async(act, get_id(), s);
        }
    };
}
```

A client side object stores the global id of the component instance it represents. This global id is accessible by calling the function `client_base<>::get_id()`. The special constructor which is provided in the example allows to

create this client side object directly using the API function `hpx::new_`.

Creating component instances

Instances of defined component types can be created in two different ways. If the component to create has a defined client side representation type, then this can be used, otherwise use the server type.

The following examples assume that `some_component_type` is the type of the server side implementation of the component to create. All additional arguments (see `,` `...` notation below) are passed through to the corresponding constructor calls of those objects:

```
// create one instance on the given locality
hpx::id_type here = hpx::find_here();
hpx::future<hpx::id_type> f =
    hpx::new_<some_component_type>(here, ...);

// create one instance using the given distribution
// policy (here: hpx::colocating_distribution_policy)
hpx::id_type here = hpx::find_here();
hpx::future<hpx::id_type> f =
    hpx::new_<some_component_type>(hpx::colocated(here), ...);

// create multiple instances on the given locality
hpx::id_type here = find_here();
hpx::future<std::vector<hpx::id_type>> f =
    hpx::new_<some_component_type[]>(here, num, ...);

// create multiple instances using the given distribution
// policy (here: hpx::binpacking_distribution_policy)
hpx::future<std::vector<hpx::id_type>> f = hpx::new_<some_component_type[]>(
    hpx::binpacking(hpx::find_all_localities()), num, ...);
```

The examples below demonstrate the use of the same API functions for creating client side representation objects (instead of just plain ids). These examples assume that `client_type` is the type of the client side representation of the component type to create. As above, all additional arguments (see `,` `...` notation below) are passed through to the corresponding constructor calls of the server side implementation objects corresponding to the `client_type`:

```
// create one instance on the given locality
hpx::id_type here = hpx::find_here();
client_type c = hpx::new_<client_type>(here, ...);

// create one instance using the given distribution
// policy (here: hpx::colocating_distribution_policy)
hpx::id_type here = hpx::find_here();
client_type c = hpx::new_<client_type>(hpx::colocated(here), ...);

// create multiple instances on the given locality
hpx::id_type here = hpx::find_here();
hpx::future<std::vector<client_type>> f =
    hpx::new_<client_type[]>(here, num, ...);

// create multiple instances using the given distribution
// policy (here: hpx::binpacking_distribution_policy)
hpx::future<std::vector<client_type>> f = hpx::new_<client_type[]>(
    hpx::binpacking(hpx::find_all_localities()), num, ...);
```

Using component instances

Segmented containers

In parallel programming, there is now a plethora of solutions aimed at implementing “partially contiguous” or segmented data structures, whether on shared memory systems or distributed memory systems. *HPX* implements such structures by drawing inspiration from Standard C++ containers.

Using segmented containers

A segmented container is a template class that is described in the namespace `hpx`. All segmented containers are very similar semantically to their sequential counterpart (defined in namespace `std` but with an additional template parameter named `DistPolicy`). The distribution policy is an optional parameter that is passed last to the segmented container constructor (after the container size when no default value is given, after the default value if not). The distribution policy describes the manner in which a container is segmented and the placement of each segment among the available runtime localities.

However, only a part of the `std` container member functions were reimplemented:

- (constructor), (destructor), operator=
- operator[]
- begin, cbegin, end, cend
- size

An example of how to use the `partitioned_vector` container would be:

```
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(double);

// By default, the number of segments is equal to the current number of
// localities
//
hpx::partitioned_vector<double> va(50);
hpx::partitioned_vector<double> vb(50, 0.0);
```

An example of how to use the `partitioned_vector` container with distribution policies would be:

```
#include <hpx/include/partitioned_vector.hpp>
#include <hpx/runtime_distributed/find_localities.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(double);

std::size_t num_segments = 10;
std::vector<hpx::id_type> locs = hpx::find_all_localities()

auto layout =
    hpx::container_layout( num_segments, locs );
```

(continues on next page)

(continued from previous page)

```
// The number of segments is 10 and those segments are spread across the
// localities collected in the variable locs in a Round-Robin manner
//
hpx::partitioned_vector<double> va(50, layout);
hpx::partitioned_vector<double> vb(50, 0.0, layout);
```

By definition, a segmented container must be accessible from any thread although its construction is synchronous only for the thread who has called its constructor. To overcome this problem, it is possible to assign a symbolic name to the segmented container:

```
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(double);

hpx::future<void> fserver = hpx::async(
    []() {
        hpx::partitioned_vector<double> v(50);

        // Register the 'partitioned_vector' with the name "some_name"
        //
        v.register_as("some_name");

        /* Do some code */
    });

hpx::future<void> fclient =
    hpx::async(
        []() {
            // Naked 'partitioned_vector'
            //
            hpx::partitioned_vector<double> v;

            // Now the variable v points to the same 'partitioned_vector' that has
            // been registered with the name "some_name"
            //
            v.connect_to("some_name");

            /* Do some code */
        });
});
```

Segmented containers

HPX provides the following segmented containers:

Table 2.21: Sequence containers

Name	Description	In header	Class page at cppreference.com
hpx::partitioned_	Dynamic segmented contiguous array.	<hpx/include/partitioned_vector.hpp>	vector ¹³⁰

Table 2.22: Unordered associative containers

Name	Description	In header	Class page at cppreference.com
hpx::unordered_map ¹³⁰	Segmented collection of key-value pairs, hashed by keys, keys are unique.	<hpx/include/unordered_map.hpp>	unordered_map ¹³¹

Segmented iterators and segmented iterator traits

The basic iterator used in the STL library is only suitable for one-dimensional structures. The iterators we use in HPX must adapt to the segmented format of our containers. Our iterators are then able to know when incrementing themselves if the next element of type T is in the same data segment or in another segment. In this second case, the iterator will automatically point to the beginning of the next segment.

Note: Note that the dereference operation operator * does not directly return a reference of type T& but an intermediate object wrapping this reference. When this object is used as an l-value, a remote write operation is performed; When this object is used as an r-value, implicit conversion to T type will take care of performing remote read operation.

It is sometimes useful not only to iterate element by element, but also segment by segment, or simply get a local iterator in order to avoid additional construction costs at each dereferencing operations. To mitigate this need, the hpx::traits::segmented_iterator_traits are used.

With segmented_iterator_traits users can uniformly get the iterators which specifically iterates over segments (by providing a segmented iterator as a parameter), or get the local begin/end iterators of the nearest local segment (by providing a per-segment iterator as a parameter):

```
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
// 
HPX_REGISTER_PARTITIONED_VECTOR(double);

using iterator = hpx::partitioned_vector<T>::iterator;
using traits = hpx::traits::segmented_iterator_traits<iterator>;

hpx::partitioned_vector<T> v;
std::size_t count = 0;

auto seg_begin = traits::segment(v.begin());
auto seg_end = traits::segment(v.end());

// Iterate over segments
for (auto seg_it = seg_begin; seg_it != seg_end; ++seg_it)
{
    auto loc_begin = traits::begin(seg_it);
    auto loc_end = traits::end(seg_it);

    // Iterate over elements inside segments
    for (auto lit = loc_begin; lit != loc_end; ++lit, ++count)
```

(continues on next page)

¹³⁰ <http://en.cppreference.com/w/cpp/container/vector>

¹³¹ http://en.cppreference.com/w/cpp/container/unordered_map

(continued from previous page)

```
{
    *lit = count;
}
}
```

Which is equivalent to:

```
hpx::partitioned_vector<T> v;
std::size_t count = 0;

auto begin = v.begin();
auto end = v.end();

for (auto it = begin; it != end; ++it, ++count)
{
    *it = count;
}
```

Using views

The use of multidimensional arrays is quite common in the numerical field whether to perform dense matrix operations or to process images. It exist many libraries which implement such object classes overloading their basic operators (e.g. ``+``, -, *, (), etc.). However, such operation becomes more delicate when the underlying data layout is segmented or when it is mandatory to use optimized linear algebra subroutines (i.e. BLAS subroutines).

Our solution is thus to relax the level of abstraction by allowing the user to work not directly on n-dimensionnal data, but on “n-dimensionnal collections of 1-D arrays”. The use of well-accepted techniques on contiguous data is thus preserved at the segment level, and the composability of the segments is made possible thanks to multidimensional array-inspired access mode.

Preface: Why SPMD?

Although *HPX* refutes by design this programming model, the *locality* plays a dominant role when it comes to implement vectorized code. To maximize local computations and avoid unneeded data transfers, a parallel section (or Single Programming Multiple Data section) is required. Because the use of global variables is prohibited, this parallel section is created via the RAII idiom.

To define a parallel section, simply write an action taking a `spmd_block` variable as a first parameter:

```
#include <hpx/collectives/spmd_block.hpp>

void bulk_function(hpx::lcos::spmd_block block /* , arg0, arg1, ... */)
{
    // Parallel section

    /* Do some code */
}
HPC_PLAIN_ACTION(bulk_function, bulk_action);
```

Note: In the following paragraphs, we will use the term “image” several times. An image is defined as a lightweight process whose entry point is a function provided by the user. It’s an “image of the function”.

The `spmd_block` class contains the following methods:

- [def Team information] `get_num_images`, `this_image`, `images_per_locality`
- [def Control statements] `sync_all`, `sync_images`

Here is a sample code summarizing the features offered by the `spmd_block` class:

```
#include <hpx/collectives/spmd_block.hpp>

void bulk_function(hpx::lcos::spmd_block block /* , arg0, arg1, ... */)
{
    std::size_t num_images = block.get_num_images();
    std::size_t this_image = block.this_image();
    std::size_t images_per_locality = block.images_per_locality();

    /* Do some code */

    // Synchronize all images in the team
    block.sync_all();

    /* Do some code */

    // Synchronize image 0 and image 1
    block.sync_images(0,1);

    /* Do some code */

    std::vector<std::size_t> vec_images = {2,3,4};

    // Synchronize images 2, 3 and 4
    block.sync_images(vec_images);

    // Alternative call to synchronize images 2, 3 and 4
    block.sync_images(vec_images.begin(), vec_images.end());

    /* Do some code */

    // Non-blocking version of sync_all()
    hpx::future<void> event =
        block.sync_all(hpx::launch::async);

    // Callback waiting for 'event' to be ready before being scheduled
    hpx::future<void> cb =
        event.then(
            [] (hpx::future<void>)
            {

                /* Do some code */

            });

    // Finally wait for the execution tree to be finished
    cb.get();
}

HPX_PLAIN_ACTION(bulk_test_function, bulk_test_action);
```

Then, in order to invoke the parallel section, call the function `define_spmd_block` specifying an arbitrary symbolic name and indicating the number of images per `locality` to create:

```

void bulk_function(hpx::lcos::spmd_block block, /* , arg0, arg1, ... */)
{
}

HPX_PLAIN_ACTION(bulk_test_function, bulk_test_action);

int main()
{
    /* std::size_t arg0, arg1, ...; */

    bulk_action act;
    std::size_t images_per_locality = 4;

    // Instantiate the parallel section
    hpx::lcos::define_spmd_block(
        "some_name", images_per_locality, std::move(act) /*, arg0, arg1, ... */);

    return 0;
}

```

Note: In principle, the user should never call the `spmd_block` constructor. The `define_spmd_block` function is responsible of instantiating `spmd_block` objects and broadcasting them to each created image.

SPMD multidimensional views

Some classes are defined as “container views” when the purpose is to observe and/or modify the values of a container using another perspective than the one that characterizes the container. For example, the values of an `std::vector` object can be accessed via the expression `[i]`. Container views can be used, for example, when it is desired for those values to be “viewed” as a 2D matrix that would have been flattened in a `std::vector`. The values would be possibly accessible via the expression `vv(i, j)` which would call internally the expression `v[k]`.

By default, the `partitioned_vector` class integrates 1-D views of its segments:

```

#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(double);

using iterator = hpx::partitioned_vector<double>::iterator;
using traits = hpx::traits::segmented_iterator_traits<iterator>;

hpx::partitioned_vector<double> v;

// Create a 1-D view of the vector of segments
auto vv = traits::segment(v.begin());

// Access segment i
std::vector<double> v = vv[i];

```

Our views are called “multidimensional” in the sense that they generalize to N dimensions the purpose of `segmented_iterator_traits::segment()` in the 1-D case. Note that in a parallel section, the 2-D expression `a(i, j) = b(i, j)` is quite confusing because without convention, each of the images invoked will race

to execute the statement. For this reason, our views are not only multidimensional but also “spmd-aware”.

Note: SPMD-awareness: The convention is simple. If an assignment statement contains a view subscript as an l-value, it is only and only the image holding the r-value who is evaluating the statement. (In MPI sense, it is called a Put operation).

Subscript-based operations

Here are some examples of using subscripts in the 2-D view case:

```
#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(double);

using Vec = hpx::partitioned_vector<double>;
using View_2D = hpx::partitioned_vector_view<double, 2>;

/* Do some code */

Vec v;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t height, width;

    // Instantiate the view
    View_2D vv(block, v.begin(), v.end(), {height, width});

    // The l-value is a view subscript, the image that owns vv(1,0)
    // evaluates the assignment.
    vv(0,1) = vv(1,0);

    // The l-value is a view subscript, the image that owns the r-value
    // (result of expression 'std::vector<double>(4,1.0)') evaluates the
    // assignment : oops! race between all participating images.
    vv(2,3) = std::vector<double>(4,1.0);
}
```

Iterator-based operations

Here are some examples of using iterators in the 3-D view case:

```
#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(int);
```

(continues on next page)

(continued from previous page)

```

using Vec = hpx::partitioned_vector<int>;
using View_3D = hpx::partitioned_vector_view<int, 3>;

/* Do some code */

Vec v1, v2;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t size_x, size_y, size_z;

    // Instantiate the views
    View_3D vv1(block, v1.begin(), v1.end(), {size_x, size_y, size_z});
    View_3D vv2(block, v2.begin(), v2.end(), {size_x, size_y, size_z});

    // Save previous segments covered by vv1 into segments covered by vv2
    auto vv2_it = vv2.begin();
    auto vv1_it = vv1.cbegin();

    for(; vv2_it != vv2.end(); vv2_it++, vv1_it++)
    {
        // It's a Put operation
        *vv2_it = *vv1_it;
    }

    // Ensure that all images have performed their Put operations
    block.sync_all();

    // Ensure that only one image is putting updated data into the different
    // segments covered by vv1
    if(block.this_image() == 0)
    {
        int idx = 0;

        // Update all the segments covered by vv1
        for(auto i = vv1.begin(); i != vv1.end(); i++)
        {
            // It's a Put operation
            *i = std::vector<float>(elt_size, idx++);
        }
    }
}

```

Here is an example that shows how to iterate only over segments owned by the current image:

```

#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/components/containers/partitioned_vector/partitioned_vector_local_view.
    ↵hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
// 
HPX_REGISTER_PARTITIONED_VECTOR(float);

using Vec = hpx::partitioned_vector<float>;

```

(continues on next page)

(continued from previous page)

```

using View_1D = hpx::partitioned_vector_view<float, 1>;
/* Do some code */

Vec v;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t num_segments;

    // Instantiate the view
    View_1D vv(block, v.begin(), v.end(), {num_segments});

    // Instantiate the local view from the view
    auto local_vv = hpx::local_view(vv);

    for (auto i = local_vv.begin(); i != local_vv.end(); i++)
    {
        std::vector<float> & segment = *i;

        /* Do some code */
    }
}
}

```

Instantiating sub-views

It is possible to construct views from other views: we call it sub-views. The constraint nevertheless for the subviews is to retain the dimension and the value type of the input view. Here is an example showing how to create a sub-view:

```

#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(float);

using Vec = hpx::partitioned_vector<float>;
using View_2D = hpx::partitioned_vector_view<float, 2>;

/* Do some code */

Vec v;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t N = 20;
    std::size_t tilesize = 5;

    // Instantiate the view
    View_2D vv(block, v.begin(), v.end(), {N, N});

    // Instantiate the subview
    View_2D svv(

```

(continues on next page)

(continued from previous page)

```

block, &vv(tilesize, 0), &vv(2*tilesize-1, tilesize-1), {tilesize, tilesize}, {N, N});

if(block.this_image() == 0)
{
    // Equivalent to 'vv(tilesize, 0) = 2.0f'
    svv(0, 0) = 2.0f;

    // Equivalent to 'vv(2*tilesize-1, tilesize-1) = 3.0f'
    svv(tilesize-1, tilesize-1) = 3.0f;
}

}

```

Note: The last parameter of the subview constructor is the size of the original view. If one would like to create a subview of the subview and so on, this parameter should stay unchanged. {N, N} for the above example).

C++ co-arrays

Fortran has extended its scalar element indexing approach to reference each segment of a distributed array. In this extension, a segment is attributed a ?co-index? and lives in a specific *locality*. A co-index provides the application with enough information to retrieve the corresponding data reference. In C++, containers present themselves as a ?smarter? alternative of Fortran arrays but there are still no corresponding standardized features similar to the Fortran co-indexing approach. We present here an implementation of such features in *HPX*.

Preface: co-array, a segmented container tied to a SPMD multidimensional views

As mentioned before, a co-array is a distributed array whose segments are accessible through an array-inspired access mode. We have previously seen that it is possible to reproduce such access mode using the concept of views. Nevertheless, the user must pre-create a segmented container to instantiate this view. We illustrate below how a single constructor call can perform those two operations:

```

#include <hpx/components/containers/coarray/coarray.hpp>
#include <hpx/collectives/spmd_block.hpp>

// The following code generates all necessary boiler plate to enable the
// co-creation of 'coarray'
//
HPX_REGISTER_COARRAY(double);

// Parallel section (suppose 'block' an spmd_block instance)
{
    using hpx::container::placeholders::_;

    std::size_t height=32, width=4, segment_size=10;

    hpx::coarray<double, 3> a(block, "a", {height, width, _}, segment_size);
    /* Do some code */
}

```

Unlike segmented containers, a co-array object can only be instantiated within a parallel section. Here is the description of the parameters to provide to the coarray constructor:

Table 2.23: Parameters of coarray constructor

Parameter	Description
block	Reference to a <code>spmd_block</code> object
"a"	Symbolic name of type <code>std::string</code>
{height, width, _}	Dimensions of the coarray object
segment_size	Size of a co-indexed element (i.e. size of the object referenced by the expression <code>a(i, j, k)</code>)

Note that the “last dimension size” cannot be set by the user. It only accepts the `constexpr` variable `hpx::container::placeholders::_`. This size, which is considered private, is equal to the number of current images (value returned by `block.get_num_images()`).

Note: An important constraint to remember about coarray objects is that all segments sharing the same “last dimension index” are located in the same image.

Using co-arrays

The member functions owned by the coarray objects are exactly the same as those of spmd multidimensional views. These are:

- * Subscript-based operations
- * Iterator-based operations

However, one additional functionality is provided. Knowing that the element `a(i, j, k)` is in the memory of the `k`th image, the use of local subscripts is possible.

Note: For spmd multidimensional views, subscripts are only global as it still involves potential remote data transfers.

Here is an example of using local subscripts:

```
#include <hpx/components/containers/coarray/coarray.hpp>
#include <hpx/collectives/spmd_block.hpp>

// The following code generates all necessary boiler plate to enable the
// co-creation of 'coarray'
//
HPX_REGISTER_COARRAY(double);

// Parallel section (suppose 'block' an spmd_block instance)
{
    using hpx::container::placeholders::_;

    std::size_t height=32, width=4, segment_size=10;

    hpx::coarray<double, 3> a(block, "a", {height, width, _}, segment_size);

    double idx = block.this_image()*height*width;

    for (std::size_t j = 0; j<width; j++)
        for (std::size_t i = 0; i<height; i++)

```

(continues on next page)

(continued from previous page)

```

{
    // Local write operation performed via the use of local subscript
    a(i,j,...) = std::vector<double>(elt_size, idx);
    idx++;
}

block.sync_all();
}

```

Note: When the “last dimension index” of a subscript is equal to `hpx::container::placeholders::_`, local subscript (and not global subscript) is used. It is equivalent to a global subscript used with a “last dimension index” equal to the value returned by `block.this_image()`.

2.3.10 Running on batch systems

This section walks you through launching *HPX* applications on various batch systems.

How to use *HPX* applications with PBS

Most *HPX* applications are executed on parallel computers. These platforms typically provide integrated job management services that facilitate the allocation of computing resources for each parallel program. *HPX* includes support for one of the most common job management systems, the Portable Batch System (PBS).

All PBS jobs require a script to specify the resource requirements and other parameters associated with a parallel job. The PBS script is basically a shell script with PBS directives placed within commented sections at the beginning of the file. The remaining (not commented-out) portions of the file executes just like any other regular shell script. While the description of all available PBS options is outside the scope of this tutorial (the interested reader may refer to in-depth documentation¹³² for more information), below is a minimal example to illustrate the approach. The following test application will use the multithreaded `hello_world_distributed` program, explained in the section *Remote execution with actions*.

```

#!/bin/bash
#
#PBS -l nodes=2:ppn=4

APP_PATH=~/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:nodes=`cat $PBS_NODEFILE`
```

Caution: If the first application specific argument (inside `$APP_OPTIONS`) is a non-option (i.e., does not start with a `-` or a `--`), then the argument has to be placed before the option `--hpx:nodes`, which, in this case, should be the last option on the command line.

Alternatively, use the option `--hpx:endnodes` to explicitly mark the end of the list of node names:

```
$ pbsdsh -u $APP_PATH --hpx:nodes`cat $PBS_NODEFILE` --hpx:endnodes $APP_OPTIONS
```

¹³² <http://www.clusterresources.com/torquedocs21/>

The `#PBS -l nodes=2:ppn=4` directive will cause two compute nodes to be allocated for the application, as specified in the option `nodes`. Each of the nodes will dedicate four cores to the program, as per the option `ppn`, short for “processors per node” (PBS does not distinguish between processors and cores). Note that requesting more cores per node than physically available is pointless and may prevent PBS from accepting the script.

On newer PBS versions the PBS command syntax might be different. For instance, the PBS script above would look like:

```
#!/bin/bash
#
#PBS -l select=2:ncpus=4

APP_PATH=~/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:nodes=`cat $PBS_NODEFILE`
```

`APP_PATH` and `APP_OPTIONS` are shell variables that respectively specify the correct path to the executable (`hello_world_distributed` in this case) and the command line options. Since the `hello_world_distributed` application doesn’t need any command line options, `APP_OPTIONS` has been left empty. Unlike in other execution environments, there is no need to use the `--hpx:threads` option to indicate the required number of OS threads per node; the *HPX* library will derive this parameter automatically from PBS.

Finally, `pbsdsh` is a PBS command that starts tasks to the resources allocated to the current job. It is recommended to leave this line as shown and modify only the PBS options and shell variables as needed for a specific application.

Important: A script invoked by `pbsdsh` starts in a very basic environment: the user’s `$HOME` directory is defined and is the current directory, the `LANG` variable is set to `C` and the `PATH` is set to the basic `/usr/local/bin:/usr/bin:/bin` as defined in a system-wide file `pbs_environment`. Nothing that would normally be set up by a system shell profile or user shell profile is defined, unlike the environment for the main job script.

Another choice is for the `pbsdsh` command in your main job script to invoke your program via a shell, like `sh` or `bash`, so that it gives an initialized environment for each instance. Users can create a small script `runme.sh`, which is used to invoke the program:

```
#!/bin/bash
# Small script which invokes the program based on what was passed on its
# command line.
#
# This script is executed by the bash shell which will initialize all
# environment variables as usual.
$@
```

Now, the script is invoked using the `pbsdsh` tool:

```
#!/bin/bash
#
#PBS -l nodes=2:ppn=4

APP_PATH=~/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u runme.sh $APP_PATH $APP_OPTIONS --hpx:nodes=`cat $PBS_NODEFILE`
```

All that remains now is submitting the job to the queuing system. Assuming that the contents of the PBS script were saved in the file `pbs_hello_world.sh` in the current directory, this is accomplished by typing:

```
$ qsub ./pbs_hello_world_pbs.sh
```

If the job is accepted, qsub will print out the assigned job ID, which may look like:

```
$ 42.supercomputer.some.university.edu
```

To check the status of your job, issue the following command:

```
$ qstat 42.supercomputer.some.university.edu
```

and look for a single-letter job status symbol. The common cases include:

- *Q* - signifies that the job is queued and awaiting its turn to be executed.
- *R* - indicates that the job is currently running.
- *C* - means that the job has completed.

The example qstat output below shows a job waiting for execution resources to become available:

Job id	Name	User	Time	Use	S	Queue
42.supercomputer	...ello_world.sh	joe_user		0	Q	batch

After the job completes, PBS will place two files, `pbs_hello_world.sh.o42` and `pbs_hello_world.sh.e42`, in the directory where the job was submitted. The first contains the standard output and the second contains the standard error from all the nodes on which the application executed. In our example, the error output file should be empty and the standard output file should contain something similar to:

```
hello world from OS-thread 3 on locality 0
hello world from OS-thread 2 on locality 0
hello world from OS-thread 1 on locality 1
hello world from OS-thread 0 on locality 0
hello world from OS-thread 3 on locality 1
hello world from OS-thread 2 on locality 1
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 1
```

Congratulations! You have just run your first distributed *HPX* application!

How to use *HPX* applications with SLURM

Just like PBS (described in section *How to use HPX applications with PBS*), SLURM is a job management system which is widely used on large supercomputing systems. Any *HPX* application can easily be run using SLURM. This section describes how this can be done.

The easiest way to run an *HPX* application using SLURM is to utilize the command line tool `srun`, which interacts with the SLURM batch scheduling system:

```
$ srun -p <partition> -N <number-of-nodes> hpx-application <application-arguments>
```

Here, `<partition>` is one of the node partitions existing on the target machine (consult the machine's documentation to get a list of existing partitions) and `<number-of-nodes>` is the number of compute nodes that should be used. By default, the *HPX* application is started with one *locality* per node and uses all available cores on a node. You can change the number of localities started per node (for example, to account for NUMA effects) by specifying the `-n` option of `srun`. The number of cores per *locality* can be set by `-c`. The `<application-arguments>` are any application specific arguments that need to be passed on to the application.

Note: There is no need to use any of the *HPX* command line options related to the number of localities, number of threads, or related to networking ports. All of this information is automatically extracted from the SLURM environment by the *HPX* startup code.

Important: The `srun` documentation explicitly states: “If `-c` is specified without `-n`, as many tasks will be allocated per node as possible while satisfying the `-c` restriction. For instance on a cluster with 8 CPUs per node, a job request for 4 nodes and 3 CPUs per task may be allocated 3 or 6 CPUs per node (1 or 2 tasks per node) depending upon resource consumption by other jobs.” For this reason, it’s recommended to always specify `-n <number-of-instances>`, even if `<number-of-instances>` is equal to one (1).

Interactive shells

To get an interactive development shell on one of the nodes, users can issue the following command:

```
$ srun -p <node-type> -N <number-of-nodes> --pty /bin/bash -l
```

After the shell has been opened, users can run their *HPX* application. By default, it uses all available cores. Note that if you requested one node, you don’t need to do `srun` again. However, if you requested more than one node, and want to run your distributed application, you can use `srun` again to start up the distributed *HPX* application. It will use the resources that have been requested for the interactive shell.

Scheduling batch jobs

The above mentioned method of running *HPX* applications is fine for development purposes. The disadvantage that comes with `srun` is that it only returns once the application is finished. This might not be appropriate for longer-running applications (for example, benchmarks or larger scale simulations). In order to cope with that limitation, users can use the `sbatch` command.

The `sbatch` command expects a script that it can run once the requested resources are available. In order to request resources, users need to add `#SBATCH` comments in their script or provide the necessary parameters to `sbatch` directly. The parameters are the same as with `run`. The commands you need to execute are the same you would need to start your application as if you were in an interactive shell.

2.3.11 Debugging *HPX* applications

Using a debugger with *HPX* applications

Using a debugger such as `gdb` with *HPX* applications is no problem. However, there are some things to keep in mind to make the experience somewhat more productive.

Call stacks in *HPX* can often be quite unwieldy as the library is heavily templated and the call stacks can be very deep. For this reason it is sometimes a good idea compile *HPX* in `RelWithDebInfo` mode, which applies some optimizations but keeps debugging symbols. This can often compress call stacks significantly. On the other hand, stepping through the code can also be more difficult because of statements being reordered and variables being optimized away. Also, note that because *HPX* implements user-space threads and context switching, call stacks may not always be complete in a debugger.

HPX launches not only worker threads but also a few helper threads. The first thread is the main thread, which typically does no work in an *HPX* application, except at startup and shutdown. If using the default settings, *HPX* will spawn six

additional threads (used for service thread pools). The first worker thread is usually the eighth thread, and most user codes will be run on these worker threads. The last thread is a helper thread used for *HPX* shutdown.

Finally, since *HPX* is a multi-threaded runtime, the following *gdb* options can be helpful:

```
set pagination off
set non-stop on
```

Non-stop mode allows users to have a single thread stop on a breakpoint without stopping all other threads as well.

Using sanitizers with *HPX* applications

Warning: Not all parts of *HPX* are sanitizer clean. This means that users may end up with false positives from *HPX* itself when using sanitizers for their applications.

To use sanitizers with *HPX*, turn on `HPX_WITH_SANITIZERS` and turn off `HPX_WITH_STACKOVERFLOW_DETECTION` during CMake configuration. It's recommended to also build Boost with the same sanitizers that will be used for *HPX*. The appropriate sanitizers can then be enabled using CMake by appending `-fsanitize=address -fno-omit-frame-pointer` to `CMAKE_CXX_FLAGS` and `-fsanitize=address` to `CMAKE_EXE_LINKER_FLAGS`. Replace `address` with the sanitizer that you want to use.

Debugging applications using core files

For *HPX* to generate useful core files, *HPX* has to be compiled without signal and exception handlers `HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL`. If this option is not specified, the signal handlers change the application state. For example, after a segmentation fault the stack trace will show the signal handler. Similarly, unhandled exceptions are also caught by these handlers and the stack trace will not point to the location where the unhandled exception was thrown.

In general, core files are a helpful tool to inspect the state of the application at the moment of the crash (post-mortem debugging), without the need of attaching a debugger beforehand. This approach to debugging is especially useful if the error cannot be reliably reproduced, as only a single crashed application run is required to gain potentially helpful information like a stacktrace.

To debug with core files, the operating system first has to be told to actually write them. On most Unix systems this can be done by calling:

```
$ ulimit -c unlimited
```

in the shell. Now the debugger can be started up with:

```
$ gdb <application> <core file name>
```

The debugger should now display the last state of the application. The default file name for core files is `core`.

2.3.12 Optimizing HPX applications

Performance counters

Performance counters in *HPX* are used to provide information as to how well the runtime system or an application is performing. The counter data can help determine system bottlenecks, and fine-tune system and application performance. The *HPX* runtime system, its networking, and other layers provide counter data that an application can consume to provide users with information about how well the application is performing.

Applications can also use counter data to determine how much system resources to consume. For example, an application that transfers data over the network could consume counter data from a network switch to determine how much data to transfer without competing for network bandwidth with other network traffic. The application could use the counter data to adjust its transfer rate as the bandwidth usage from other network traffic increases or decreases.

Performance counters are *HPX* parallel processes that expose a predefined interface. *HPX* exposes special API functions that allow one to create, manage, and read the counter data, and release instances of performance counters. Performance Counter instances are accessed by name, and these names have a predefined structure which is described in the section *Performance counter names*. The advantage of this is that any Performance Counter can be accessed remotely (from a different *locality*) or locally (from the same *locality*). Moreover, since all counters expose their data using the same API, any code consuming counter data can be utilized to access arbitrary system information with minimal effort.

Counter data may be accessed in real time. More information about how to consume counter data can be found in the section *Consuming performance counter data*.

All *HPX* applications provide command line options related to performance counters, such as the ability to list available counter types, or periodically query specific counters to be printed to the screen or save them in a file. For more information, please refer to the section *HPX Command Line Options*.

Performance counter names

All Performance Counter instances have a name uniquely identifying each instance. This name can be used to access the counter, retrieve all related meta data, and to query the counter data (as described in the section *Consuming performance counter data*). Counter names are strings with a predefined structure. The general form of a countername is:

```
/objectname{full_instancename}/countername@parameters
```

where `full_instancename` could be either another (full) counter name or a string formatted as:

```
parentinstancename#parentindex/instancename#instanceindex
```

Each separate part of a countername (e.g., `objectname`, `countername`, `parentinstancename`, `instancename`, and `parameters`) should start with a letter ('a'...'z', 'A'...'Z') or an underscore character ('_'), optionally followed by letters, digits ('0'...'9'), hyphen ('-'), or underscore characters. Whitespace is not allowed inside a counter name. The characters '/', '{', '}', '#', and '@' have a special meaning and are used to delimit the different parts of the counter name.

The parts `parentinstanceindex` and `instanceindex` are integers. If an index is not specified, *HPX* will assume a default of -1.

Two counter name examples

This section gives examples of both simple counter names and aggregate counter names. For more information on simple and aggregate counter names, please see [Performance counter instances](#).

An example of a well-formed (and meaningful) simple counter name would be:

```
/threads{locality#0/total}/count/cumulative
```

This counter returns the current cumulative number of executed (retired) HPX threads for the *locality* 0. The counter type of this counter is /threads/count/cumulative and the full instance name is locality#0/total. This counter type does not require an `instanceindex` or `parameters` to be specified.

In this case, the `parentindex` (the '0') designates the *locality* for which the counter instance is created. The counter will return the number of HPX threads retired on that particular *locality*.

Another example for a well formed (aggregate) counter name is:

```
/statistics{/threads{locality#0/total}/count/cumulative}/average@500
```

This counter takes the simple counter from the first example, samples its values every 500 milliseconds, and returns the average of the value samples whenever it is queried. The counter type of this counter is /statistics/average and the instance name is the full name of the counter for which the values have to be averaged. In this case, the `parameters` (the '500') specify the sampling interval for the averaging to take place (in milliseconds).

Performance counter types

Every performance counter belongs to a specific performance counter type which classifies the counters into groups of common semantics. The type of a counter is identified by the `objectname` and the `countername` parts of the name.

```
/objectname/countername
```

When an application starts HPX will register all available counter types on each of the localities. These counter types are held in a special performance counter registration database, which can be used to retrieve the meta data related to a counter type and to create counter instances based on a given counter instance name.

Performance counter instances

The `full_instancename` distinguishes different counter instances of the same counter type. The formatting of the `full_instancename` depends on the counter type. There are two types of counters: simple counters, which usually generate the counter values based on direct measurements, and aggregate counters, which take another counter and transform its values before generating their own counter values. An example for a simple counter is given [above](#): counting retired HPX threads. An aggregate counter is shown as an example [above](#) as well: calculating the average of the underlying counter values sampled at constant time intervals.

While simple counters use instance names formatted as `parentinstancename#parentindex/instancename#instanceindex`, most aggregate counters have the full counter name of the embedded counter as their instance name.

Not all simple counter types require specifying all four elements of a full counter instance name; some of the parts (`parentinstancename`, `parentindex`, `instancename`, and `instanceindex`) are optional for specific counters. Please refer to the documentation of a particular counter for more information about the formatting requirements for the name of this counter (see [Existing HPX performance counters](#)).

The parameters are used to pass additional information to a counter at creation time. They are optional, and they fully depend on the concrete counter. Even if a specific counter type allows additional parameters to be given, those usually are not required as sensible defaults will be chosen. Please refer to the documentation of a particular counter for more information about what parameters are supported, how to specify them, and what default values are assumed (see also [Existing HPX performance counters](#)).

Every *locality* of an application exposes its own set of performance counter types and performance counter instances. The set of exposed counters is determined dynamically at application start based on the execution environment of the application. For instance, this set is influenced by the current hardware environment for the *locality* (such as whether the *locality* has access to accelerators), and the software environment of the application (such as the number of OS threads used to execute *HPX* threads).

Using wildcards in performance counter names

It is possible to use wildcard characters when specifying performance counter names. Performance counter names can contain two types of wildcard characters:

- Wildcard characters in the performance counter type
- Wildcard characters in the performance counter instance name

A wildcard character has a meaning which is very close to usual file name wildcard matching rules implemented by common shells (like bash).

Table 2.24: Wildcard characters in the performance counter type

Wild-card	Description
*	This wildcard character matches any number (zero or more) of arbitrary characters.
?	This wildcard character matches any single arbitrary character.
[. . .]	This wildcard character matches any single character from the list of specified within the square brackets.

Table 2.25: Wildcard characters in the performance counter instance name

Wild-card	Description
*	This wildcard character matches any <i>locality</i> or any thread, depending on whether it is used for <code>locality#*</code> or <code>worker-thread#*</code> . No other wildcards are allowed in counter instance names.

Consuming performance counter data

You can consume performance data using either the command line interface, the *HPX* application or the *HPX* API. The command line interface is easier to use, but it is less flexible and does not allow one to adjust the behaviour of your application at runtime. The command line interface provides a convenience abstraction but simplified abstraction for querying and logging performance counter data for a set of performance counters.

Consuming performance counter data from the command line

HPX provides a set of predefined command line options for every application that uses `hpx::init` for its initialization. While there are many more command line options available (see [HPX Command Line Options](#)), the set of options related to performance counters allows one to list existing counters, and query existing counters once at application termination or repeatedly after a constant time interval.

The following table summarizes the available command line options:

Table 2.26: HPX Command Line Options Related to Performance Counters

Com-mand line option	Description
<code>--hpx:print</code>	Prints the specified performance counter either repeatedly and/or at the times specified by <code>--hpx:print-counter-at</code> (see also option <code>--hpx:print-counter-interval</code>).
<code>--hpx:print-counter-at</code>	Prints the specified performance counter either repeatedly and/or at the times specified by <code>--hpx:print-counter-at</code> . Reset the counter after the value is queried (see also option <code>--hpx:print-counter-interval</code>).
<code>--hpx:print-counter-interval</code>	Prints the performance counter(s) specified with <code>--hpx:print-counter</code> repeatedly after the time interval (specified in milliseconds) (default: 0 which means print once at shutdown).
<code>--hpx:print-counter-file</code>	Prints the performance counter(s) specified with <code>--hpx:print-counter</code> to the given file (default: console).
<code>--hpx:list</code>	Lists the names of all registered performance counters.
<code>--hpx:list-info</code>	Lists the descriptions of all registered performance counters.
<code>--hpx:print-format</code>	Prints the performance counter(s) specified with <code>--hpx:print-counter</code> . Possible formats in CSV format with header or without any header (see option <code>--hpx:no-csv-header</code>), possible values: csv (prints counter values in CSV format with full names as header) csv-short (prints counter values in CSV format with shortnames provided with <code>--hpx:print-counter</code> as <code>--hpx:print-counter shortname,full-countername</code>).
<code>--hpx:print-counter-format</code>	Prints the performance counter(s) specified with <code>--hpx:print-counter</code> and csv or csv-short format specified with <code>--hpx:print-counter-format</code> without header.
<code>--hpx:print-counter-reset</code>	Prints the performance counter(s) specified with <code>--hpx:print-counter</code> (or <code>--hpx:print-counter-reset</code>) at the given point in time. Possible argument values: startup, shutdown (default), nosutdown.
<code>--hpx:reset</code>	Resets all performance counter(s) specified with <code>--hpx:print-counter</code> after they have been evaluated.
<code>--hpx:print-type</code>	Appends counter type description to generated output.
<code>--hpx:print-locality</code>	Each locality prints only its own local counters.

While the options `--hpx:list-counters` and `--hpx:list-counter-infos` give a short list of all available counters, the full documentation for those can be found in the section [Existing HPX performance counters](#).

A simple example

All of the commandline options mentioned above can be tested using the `hello_world_distributed` example.

Listing all available counters `hello_world_distributed --hpx:list-counters` yields:

```
List of available counter instances (replace * below with the appropriate sequence number)
-----
/agas/count/allocate /agas/count/bind /agas/count/bind_gid
/agas/count/bind_name ... /threads{locality#*/allocator#*/}count/objects
/threads{locality#*/total}/count/stack-recycles
/threads{locality#*/total}/idle-rate
/threads{locality#*/worker-thread#*/}idle-rate
```

Providing more information about all available counters, `hello_world_distributed --hpx:list-counter-infos` yields:

```
Information about available counter instances (replace * below with the appropriate sequence number)
-----
fullname: /agas/count/allocate helptext: returns the number of invocations of the AGAS service 'allocate' type: counter_type::raw version: 1.0.0
-----

fullname: /agas/count/bind helptext: returns the number of invocations of the AGAS service 'bind' type: counter_type::raw version: 1.0.0
-----

fullname: /agas/count/bind_gid helptext: returns the number of invocations of the AGAS service 'bind_gid' type: counter_type::raw version: 1.0.0
-----

...

```

This command will not only list the counter names but also a short description of the data exposed by this counter.

Note: The list of available counters may differ depending on the concrete execution environment (hardware or software) of your application.

Requesting the counter data for one or more performance counters can be achieved by invoking `hello_world_distributed` with a list of counter names:

```
$ hello_world_distributed \
  --hpx:print-counter=/threads{locality#0/total}/count/cumulative \
  --hpx:print-counter=/agas{locality#0/total}/count/bind
```

which yields for instance:

```
hello world from OS-thread 0 on locality 0
/threads{locality#0/total}/count/cumulative,1,0.212527,[s],33
/agas{locality#0/total}/count/bind,1,0.212790,[s],11
```

The first line is the normal output generated by `hello_world_distributed` and has no relation to the counter data listed. The last two lines contain the counter data as gathered at application shutdown. These lines have six

fields, the counter name, the sequence number of the counter invocation, the time stamp at which this information has been sampled, the unit of measure for the time stamp, the actual counter value and an optional unit of measure for the counter value.

Note: The command line option `--hpx:print-counter-types` will append a seventh field to the generated output. This field will hold an abbreviated counter type.

The actual counter value can be represented by a single number (for counters returning singular values) or a list of numbers separated by '`:`' (for counters returning an array of values, like for instance a histogram).

Note: The name of the performance counter will be enclosed in double quotes '`"`' if it contains one or more commas '`,`'.

Requesting to query the counter data once after a constant time interval with this command line:

```
$ hello_world_distributed \
  --hpx:print-counter=/threads{locality#0/total}/count/cumulative \
  --hpx:print-counter=/agas{locality#0/total}/count/bind \
  --hpx:print-counter-interval=20
```

yields for instance (leaving off the actual console output of the `hello_world_distributed` example for brevity):

```
threads{locality#0/total}/count/cumulative,1,0.002409,[s],22
agas{locality#0/total}/count/bind,1,0.002542,[s],9
threads{locality#0/total}/count/cumulative,2,0.023002,[s],41
agas{locality#0/total}/count/bind,2,0.023557,[s],10
threads{locality#0/total}/count/cumulative,3,0.037514,[s],46
agas{locality#0/total}/count/bind,3,0.038679,[s],10
```

The command `--hpx:print-counter-destination=<file>` will redirect all counter data gathered to the specified file name, which avoids cluttering the console output of your application.

The command line option `--hpx:print-counter` supports using a limited set of wildcards for a (very limited) set of use cases. In particular, all occurrences of `#*` as in `locality#*` and in `worker-thread#*` will be automatically expanded to the proper set of performance counter names representing the actual environment for the executed program. For instance, if your program is utilizing four worker threads for the execution of *HPX* threads (see command line option `--hpx:threads`) the following command line

```
$ hello_world_distributed \
  --hpx:threads=4 \
  --hpx:print-counter=/threads{locality#0/worker-thread#*}/count/cumulative
```

will print the value of the performance counters monitoring each of the worker threads:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
hello world from OS-thread 3 on locality 0
hello world from OS-thread 2 on locality 0
/threads{locality#0/worker-thread#0}/count/cumulative,1,0.0025214,[s],27
/threads{locality#0/worker-thread#1}/count/cumulative,1,0.0025453,[s],33
/threads{locality#0/worker-thread#2}/count/cumulative,1,0.0025683,[s],29
/threads{locality#0/worker-thread#3}/count/cumulative,1,0.0025904,[s],33
```

The command `--hpx:print-counter-format` takes values `csv` and `csv-short` to generate CSV formatted counter values with a header.

With format as csv:

```
$ hello_world_distributed \
--hpx:threads=2 \
--hpx:print-counter-format csv \
--hpx:print-counter /threads{locality#/*/total}/count/cumulative \
--hpx:print-counter /threads{locality#/*/total}/count/cumulative-phases
```

will print the values of performance counters in CSV format with the full countername as a header:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
/threads{locality#/*/total}/count/cumulative,/threads{locality#/*/total}/count/
↪cumulative-phases
39,93
```

With format csv-short:

```
$ hello_world_distributed \
--hpx:threads 2 \
--hpx:print-counter-format csv-short \
--hpx:print-counter cumulative,/threads{locality#/*/total}/count/cumulative \
--hpx:print-counter phases,/threads{locality#/*/total}/count/cumulative-phases
```

will print the values of performance counters in CSV format with the short countername as a header:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
cumulative,phases
39,93
```

With format csv and csv-short when used with --hpx:print-counter-interval:

```
$ hello_world_distributed \
--hpx:threads 2 \
--hpx:print-counter-format csv-short \
--hpx:print-counter cumulative,/threads{locality#/*/total}/count/cumulative \
--hpx:print-counter phases,/threads{locality#/*/total}/count/cumulative-phases \
--hpx:print-counter-interval 5
```

will print the header only once repeating the performance counter value(s) repeatedly:

```
cum,phases
25,42
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
44,95
```

The command --hpx:no-csv-header can be used with --hpx:print-counter-format to print performance counter values in CSV format without any header:

```
$ hello_world_distributed \
--hpx:threads 2 \
--hpx:print-counter-format csv-short \
--hpx:print-counter cumulative,/threads{locality#/*/total}/count/cumulative \
--hpx:print-counter phases,/threads{locality#/*/total}/count/cumulative-phases \
--hpx:no-csv-header
```

will print:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
37,91
```

Consuming performance counter data using the *HPX API*

HPX provides an API that allows users to discover performance counters and to retrieve the current value of any existing performance counter from any application.

Discover existing performance counters

Retrieve the current value of any performance counter

Performance counters are specialized *HPX* components. In order to retrieve a counter value, the performance counter needs to be instantiated. *HPX* exposes a client component object for this purpose:

```
hpx::performance_counters::performance_counter counter(std::string const& name);
```

Instantiating an instance of this type will create the performance counter identified by the given name. Only the first invocation for any given counter name will create a new instance of that counter. All following invocations for a given counter name will reference the initially created instance. This ensures that at any point in time there is never more than one active instance of any of the existing performance counters.

In order to access the counter value (or to invoke any of the other functionality related to a performance counter, like start, stop or reset) member functions of the created client component instance should be called:

```
// print the current number of threads created on locality 0
hpx::performance_counters::performance_counter count(
    "/threads{locality#0/total}/count/cumulative");
hpx::cout << count.get_value<int>().get() << std::endl;
```

For more information about the client component type, see `hpx::performance_counters::performance_counter`

Note: In the above example `count.get_value()` returns a future. In order to print the result we must append `.get()` to retrieve the value. You could write the above example like this for more clarity:

```
// print the current number of threads created on locality 0
hpx::performance_counters::performance_counter count(
    "/threads{locality#0/total}/count/cumulative");
hpx::future<int> result = count.get_value<int>();
hpx::cout << result.get() << std::endl;
```

Providing performance counter data

HPX offers several ways by which you may provide your own data as a performance counter. This has the benefit of exposing additional, possibly application-specific information using the existing Performance Counter framework, unifying the process of gathering data about your application.

An application that wants to provide counter data can implement a performance counter to provide the data. When a consumer queries performance data, the HPX runtime system calls the provider to collect the data. The runtime system uses an internal registry to determine which provider to call.

Generally, there are two ways of exposing your own performance counter data: a simple, function-based way and a more complex, but more powerful way of implementing a full performance counter. Both alternatives are described in the following sections.

Exposing performance counter data using a simple function

The simplest way to expose arbitrary numeric data is to write a function which will then be called whenever a consumer queries this counter. Currently, this type of performance counter can only be used to expose integer values. The expected signature of this function is:

```
std::int64_t some_performance_data(bool reset);
```

The argument `bool reset` (which is supplied by the runtime system when the function is invoked) specifies whether the counter value should be reset after evaluating the current value (if applicable).

For instance, here is such a function returning how often it was invoked:

```
// The atomic variable 'counter' ensures the thread safety of the counter.
boost::atomic<std::int64_t> counter(0);

std::int64_t some_performance_data(bool reset)
{
    std::int64_t result = ++counter;
    if (reset)
        counter = 0;
    return result;
}
```

This example function exposes a linearly-increasing value as our performance data. The value is incremented on each invocation, i.e., each time a consumer requests the counter data of this performance counter.

The next step in exposing this counter to the runtime system is to register the function as a new raw counter type using the HPX API function `hpx::performance_counters::install_counter_type`. A counter type represents certain common characteristics of counters, like their counter type name and any associated description information. The following snippet shows an example of how to register the function `some_performance_data`, which is shown above, for a counter type named "/test/data". This registration has to be executed before any consumer instantiates, and queries an instance of this counter type:

```
#include <hpx/include/performance_counters.hpp>

void register_counter_type()
{
    // Call the HPX API function to register the counter type.
    hpx::performance_counters::install_counter_type(
        "/test/data",                                         // counter type name
        &some_performance_data,                            // function providing counter_
        &data);
```

(continues on next page)

(continued from previous page)

```

    "returns a linearly increasing counter value" // description text (optional)
    ""
);
}

```

Now it is possible to instantiate a new counter instance based on the naming scheme "/test{locality#*/total}/data" where * is a zero-based integer index identifying the *locality* for which the counter instance should be accessed. The function `hpx::performance_counters::install_counter_type` enables users to instantiate exactly one counter instance for each *locality*. Repeated requests to instantiate such a counter will return the same instance, i.e., the instance created for the first request.

If this counter needs to be accessed using the standard *HPX* command line options, the registration has to be performed during application startup, before `hpx_main` is executed. The best way to achieve this is to register an *HPX* startup function using the API function `hpx::register_startup_function` before calling `hpx::init` to initialize the runtime system:

```

int main(int argc, char* argv[])
{
    // By registering the counter type we make it available to any consumer
    // who creates and queries an instance of the type "/test/data".
    //
    // This registration should be performed during startup. The
    // function 'register_counter_type' should be executed as an HPX thread right
    // before hpx_main is executed.
    hpx::register_startup_function(&register_counter_type);

    // Initialize and run HPX.
    return hpx::init(argc, argv);
}

```

Please see the code in `simplest_performance_counter.cpp` for a full example demonstrating this functionality.

Implementing a full performance counter

Sometimes, the simple way of exposing a single value as a performance counter is not sufficient. For that reason, *HPX* provides a means of implementing full performance counters which support:

- Retrieving the descriptive information about the performance counter
- Retrieving the current counter value
- Resetting the performance counter (value)
- Starting the performance counter
- Stopping the performance counter
- Setting the (initial) value of the performance counter

Every full performance counter will implement a predefined interface:

```

// Copyright (c) 2007-2020 Hartmut Kaiser
//
// SPDX-License-Identifier: BSL-1.0
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

```

(continues on next page)

(continued from previous page)

```

#pragma once

#include <hpx/config.hpp>
#include <hpx/async_base/launch_policy.hpp>
#include <hpx/components/client_base.hpp>
#include <hpx/functional/bind_front.hpp>
#include <hpx/futures/future.hpp>
#include <hpx/modules/execution.hpp>

#include <hpx/performance_counters/counters_fwd.hpp>
#include <hpx/performance_counters/server/base_performance_counter.hpp>

#include <string>
#include <utility>
#include <vector>

///////////////////////////////
namespace hpx { namespace performance_counters {

/////////////////////////////
struct HPX_EXPORT performance_counter
    : components::client_base<performance_counter,
      server::base_performance_counter>
{
    using base_type = components::client_base<performance_counter,
      server::base_performance_counter>;

    performance_counter() = default;

    performance_counter(std::string const& name);

    performance_counter(
        std::string const& name, hpx::id_type const& locality);

    performance_counter(id_type const& id)
        : base_type(id)
    {
    }

    performance_counter(future<id_type>&& id)
        : base_type(HPX_MOVE(id))
    {
    }

    performance_counter(hpx::future<performance_counter>&& c)
        : base_type(HPX_MOVE(c))
    {
    }

    /////////////////////////
    future<counter_info> get_info() const;
    counter_info get_info(
        launch::sync_policy, error_code& ec = throws) const;

    future<counter_value> get_counter_value(bool reset = false);
    counter_value get_counter_value(

```

(continues on next page)

(continued from previous page)

```

    launch::sync_policy, bool reset = false, error_code& ec = throws);

future<counter_value> get_counter_value() const;
counter_value get_counter_value(
    launch::sync_policy, error_code& ec = throws) const;

future<counter_values_array> get_counter_values_array(
    bool reset = false);
counter_values_array get_counter_values_array(
    launch::sync_policy, bool reset = false, error_code& ec = throws);

future<counter_values_array> get_counter_values_array() const;
counter_values_array get_counter_values_array(
    launch::sync_policy, error_code& ec = throws) const;

///////////////////////////////
future<bool> start();
bool start(launch::sync_policy, error_code& ec = throws);

future<bool> stop();
bool stop(launch::sync_policy, error_code& ec = throws);

future<void> reset();
void reset(launch::sync_policy, error_code& ec = throws);

future<void> reinit(bool reset = true);
void reinit(
    launch::sync_policy, bool reset = true, error_code& ec = throws);

/////////////////////////////
future<std::string> get_name() const;
std::string get_name(
    launch::sync_policy, error_code& ec = throws) const;

private:
    template <typename T>
    static T extract_value(future<counter_value>&& value)
    {
        return value.get().get_value<T>();
    }

public:
    template <typename T>
    future<T> get_value(bool reset = false)
    {
        return get_counter_value(reset).then(hpx::launch::sync,
            hpx::bind_front(&performance_counter::extract_value<T>));
    }
    template <typename T>
    T get_value(
        launch::sync_policy, bool reset = false, error_code& ec = throws)
    {
        return get_counter_value(launch::sync, reset).get_value<T>(ec);
    }

    template <typename T>
    future<T> get_value() const

```

(continues on next page)

(continued from previous page)

```

    {
        return get_counter_value().then(hpx::launch::sync,
            hpx::bind_front(&performance_counter::extract_value<T>));
    }
template <typename T>
T get_value(launch::sync_policy, error_code& ec = throws) const
{
    return get_counter_value(launch::sync).get_value<T>(ec);
}
};

// Return all counters matching the given name (with optional wild cards).
HPX_EXPORT std::vector<performance_counter> discover_counters(
    std::string const& name, error_code& ec = throws);
} } // namespace hpx::performance_counters

```

In order to implement a full performance counter, you have to create an *HPX* component exposing this interface. To simplify this task, *HPX* provides a ready-made base class which handles all the boiler plate of creating a component for you. The remainder of this section will explain the process of creating a full performance counter based on the Sine example, which you can find in the directory examples/performance_counters/sine/.

The base class is defined in the header file `base_performance_counter.cpp` as:

```

// Copyright (c) 2007-2018 Hartmut Kaiser
//
// SPDX-License-Identifier: BSL-1.0
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

#pragma once

#include <hpx/config.hpp>
#include <hpx/actions_base/component_action.hpp>
#include <hpx/components_base/component_type.hpp>
#include <hpx/components_base/server/component_base.hpp>
#include <hpx/performance_counters/counters.hpp>
#include <hpx/performance_counters/server/base_performance_counter.hpp>

///////////////////////////////
//[performance_counter_base_class
namespace hpx { namespace performance_counters {
    template <typename Derived>
    class base_performance_counter;
}} // namespace hpx::performance_counters
//]

/////////////////////////////
namespace hpx { namespace performance_counters {
    template <typename Derived>
    class base_performance_counter
        : public hpx::performance_counters::server::base_performance_counter
        , public hpx::components::component_base<Derived>
    {
        private:
            typedef hpx::components::component_base<Derived> base_type;
    public:

```

(continues on next page)

(continued from previous page)

```

typedef Derived type_holder;
typedef hpx::performance_counters::server::base_performance_counter
    base_type_holder;

base_performance_counter() = default;

base_performance_counter(
    hpx::performance_counters::counter_info const& info)
: base_type_holder(info)
{
}

// Disambiguate finalize() which is implemented in both base classes
void finalize()
{
    base_type_holder::finalize();
    base_type::finalize();
}

hpx::naming::address get_current_address() const
{
    return hpx::naming::address(
        hpx::naming::get_gid_from_locality_id(hpx::get_locality_id()),
        hpx::components::get_component_type<Derived>(),
        const_cast<Derived*>(static_cast<Derived const*>(this)));
}
};

} } // namespace hpx::performance_counters

```

The single template parameter is expected to receive the type of the derived class implementing the performance counter. In the Sine example this looks like:

```

// Copyright (c) 2007-2012 Hartmut Kaiser
//
// SPDX-License-Identifier: BSL-1.0
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

#pragma once

#include <hpx/config.hpp>
#if !defined(HPX_COMPUTE_DEVICE_CODE)
#include <hpx/hpx.hpp>
#include <hpx/include/lcos_local.hpp>
#include <hpx/include/performance_counters.hpp>
#include <hpx/include/util.hpp>

#include <cstdint>

namespace performance_counters { namespace sine { namespace server {
    /////////////////////////////////
    //[sine_counter_definition
    class sine_counter
        : public hpx::performance_counters::base_performance_counter<sine_counter>
    []
{

```

(continues on next page)

(continued from previous page)

```

public:
    sine_counter()
        : current_value_(0)
        , evaluated_at_(0)
    {
    }
    explicit sine_counter(
        hpx::performance_counters::counter_info const& info);

    /// This function will be called in order to query the current value of
    /// this performance counter
    hpx::performance_counters::counter_value get_counter_value(bool reset);

    /// The functions below will be called to start and stop collecting
    /// counter values from this counter.
    bool start();
    bool stop();

    /// finalize() will be called just before the instance gets destructed
    void finalize();

protected:
    bool evaluate();

private:
    typedef hpx::spinlock mutex_type;

    mutable mutex_type mtx_;
    double current_value_;
    std::uint64_t evaluated_at_;

    hpx::util::interval_timer timer_;
};

} } // namespace performance_counters::sine::server
#endif

```

i.e., the type `sine_counter` is derived from the base class passing the type as a template argument (please see `simplest_performance_counter.cpp` for the full source code of the counter definition). For more information about this technique (called Curiously Recurring Template Pattern - CRTP), please see for instance the corresponding [Wikipedia article¹³³](#). This base class itself is derived from the `performance_counter` interface described above.

Additionally, a full performance counter implementation not only exposes the actual value but also provides information about:

- The point in time a particular value was retrieved.
- A (sequential) invocation count.
- The actual counter value.
- An optional scaling coefficient.
- Information about the counter status.

¹³³ http://en.wikipedia.org/wiki/Curiously_recurring_template_pattern

Existing HPX performance counters

The *HPX* runtime system exposes a wide variety of predefined performance counters. These counters expose critical information about different modules of the runtime system. They can help determine system bottlenecks and fine-tune system and application performance.

Table 2.27: AGAS performance counters

Counter type	Counter instance formatting	Description	Parameters
/agas/count/<agas_service> ?? where: <agas_service> is one of the following: <i>primary namespace services</i> : route, bind_gid, resolve_gid, unbind_gid, increment_credit, decrement_credit, allocate, begin_migration, end_migration <i>component namespace services</i> : bind_prefix, bind_name, resolve_id, unbind_name, iterate_types, get_component_typename, num_localities_type <i>locality namespace services</i> : free, localities, num_localities, num_threads, resolve_locality, resolved_localities <i>symbol namespace services</i> : bind, resolve, unbind, iterate_names, on_symbol_namespace_event	<agas_instance>/total where: <agas_instance> is the name of the AGAS service to query. Currently, this value will be locality#0 where 0 is the root <i>locality</i> (the id of the locality hosting the AGAS service). The value for * can be any <i>locality</i> id for the following <agas_service>: route, bind_gid, resolve_gid, unbind_gid, increment_credit, decrement_credit, bind, resolve, unbind, and iterate_names (only the primary and symbol AGAS service components live on all localities, whereas all other AGAS services are available on locality#0 only).	None	Returns the total number of invocations of the specified AGAS service since its creation.
/agas/<agas_service_category>/count ?? where: <agas_service_category> is one of the following: primary, locality, component or symbol	<agas_instance>/total where: <agas_instance> is the name of the AGAS service to query. Currently, this value will be locality#0 where 0 is the root <i>locality</i> (the id of the <i>locality</i> hosting the AGAS service). Except for <agas_service_category>, primary or symbol for which the value for * can be any <i>locality</i> id (only the primary and symbol AGAS service components live on all localities, whereas all other AGAS services are available on locality#0 only).	None	Returns the overall total number of invocations of all AGAS services provided by the given AGAS service category since its creation.
agas/time/<agas_service> ?? where: <agas_service> is one of the following: <i>primary namespace services</i> : route, bind_gid, resolve_gid, unbind_gid, increment_credit, decrement_credit, allocate, begin_migration, end_migration <i>component namespace services</i> : bind_prefix, bind_name, resolve_id, unbind_name, iterate_types, get_component_typename, num_localities_type <i>locality namespace services</i> : free, localities, num_localities, num_threads, resolve_locality, resolved_localities	<agas_instance>/total where: <agas_instance> is the name of the AGAS service to query. Currently, this value will be locality#0 where 0 is the root <i>locality</i> (the id of the <i>locality</i> hosting the AGAS service). The value for * can be any <i>locality</i> id for the following <agas_service>: route, bind_gid, resolve_gid, unbind_gid, increment_credit, decrement_credit, bind, resolve, unbind, and iterate_names (only the primary and symbol AGAS service components live on all localities, whereas all other AGAS services are available on locality#0 only).	None	Returns the overall execution time of the specified AGAS service since its creation (in nanoseconds).
164 locality namespace services: free, localities, num_localities, num_threads, resolve_locality, resolved_localities	Chapter 2: What's so special about HPX?		

Table 2.28: Parcel layer performance counters

Counter type	Counter instance for-formatting	Description	Parameters
/data/count/ <connection_type> <operation> ?? where: <operation> is one of the following: sent, received <connection_type> is one of the fol- lowing: tcp, mpi	locality# where: * is the <i>lo- cality</i> id of the <i>locality</i> the overall number of transmit- ted bytes should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	* Returns the overall number of raw (uncompressed) bytes sent or received (see <operation>, e.g. sent or received) for the specified <connection_type>. The performance counters are available only if the compile time constant HPX_HAVE_PARCELPORT_COUNTERS was defined while compiling the HPX core library (which is not defined by default). The corresponding cmake configuration constant is HPX_WITH_PARCELPORT_COUNTERS. The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default). The corresponding cmake configuration constant is HPX_WITH_PARCELPORT_MPI. Please see CMake variables used to configure HPX for more details.	None
/data/time/ <connection_type> <operation> ?? where: <operation> is one of the following: sent, received <connection_type> is one of the fol- lowing: tcp, mpi	locality# where: * is the <i>lo- cality</i> id of the <i>locality</i> the total trans- mission time should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	* Returns the total time (in nanoseconds) between the start of each asynchronous transmission operation and the end of the corresponding operation for the specified <connection_type> the given <i>locality</i> (see <operation>, e.g. sent or received). The performance counters are available only if the compile time constant HPX_HAVE_PARCELPORT_COUNTERS was defined while compiling the HPX core library (which is not defined by default). The corresponding cmake configuration constant is HPX_WITH_PARCELPORT_COUNTERS. The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default). The corresponding cmake configuration constant is HPX_WITH_PARCELPORT_MPI. Please see CMake variables used to configure HPX for more details.	None
/serialize/ count/ <connection_type> <operation> ?? where: <operation> is one of the following: sent, received <connection_type> is one of the fol- lowing: tcp, mpi	locality# total where: * is the <i>lo- cality</i> id of the <i>locality</i> the overall number of transmit- ted bytes should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	* Returns the overall number of bytes transferred (see <operation>, e.g. sent or received possibly compressed) for the specified <connection_type> by the given <i>locality</i> . The performance counters are available only if the compile time constant HPX_HAVE_PARCELPORT_COUNTERS was defined while compiling the HPX core library (which is not defined by default). The corresponding cmake configuration constant is HPX_WITH_PARCELPORT_COUNTERS. The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default). The corresponding cmake configuration constant is HPX_WITH_PARCELPORT_MPI. Please see CMake variables used to configure HPX for more details.	If the configura- time option -DHPX_WITH_PARCELPORT_ACT was specified, this counter allows one to specify an op- tional action name as its pa- rameter. In this case the counter will report the number of bytes transmitted for the given action only.
2.3. Manual			165

Table 2.29: Thread manager performance counters

Counter type	Counter instance formatting	Description	Parameters
/threads/count/ cumulative ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where: locality#* is defining the <i>locality</i> for which the overall number of retired HPX-threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the overall number of retired HPX-threads should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the 'default' pool.</p>	<p>Returns the overall number of executed (retired) HPX-threads on the given <i>locality</i> since application start. If the instance name is <i>total</i> the counter returns the accumulated number of retired HPX-threads for all worker threads (cores) on that <i>locality</i>. If the instance name is <i>worker-thread#*</i> the counter will return the overall number of retired HPX-threads for all worker threads separately which the current value of the available only if the configuration time constant <code>HPX_WITH_THREAD_CUMULATIVE_COUNTS</code></p>	None

continues on next page

¹³⁴ A message can potentially consist of more than one *parcel*.

Table 2.29 – continued from previous page

/threads/time/ average ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the average time spent executing one HPX-thread should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the average time spent executing one HPX-thread should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	<p>Returns the average time spent executing one HPX-thread on the given <i>locality</i> since application start. If the instance name is total the counter returns the average time spent executing one HPX-thread for all worker threads (cores) on that <i>locality</i>. If the instance name is worker-thread#* the counter will return the average time spent executing one HPX-thread for all worker threads separately. This counter is available only if the configuration time constants <code>HPX_WITH_THREAD_CUMULATIVE_COUNTS</code> (default: ON) and <code>HPX_WITH_THREAD_IDLE_RATES</code> are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].</p>	None
---------------------------------	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ average-overhead ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the average overhead spent executing one <i>HPX</i>-thread should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the average overhead spent executing one <i>HPX</i>-thread should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	<p>Returns the average time spent on overhead while executing one <i>HPX</i>-thread on the given <i>locality</i> since application start. If the instance name is total the counter returns the average time spent on overhead while executing one <i>HPX</i>-thread for all worker threads (cores) on that <i>locality</i>. If the instance name is worker-thread#* the counter will return the average time spent on overhead executing one <i>HPX</i>-thread for all worker threads separately. This counter is available only if the configuration time constants <code>HPX_WITH_THREAD_CUMULATIVE_COUNTS</code> (default: ON) and <code>HPX_WITH_THREAD_IDLE_RATES</code> are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].</p>	None
--	---	---	------

continues on next page

Table 2.29 – continued from previous page

/threads/count/ cumulative-phases ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the overall number of executed HPX-thread phases (invocations) should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the overall number of executed HPX-thread phases (invocations) should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	<p>Returns the overall number of executed HPX-thread phases (invocations) on the given <i>locality</i> since application start. If the instance name is <i>total</i> the counter returns the accumulated number of executed HPX-thread phases (invocations) for all worker threads (cores) on that <i>locality</i>. If the instance name is <i>worker-thread#*</i> the counter will return the overall number of executed HPX-thread phases for all worker threads separately. This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_CUMULATIVE_COUNTS</code> is set to ON (default: ON). The unit of measure for this counter is nanosecond [ns].</p>	None
--	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ average-phase ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the average time spent executing one HPX-thread phase (invocation) should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the average time executing one HPX-thread phase (invocation) should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	<p>Returns the average time spent executing one HPX-thread phase (invocation) on the given <i>locality</i> since application start. If the instance name is total the counter returns the average time spent executing one HPX-thread phase (invocation) for all worker threads (cores) on that <i>locality</i>. If the instance name is worker-thread#* the counter will return the average time spent executing one HPX-thread phase for all worker threads separately. This counter is available only if the configuration time constants <code>HPX_WITH_THREAD_CUMULATIVE_COUNTS</code> (default: ON) and <code>HPX_WITH_THREAD_IDLE_RATES</code> are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].</p>	None
---------------------------------------	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ average-phase-overhead ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the average time overhead executing one HPX-thread phase (invocation) should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the average overhead executing one HPX-thread phase (invocation) should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the average time spent on overhead executing one HPX-thread phase (invocation) on the given <i>locality</i> since application start. If the instance name is total the counter returns the average time spent on overhead while executing one HPX-thread phase (invocation) for all worker threads (cores) on that <i>locality</i> . If the instance name is worker-thread#* the counter will return the average time spent on overhead executing one HPX-thread phase for all worker threads separately. This counter is available only if the configuration time constants <code>HPX_WITH_THREAD_CUMULATIVE_COUNTS</code> (default: ON) and <code>HPX_WITH_THREAD_IDLE_RATES</code> are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].	None
--	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ overall ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the overall time spent running the scheduler should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the overall time spent running the sched- uler should be queried for. The worker thread number (given by the * is a (zero based) num- ber identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘de- fault’ pool.	Returns the overall time spent running the sched- uler on the given <i>locality</i> since application start. If the instance name is <i>total</i> the counter returns the overall time spent running the sched- uler for all worker threads (cores) on that <i>locality</i> . If the instance name is <i>worker-thread#*</i> the counter will return the overall time spent running the scheduler for all worker threads separately. This counter is available only if the config- uration time constant <code>HPX_WITH_THREAD_IDLE_RATES</code> is set to <code>ON</code> (default: <code>OFF</code>). The unit of measure for this counter is nanosecond [ns].	None
---------------------------------	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ cumulative ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the overall time spent executing all HPX-threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the overall time spent executing all HPX-threads should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	<p>Returns the overall time spent executing all HPX-threads on the given <i>locality</i> since application start. If the instance name is total the counter returns the overall time spent executing all HPX-threads for all worker threads (cores) on that <i>locality</i>. If the instance name is worker-thread#* the counter will return the overall time spent executing all HPX-threads for all worker threads separately. This counter is available only if the configuration time constants <code>HPX_THREAD_MAINTAIN_CUMULATIVE_COUNTS</code> (default: ON) and <code>HPX_THREAD_MAINTAIN_IDLE_RATES</code> are set to ON (default: OFF).</p>	None
------------------------------------	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ cumulative-overhead ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the overall overhead time in- curred by executing all <i>HPX</i> -threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>). pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the the over- all overhead time incurred by executing all <i>HPX</i> - threads should be queried for. The worker thread number (given by the * is a (zero based) num- ber identifying the worker thread. The number of available worker threads is usually specified on the command line for the ap- plication using the option -- <i>hpx:threads</i> . If no pool-name is specified the counter refers to the ‘de- fault’ pool.	Returns the overall overhead time incurred executing all <i>HPX</i> -threads on the given <i>locality</i> since application start. If the instance name is <i>total</i> the counter returns the overall overhead time incurred executing all <i>HPX</i> -threads for all worker threads (cores) on that <i>locality</i> . If the instance name is <i>worker-thread#*</i> the counter will return the overall overhead time incurred executing all <i>HPX</i> -threads for all worker threads sepa- rately. This counter is available only if the con- figuration time constants HPX_THREAD_MAINTAIN_CUMULATIVE_COUNTS (default: ON) and HPX_THREAD_MAINTAIN_IDLE_RATES are set to ON (default: OFF). The unit of mea- sure for this counter is nanosecond [ns].	None
---	--	---	------

continues on next page

Table 2.29 – continued from previous page

threads/count/ instantaneous/ <thread-state> ?? where: <thread-state> is one of the following: all, active, pending, suspended, terminated, staged	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the current number of threads with the given state should be queried for. The <i>locality</i> id (given by *) is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the current number of threads with the given state should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool. The staged thread state refers to registered tasks before they are converted to thread objects.	Returns the current number of HPX-threads having the given thread state on the given <i>locality</i> . If the instance name is <i>total</i> the counter returns the current number of HPX-threads of the given state for all worker threads (cores) on that <i>locality</i> . If the instance name is <i>worker-thread#*</i> the counter will return the current number of HPX-threads in the given state for all worker threads separately.	None
--	--	--	------

continues on next page

Table 2.29 – continued from previous page

<pre>threads/ wait-time/ <thread-state> ?? where: <thread-state> is one of the following: pending staged</pre>	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the average wait time of <i>HPX</i>-threads (pending) or thread descriptions (staged) with the given state should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the average wait time for the given state should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p> <p>The staged thread state refers to the wait time of registered tasks before they are converted into thread objects, while the pending thread state refers to the wait time of threads in any of the scheduling queues.</p>	<p>Returns the average wait time of <i>HPX</i>-threads (if the thread state is pending or of task descriptions (if the thread state is staged on the given <i>locality</i> since application start. If the instance name is <i>total</i> the counter returns the wait time of <i>HPX</i>-threads of the given state for all worker threads (cores) on that <i>locality</i>. If the instance name is <i>worker-thread#*</i> the counter will return the wait time of <i>HPX</i>-threads in the given state for all worker threads separately. These counters are available only if the compile time constant <code>HPX_WITH_THREAD_QUEUE_WAITTIME</code> was defined while compiling the <i>HPX</i> core library (default: OFF). The unit of measure for this counter is nanosecond [ns].</p>	None
--	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/ idle-rate ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the average idle rate of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i></p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the averaged idle rate should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	Returns the average idle rate for the given worker thread(s) on the given <i>locality</i> . The idle rate is defined as the ratio of the time spent on scheduling and management tasks and the overall time spent executing work since the application started. This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_IDLE_RATES</code> is set to ON (default: OFF).	None
------------------------------	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/ creation-idle-rate ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the average creation idle rate of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the averaged idle rate should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the average idle rate for the given worker thread(s) on the given <i>locality</i> which is caused by creating new threads. The creation idle rate is defined as the ratio of the time spent on creating new threads and the overall time spent executing work since the application started. This counter is available only if the configuration time constants HPX_WITH_THREAD_IDLE_RATES (default: OFF) and HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES are set to ON.	None
---------------------------------------	--	---	------

continues on next page

Table 2.29 – continued from previous page

/threads/ cleanup-idle-rate ??	<p>locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the average cleanup idle rate of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i>.</p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the averaged cleanup idle rate should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	<p>Returns the average idle rate for the given worker thread(s) on the given <i>locality</i> which is caused by cleaning up terminated threads. The cleanup idle rate is defined as the ratio of the time spent on cleaning up terminated thread objects and the overall time spent executing work since the application started. This counter is available only if the configuration time constants <code>HPX_WITH_THREAD_IDLE_RATES</code> (default: OFF) and <code>HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES</code> are set to ON.</p>	None
--------------------------------------	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threadqueue/ length ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the current length of all thread queues in the scheduler for all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the current length of all thread queues in the scheduler should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the overall length of all queues for the given worker thread(s) on the given <i>locality</i> .	None
/threads/count/ stack-unbinds ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the unbind (madvise) operations should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the total number of HPX-thread unbind (madvise) operations performed for the referenced <i>locality</i> . Note that this counter is not available on Windows based platforms.	None

continues on next page

Table 2.29 – continued from previous page

/threads/count/ stack-recycles ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the recycling operations should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the total number of HPX-thread recycling operations performed.	None
/threads/count/ stolen-from-pending ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the number of ‘stole’ threads should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the total number of HPX-threads ‘stolen’ from the pending thread queue by a neighboring thread worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).	None

continues on next page

Table 2.29 – continued from previous page

/threads/count/pending-misses??	<p>locality#*/total or locality#*/worker-thread#* or locality#*/pool#*/worker-thread#*</p> <p>where:</p> <p>locality#* is defining the <i>locality</i> for which the number of pending queue misses of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i></p> <p>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</p> <p>worker-thread#* is defining the worker thread for which the number of pending queue misses should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</p>	Returns the total number of times that the referenced worker-thread on the referenced <i>locality</i> failed to find pending HPX-threads in its associated queue. This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).	None
---------------------------------	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/count/ pending-accesses ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the number of pending queue accesses of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the number of pending queue accesses should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the total number of times that the referenced worker-thread on the referenced <i>locality</i> looked for pending HPX-threads in its associated queue. This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).	None
---	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/count/ stolen-from-staged ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the number of <i>HPX</i> -threads stolen from the staged queue of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the number of <i>HPX</i> -threads stolen from the staged queue should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the total number of <i>HPX</i> -threads ‘stolen’ from the staged thread queue by a neighboring worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).	None
---	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/count/ stolen-to-pending ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the number of <i>HPX</i> -threads stolen to the pending queue of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the number of <i>HPX</i> -threads stolen to the pending queue should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the total number of <i>HPX</i> -threads ‘stolen’ to the pending thread queue of the worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).	None
--	--	---	------

continues on next page

Table 2.29 – continued from previous page

/threads/count/ stolen-to-staged ??	locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the number of HPX-threads stolen to the staged queue of all (or one) worker threads should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the number of HPX-threads stolen to the staged queue should be queried for. The worker thread number (given by the * is a (zero based) worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the total number of HPX-threads ‘stolen’ to the staged thread queue of a neighboring worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).	None
---	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/count/ objects ??	locality#*/total or locality#*/ allocator#* where: locality#* is defining the <i>locality</i> for which the current (cumulative) num- ber of all created <i>HPX</i> - thread objects should be queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . allocator#* is defin- ing the number of the allo- cator instance using which the threads have been cre- ated. <i>HPX</i> uses a vary- ing number of allocators to create (and recycle) <i>HPX</i> -thread objects, most likely these counters are of use for debugging pur- poses only. The allocator id (given by * is a (zero based) number identifying the allocator to query.	Returns the total num- ber of <i>HPX</i> -thread ob- jects created. Note that thread objects are reused to improve system perfor- mance, thus this number does not reflect the num- ber of actually executed (retired) <i>HPX</i> -threads.	None
/scheduler/ utilization/ instantaneous ??	locality#*/total where: locality#* is defining the <i>locality</i> for which the current (instantaneous) scheduler utilization queried for. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> .	Returns the total (instantaneous) scheduler utilization. This is the current percentage of scheduler threads executing <i>HPX</i> threads.	Percent

continues on next page

Table 2.29 – continued from previous page

/threads/ idle-loop-count/ instantaneous ??	locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the current current accumulated value of all idle-loop counters of all worker threads should be queried. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the current value of the idle-loop counter should be queried for. The worker thread number (given by the * is a (zero based) worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the current (instantaneous) idle-loop count for the given HPX-worker thread or the accumulated value for all worker threads.	None
--	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/ busy-loop-count/ instantaneous ??	locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the <i>locality</i> for which the current accumulated value of all busy-loop counters of all worker threads should be queried. The <i>locality</i> id (given by * is a (zero based) number identifying the <i>locality</i> . pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the current value of the busy-loop counter should be queried for. The worker thread number (given by the * is a (zero based) worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> . If no pool-name is specified the counter refers to the ‘default’ pool.	Returns the current (instantaneous) busy-loop count for the given HPX-worker thread or the accumulated value for all worker threads.	None
--	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/background-work-duration??	locality#*/total locality#*/worker-thread#* where: locality#* is defining the locality for which the overall time spent performing background work should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. worker-thread#* is defining the worker thread for which the overall time spent performing background work should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code> .	Returns the overall time spent performing background work on the given locality since application start. If the instance name is total the counter returns the overall time spent performing background work for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent performing background work for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].	None
--	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/ background-overhead ??	locality#*/total or locality#*/ worker-thread#* where: locality#* is defining the locality for which the background overhead should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. worker-thread#* is defining the worker thread for which the background overhead should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <i>--hpx:threads</i> .	Returns the background overhead on the given locality since application start. If the instance name is total the counter returns the background overhead for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return background overhead for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure displayed for this counter is 0.1%.	None
--	---	---	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/background-send-duration??	<p>locality#*/total locality#*/worker-thread#*</p> <p>where:</p> <p>locality#* is defining the locality for which the overall time spent performing background work related to sending parcels should be queried for. The locality id (given by *) is a (zero based) number identifying the locality.</p> <p>worker-thread#* is defining the worker thread for which the overall time spent performing background work related to sending parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>.</p>	<p>Returns the overall time spent performing background work related to sending parcels on the given locality since application start. If the instance name is total the counter returns the overall time spent performing background work for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent performing background work for all worker threads separately. This counter is available only if the configuration time constants <code>HPX_WITH_BACKGROUND_THREAD_COUNTERS</code> (default: OFF) and <code>HPX_WITH_THREAD_IDLE_RATES</code> are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. This counter will currently return meaningful values for the MPI parcel-port only.</p>	None
--	---	---	------

continues on next page

Table 2.29 – continued from previous page

/threads/ background-send-overhead ??	locality#*/total locality#*/ worker-thread#* where: locality#* is defining the locality for which the background overhead re- lated to sending parcels should be queried for. The locality id (given by *) is a (zero based) number iden- tifying the locality. worker-thread#* is defining the worker thread for which the background overhead related to sending parcels should be queried for. The worker thread num- ber (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the ap- plication using the option <i>--hpx:threads</i> .	Returns the background overhead related to sending parcels on the given locality since ap- plication start. If the instance name is total the counter returns the background overhead for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return background overhead for all worker threads separately. This counter is available only if the con- figuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of mea- sure displayed for this counter is 0.1%. This counter will cur- rently return meaningful values for the MPI parcel- port only.	None
---	--	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/time/ background-receive ??	locality#*/total duration locality#*/ worker-thread#* where: locality#* is defining the locality for which the overall time spent performing background work related to receiving parcels should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. worker-thread#* is defining the worker thread for which the overall time spent per- forming background work related to receiving parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the app- lication using the option <i>--hpx:threads</i> .	Returns the overall time spent performing back- ground work related to receiving parcels on the given locality since application start. If the instance name is total the counter returns the overall time spent per- forming background work for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent per- forming background work for all worker threads separately. This counter is available only if the con- figuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of mea- sure for this counter is nanosecond [ns]. This counter will cur- rently return meaningful values for the MPI parcel- port only.	None
--	---	--	------

continues on next page

Table 2.29 – continued from previous page

/threads/ background-receive ??	locality#*/total overhead locality#*/ worker-thread#* where: locality#* is defining the locality for which the background overhead re- lated to receiving should be queried for. The lo- cality id (given by *) is a (zero based) number iden- tifying the locality. worker-thread#* is defining the worker thread for which the background overhead related to receiving parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the ap- plication using the option <i>--hpx:threads</i> .	Returns the background overhead related to re- ceiving parcels on the given locality since ap- plication start. If the instance name is total the counter returns the background overhead for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return background overhead for all worker threads separately. This counter is available only if the configura- tion time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of mea- sure displayed for this counter is 0.1%. This counter will cur- rently return meaningful values for the MPI parcel- port only.	None
---------------------------------------	--	---	------

Table 2.30: General performance counters exposing characteristics of localities

Counter type	Counter instance formatting	Description	Parameters
/runtime/count/component ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the number of components should be queried. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the overall number of currently active components of the specified type on the given <i>locality</i> .	The type of the component. This is the string which has been used while registering the component with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_COMPONENT</i> .
/runtime/count/action-invocation ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the number of action invocations should be queried. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the overall (local) invocation count of the specified action type on the given <i>locality</i> .	The action type. This is the string which has been used while registering the action with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_ACTION</i> or <i>HPX_REGISTER_ACTION_ID</i> .
/runtime/count/remote-action-invocation ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the number of action invocations should be queried. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the overall (remote) invocation count of the specified action type on the given <i>locality</i> .	The action type. This is the string which has been used while registering the action with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_ACTION</i> or <i>HPX_REGISTER_ACTION_ID</i> .
/runtime/uptime ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the system uptime should be queried. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the overall time since application start on the given <i>locality</i> in nanoseconds.	None
/runtime/memory/virtual ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the allocated virtual memory should be queried. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the amount of virtual memory currently allocated by the referenced <i>locality</i> (in bytes).	None
/runtime/memory/resident ??	locality#*/total where: * is the <i>locality</i> id of the <i>locality</i> the allocated resident memory should be queried. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the amount of resident memory currently allocated by the referenced <i>locality</i> (in bytes).	None
196		Chapter 2. What's so special about HPX?	
/runtime/memory/total	locality#*/total where:	Returns the total available <i>locality</i> (in bytes)	None

Table 2.31: Performance counters exposing PAPI hardware counters

Counter type	Counter instance formatting	Description	Parameters
/papi/<papi_event> ?? where: <papi_event> is the name of the PAPI event to expose as a performance counter (such as PAPI_SR_INS). Note that the list of available PAPI events changes depending on the used architecture. For a full list of available PAPI events and their (short) description use the --hpx:list-counters and --hpx:papi-event-info= command line options.	locality#*/total or locality#*/worker-thread#* where: locality#* is defining the <i>locality</i> for which the current accumulated value of all busy-loop counters of all worker threads should be queried. The <i>locality</i> id (given by *) is a (zero based) number identifying the <i>locality</i> . worker-thread#* is defining the worker thread for which the current value of the busy-loop counter should be queried for. The worker thread number (given by the *) is a (zero based) worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads.	This counter returns the current count of occurrences of the specified PAPI event. This counter is available only if the configuration time constant HPX_WITH_PAPI is set to ON (default: OFF).	None

Table 2.32: Performance counters for general statistics

Counter type	Counter instance format-	Description	Parameters
/?	Any full performance counter average name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.	Returns the current average (mean) value calculated based on the values queried from the underlying counter (the one specified as the instance name).	Any parameter will be interpreted as a list of up to two comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.
/?	Any full performance counter rolling average name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.	Returns the current rolling average (mean) value calculated based on the values queried from the underlying counter (the one specified as the instance name).	Any parameter will be interpreted as a list of up to three comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value will be interpreted as the size of the rolling window (the number of latest values to use to calculate the rolling average). The default value for this is 10. The third value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.
/?	Any full performance counter stddev name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.	Returns the current standard deviation (stddev) value calculated based on the values queried from the underlying counter (the one specified as the instance name).	Any parameter will be interpreted as a list of up to two comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.
/?	Any full performance counter rolling stddev name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.	Returns the current rolling variance (stddev) value calculated based on the values queried from the underlying counter (the one specified as the instance name).	Any parameter will be interpreted as a list of up to three comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value will be interpreted as the size of the rolling window (the number of latest values to use to calculate the rolling average). The default value for this is 10. The third value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.
/?	Any full performance counter median name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.	Returns the current (statistically estimated) median value calculated based on the values queried from the underlying counter (the one specified as the instance name).	Any parameter will be interpreted as a list of up to two comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.
198			Chapter 2. What's so special about HPX?

Table 2.33: Performance counters for elementary arithmetic operations

Counter type	Counter instance formatting	Description	Parameters
/arithmetics/add ??	None	Returns the sum calculated based on the values queried from the underlying counters (the ones specified as the parameters).	The parameter will be interpreted as a comma separated list of full performance counter names which are queried whenever this counter is accessed. Any wildcards in the counter names will be expanded.
/arithmetics/subtract ??	None	Returns the difference calculated based on the values queried from the underlying counters (the ones specified as the parameters).	The parameter will be interpreted as a comma separated list of full performance counter names which are queried whenever this counter is accessed. Any wildcards in the counter names will be expanded.
/arithmetics/multiply ??	None	Returns the product calculated based on the values queried from the underlying counters (the ones specified as the parameters).	The parameter will be interpreted as a comma separated list of full performance counter names which are queried whenever this counter is accessed. Any wildcards in the counter names will be expanded.
/arithmetics/divide ??	None	Returns the result of division of the values queried from the underlying counters (the ones specified as the parameters).	The parameter will be interpreted as a comma separated list of full performance counter names which are queried whenever this counter is accessed. Any wildcards in the counter names will be expanded.
/arithmetics/mean ??	None	Returns the average value of all values queried from the underlying counters (the ones specified as the parameters).	The parameter will be interpreted as a comma separated list of full performance counter names which are queried whenever this counter is accessed. Any wildcards in the counter names will be expanded.
/arithmetics/variance ??	None	Returns the standard deviation of all values queried from the underlying counters (the ones specified as the parameters).	The parameter will be interpreted as a comma separated list of full performance counter names which are queried whenever this counter is accessed. Any wildcards in the counter names will be expanded.
2.3. Manual			199
/arithmetics/median	None	Returns the median value of all values queried from	The parameter will be interpreted as a comma sep-

Note: The `/arithmetics` counters can consume an arbitrary number of other counters. For this reason those have to be specified as parameters (a comma separated list of counters appended after a '@'). For instance:

```
$ ./bin/hello_world_distributed -t2 \
    --hpx:print-counter=/threads{locality#0/worker-thread#*}/count/cumulative \
    --hpx:print-counter=/arithmetics/add@/threads{locality#0/worker-thread#*}/count/
    ↵cumulative
hello world from OS-thread 0 on locality 0
hello world from OS-thread 1 on locality 0
/threads{locality#0/worker-thread#0}/count/cumulative,1,0.515640,[s],25
/threads{locality#0/worker-thread#1}/count/cumulative,1,0.515520,[s],36
/arithmetics/add@/threads{locality#0/worker-thread#*}/count/cumulative,1,0.516445,[s],
    ↵64
```

Since all wildcards in the parameters are expanded, this example is fully equivalent to specifying both counters separately to `/arithmetics/add`:

```
$ ./bin/hello_world_distributed -t2 \
    --hpx:print-counter=/threads{locality#0/worker-thread#*}/count/cumulative \
    --hpx:print-counter=/arithmetics/add@\
        /threads{locality#0/worker-thread#0}/count/cumulative, \
        /threads{locality#0/worker-thread#1}/count/cumulative
```

Table 2.34: Performance counters tracking parcel coalescing

Counter type	Counter instance formatting	Description	Parameters
/coalesced/count/where: parcels is the <i>locality</i> ??	locality#*/coalesced/count/where: parcels is the <i>locality</i> id of the <i>locality</i> the number of parcels for the given action should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the number of parcels handled by the message handler associated with the action which is given by the counter parameter.	The action type. This is the string which has been used while registering the action with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_ACTION</i> or <i>HPX_REGISTER_ACTION_ID</i> .
/coalesced/count/where: messages is the <i>locality</i> ??	locality#*/coalesced/count/where: messages is the <i>locality</i> id of the <i>locality</i> the number of messages for the given action should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the number of messages generated by the message handler associated with the action which is given by the counter parameter.	The action type. This is the string which has been used while registering the action with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_ACTION</i> or <i>HPX_REGISTER_ACTION_ID</i> .
/coalesced/count/where: average is the <i>locality</i> ??	locality#*/coalesced/count/where: average is the <i>locality</i> id of the <i>locality</i> the number of messages for the given action should be queried for. The <i>locality</i> id is a (zero based) number identifying the <i>locality</i> .	Returns the average number of parcels sent in a message generated by the message handler associated with the action which is given by the counter parameter.	The action type. This is the string which has been used while registering the action with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_ACTION</i> or <i>HPX_REGISTER_ACTION_ID</i>
2.3. Manual	locality#*/coalesced/time/where: average is the <i>locality</i> -arrival	Returns the average time between arriving parcels for the action which is given by the counter parameter.	The action type. This is the string which has been used while registering the action with <i>HPX</i> , e.g. which has been passed as the second parameter to the macro <i>HPX_REGISTER_ACTION</i> or <i>HPX_REGISTER_ACTION_ID</i> . 201

Note: The performance counters related to *parcel* coalescing are available only if the configuration time constant `HPX_WITH_PARCEL_COALESCING` is set to ON (default: ON). However, even in this case it will be available only for actions that are enabled for parcel coalescing (see the macros `HPX_ACTIONUSESMESSAGECOALESCING` and `HPX_ACTIONUSESMESSAGECOALESCING_NOTHROW`).

APEX integration

HPX provides integration with [APEX](#)¹³⁵, which is a framework for application profiling using task timers and various performance counters Huck *et al.*¹³⁸. It can be added as a git submodule by turning on the option `HPX_WITH_APEX:BOOL` during CMake configuration. [TAU](#)¹³⁶ is an optional dependency when using APEX.

To build *HPX* with APEX, add `HPX_WITH_APEX=ON`, and, optionally, `TAU_ROOT=$PATH_TO_TAU` to your CMake configuration. In addition, you can override the tag used for APEX with the `HPX_WITH_APEX_TAG` option. Please see the [APEX HPX documentation](#)¹³⁷ for detailed instructions on using APEX with *HPX*.

References

2.3.13 *HPX* runtime and resources

HPX thread scheduling policies

The *HPX* runtime has five thread scheduling policies: local-priority, static-priority, local, static and abp-priority. These policies can be specified from the command line using the command line option `--hpx:queuing`. In order to use a particular scheduling policy, the runtime system must be built with the appropriate scheduler flag turned on (e.g. `cmake -DHPX_THREAD_SCHEDULERS=local`, see [CMake variables used to configure HPX](#) for more information).

Priority local scheduling policy (default policy)

The priority local scheduling policy maintains one queue per operating system (OS) thread. The OS thread pulls its work from this queue. By default the number of high priority queues is equal to the number of OS threads; the number of high priority queues can be specified on the command line using `--hpx:high-priority-threads`. High priority threads are executed by any of the OS threads before any other work is executed. When a queue is empty, work will be taken from high priority queues first. There is one low priority queue from which threads will be scheduled only when there is no other work.

For this scheduling policy there is an option to turn on NUMA sensitivity using the command line option `--hpx:numa-sensitive`. When NUMA sensitivity is turned on, work stealing is done from queues associated with the same NUMA domain first, only after that work is stolen from other NUMA domains.

This scheduler is enabled at build time by default using the FIFO (first-in-first-out) queing policy. This policy can be invoked using `--hpx:queuinglocal-priority-fifo`. The scheduler can also be enabled using the LIFO (last-in-first-out) policy. This is not the default policy and must be invoked using the command line option `--hpx:queuing=local-priority-lifo`.

¹³⁵ <http://uo-oaciss.github.io/apex>

¹³⁸ K. A. Huck, A. Porterfield, N Chaimov, H. Kaiser, A. D. Malony, T. Sterling, and R. Fowler. *An autonomic performance environment for exascale*. Supercomputing Frontiers and Innovations, 2015.

¹³⁶ <https://www.cs.uoregon.edu/research/tau/home.php>

¹³⁷ <https://uo-oaciss.github.io/apex/usage/#hpx-louisiana-state-university>

Static priority scheduling policy

- invoke using: `--hpx:queuing=static-priority` (or `-qs`)
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=static-priority`

The static scheduling policy maintains one queue per OS thread from which each OS thread pulls its tasks (user threads). Threads are distributed in a round robin fashion. There is no thread stealing in this policy.

Local scheduling policy

- invoke using: `--hpx:queuing=local` (or `-ql`)
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=local`

The local scheduling policy maintains one queue per OS thread from which each OS thread pulls its tasks (user threads).

Static scheduling policy

- invoke using: `--hpx:queuing=static`
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=static`

The static scheduling policy maintains one queue per OS thread from which each OS thread pulls its tasks (user threads). Threads are distributed in a round robin fashion. There is no thread stealing in this policy.

Priority ABP scheduling policy

- invoke using: `--hpx:queuing=abp-priority-fifo`
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=abp-priority`

Priority ABP policy maintains a double ended lock free queue for each OS thread. By default the number of high priority queues is equal to the number of OS threads; the number of high priority queues can be specified on the command line using `--hpx:high-priority-threads`. High priority threads are executed by the first OS threads before any other work is executed. When a queue is empty work will be taken from high priority queues first. There is one low priority queue from which threads will be scheduled only when there is no other work. For this scheduling policy there is an option to turn on NUMA sensitivity using the command line option `--hpx:numa-sensitive`. When NUMA sensitivity is turned on work stealing is done from queues associated with the same NUMA domain first, only after that work is stolen from other NUMA domains.

This scheduler can be used with two underlying queuing policies (FIFO: first-in-first-out, and LIFO: last-in-first-out). In order to use the LIFO policy use the command line option `--hpx:queuing=abp-priority-lifo`.

The HPX resource partitioner

The *HPX* resource partitioner lets you take the execution resources available on a system—processing units, cores, and numa domains—and assign them to thread pools. By default *HPX* creates a single thread pool name `default`. While this is good for most use cases, the resource partitioner lets you create multiple thread pools with custom resources and options.

Creating custom thread pools is useful for cases where you have tasks which absolutely need to run without interference from other tasks. An example of this is when using [MPI¹³⁹](#) for distribution instead of the built-in mechanisms in *HPX* (useful in legacy applications). In this case one can create a thread pool containing a single thread for MPI communication. MPI tasks will then always run on the same thread, instead of potentially being stuck in a queue behind other threads.

Note that *HPX* thread pools are completely independent from each other in the sense that task stealing will never happen between different thread pools. However, tasks running on a particular thread pool can schedule tasks on another thread pool.

Note: It is simpler in some situations to schedule important tasks with high priority instead of using a separate thread pool.

Using the resource partitioner

The `hpx::resource::partitioner` is now created during *HPX* runtime initialization without explicit action needed from the user. To specify some of the initialization parameters you can use the `hpx::init_params`.

The resource partitioner callback is the interface to add thread pools to the *HPX* runtime and to assign resources to the thread pools. In order to create custom thread pools you can specify the resource partitioner callback `hpx::init_params::rp_callback` which will be called once the resource partitioner will be created , see the example below. You can also specify other parameters, see `hpx::init_params`.

To add a thread pool use the `hpx::resource::partitioner::create_thread_pool` method. If you simply want to use the default scheduler and scheduler options, it is enough to call `rp.create_thread_pool("my-thread-pool")`.

Then, to add resources to the thread pool you can use the `hpx::resource::partitioner::add_resource` method. The resource partitioner exposes the hardware topology retrieved using [Portable Hardware Locality \(HWLOC\)](#)¹⁴⁰ and lets you iterate through the topology to add the wanted processing units to the thread pool. Below is an example of adding all processing units from the first NUMA domain to a custom thread pool, unless there is only one NUMA domain in which case we leave the first processing unit for the default thread pool:

Note: Whatever processing units are not assigned to a thread pool by the time `hpx::init` is called will be added to the default thread pool. It is also possible to explicitly add processing units to the default thread pool, and to create the default thread pool manually (in order to e.g. set the scheduler type).

Tip: The command line option `--hpx:print-bind` is useful for checking that the thread pools have been set up the way you expect.

¹³⁹ https://en.wikipedia.org/wiki/Message_Passing_Interface

¹⁴⁰ <https://www.open-mpi.org/projects/hwloc/>

Difference between the old and new version

In the old version, you had to create an instance of the `resource_partitioner` with `argc` and `argv`.

```
int main(int argc, char** argv)
{
    hpx::resource::partitioner rp(argc, argv);
    hpx::init();
}
```

From *HPX 1.5.0* onwards, you just pass `argc` and `argv` to `hpx::init()` or `hpx::start()` for the binding options to be parsed by the resource partitioner.

```
int main(int argc, char** argv)
{
    hpx::init_params init_args;
    hpx::init(argc, argv, init_args);
}
```

In the old version, when creating a custom thread pool, you just called the utilities on the resource partitioner instantiated previously.

```
int main(int argc, char** argv)
{
    hpx::resource::partitioner rp(argc, argv);

    rp.create_thread_pool("my-thread-pool");

    bool one numa_domain = rp.numa_domains().size() == 1;
    bool skipped_first_pu = false;

    hpx::resource::numa_domain const& d = rp.numa_domains()[0];

    for (const hpx::resource::core& c : d.cores())
    {
        for (const hpx::resource::pu& p : c.pus())
        {
            if (one numa_domain && !skipped_first_pu)
            {
                skipped_first_pu = true;
                continue;
            }

            rp.add_resource(p, "my-thread-pool");
        }
    }

    hpx::init();
}
```

You now specify the resource partitioner callback which will tie the resources to the resource partitioner created during runtime initialization.

```
void init_resource_partitioner_handler(hpx::resource::partitioner& rp)
{
    rp.create_thread_pool("my-thread-pool");
```

(continues on next page)

(continued from previous page)

```

bool one numa_domain = rp.numa_domains().size() == 1;
bool skipped_first_pu = false;

hpx::resource::numa_domain const& d = rp.numa_domains()[0];

for (const hpx::resource::core& c : d.cores())
{
    for (const hpx::resource::pu& p : c.pus())
    {
        if (one numa_domain && !skipped_first_pu)
        {
            skipped_first_pu = true;
            continue;
        }

        rp.add_resource(p, "my-thread-pool");
    }
}

int main(int argc, char* argv[])
{
    hpx::init_params init_args;
    init_args.rp_callback = &init_resource_partitioner_handler;

    hpx::init(argc, argv, init_args);
}

```

Advanced usage

It is possible to customize the built in schedulers by passing scheduler options to `hpx::resource::partitioner::create_thread_pool`. It is also possible to create and use custom schedulers.

Note: It is not recommended to create your own scheduler. The HPX developers use this to experiment with new scheduler designs before making them available to users via the standard mechanisms of choosing a scheduler (command line options). If you would like to experiment with a custom scheduler the resource partitioner example `shared_priority_queue_scheduler.cpp` contains a fully implemented scheduler with logging, etc. to make exploration easier.

To choose a scheduler and custom mode for a thread pool, pass additional options when creating the thread pool like this:

```

rp.create_thread_pool("my-thread-pool",
    hpx::resource::policies::local_priority_lifo,
    hpx::policies::scheduler_mode(
        hpx::policies::scheduler_mode::default_ |
        hpx::policies::scheduler_mode::enable_elasticity));

```

The available schedulers are documented here: `hpx::resource::scheduling_policy`, and the available scheduler modes here: `hpx::threads::policies::scheduler_mode`. Also see the examples folder for examples of advanced resource partitioner usage: `simple_resource_partitioner.cpp` and `oversubscribing_resource_partitioner.cpp`.

2.3.14 Miscellaneous

Error handling

Like in any other asynchronous invocation scheme, it is important to be able to handle error conditions occurring while the asynchronous (and possibly remote) operation is executed. In *HPX* all error handling is based on standard C++ exception handling. Any exception thrown during the execution of an asynchronous operation will be transferred back to the original invocation *locality*, where it will be rethrown during synchronization with the calling thread.

The source code for this example can be found here: `error_handling.cpp`.

Working with exceptions

For the following description assume that the function `raise_exception()` is executed by invoking the plain action `raise_exception_type`.

```
#include <hpx/iostream.hpp>
#include <hpx/modules/runtime_local.hpp>

// [[error_handling_raise_exception
void raise_exception()
```

The exception is thrown using the macro `HPX_THROW_EXCEPTION`. The type of the thrown exception is `hpx::exception`. This associates additional diagnostic information with the exception, such as file name and line number, *locality* id and thread id, and stack backtrace from the point where the exception was thrown.

Any exception thrown during the execution of an action is transferred back to the (asynchronous) invocation site. It will be rethrown in this context when the calling thread tries to wait for the result of the action by invoking either `future<>::get()` or the synchronous action invocation wrapper as shown here:

```
{
    ///////////////////////////////////////////////////////////////////
    // Error reporting using exceptions
    // [[exception_diagnostic_information
    hpx::cout << "Error reporting using exceptions\n";
    try
    {
        // invoke raise_exception() which throws an exception
        raise_exception_action do_it;
        do_it(hpx::find_here());
    }
    catch (hpx::exception const& e)
    {
        // Print just the essential error information.
        hpx::cout << "caught exception: " << e.what() << "\n\n";
```

Note: The exception is transferred back to the invocation site even if it is executed on a different *locality*.

Additionally, this example demonstrates how an exception thrown by an (possibly remote) action can be handled. It shows the use of `hpx::diagnostic_information`, which retrieves all available diagnostic information from the exception as a formatted string. This includes, for instance, the name of the source file and line number, the sequence number of the OS thread and the *HPX* thread id, the *locality* id and the stack backtrace of the point where the original exception was thrown.

Under certain circumstances it is desirable to output only some of the diagnostics, or to output those using different formatting. For this case, *HPX* exposes a set of lower-level functions as demonstrated in the following code snippet:

```
}

hpx::cout << std::flush;
//]

// Detailed error reporting using exceptions
//[exception_diagnostic_elements
hpx::cout << "Detailed error reporting using exceptions\n";
try
{
    // Invoke raise_exception() which throws an exception.
    raise_exception_action do_it;
    do_it(hpx::find_here());
}
catch (hpx::exception const& e)
{
    // Print the elements of the diagnostic information separately.
    hpx::cout << "{what}: " << hpx::get_error_what(e) << "\n";
    hpx::cout << "{locality-id}: " << hpx::get_error_locality_id(e)
        << "\n";
    hpx::cout << "{hostname}: " << hpx::get_error_host_name(e) << "\n";
    hpx::cout << "{pid}: " << hpx::get_error_process_id(e) << "\n";
    hpx::cout << "{function}: " << hpx::get_error_function_name(e)
        << "\n";
    hpx::cout << "{file}: " << hpx::get_error_file_name(e) << "\n";
    hpx::cout << "{line}: " << hpx::get_error_line_number(e) << "\n";
    hpx::cout << "{os-thread}: " << hpx::get_error_os_thread(e) << "\n";
```

Working with error codes

Most of the API functions exposed by *HPX* can be invoked in two different modes. By default those will throw an exception on error as described above. However, sometimes it is desirable not to throw an exception in case of an error condition. In this case an object instance of the `hpx::error_code` type can be passed as the last argument to the API function. In case of an error, the error condition will be returned in that `hpx::error_code` instance. The following example demonstrates extracting the full diagnostic information without exception handling:

```
hpx::cout << "{stack-trace}: " << hpx::get_error_backtrace(e)
    << "\n";
hpx::cout << "{env}: " << hpx::get_error_env(e) << "\n";
}

hpx::cout << std::flush;
//]

///////////////////////////////
// Error reporting using error code
{
    //#[error_handling_diagnostic_information
    hpx::cout << "Error reporting using error code\n";

    // Create a new error_code instance.
    hpx::error_code ec;

    // If an instance of an error_code is passed as the last argument while
    // invoking the action, the function will not throw in case of an error
```

(continues on next page)

(continued from previous page)

```
// but store the error information in this error_code instance instead.
raise_exception_action do_it;
do_it(hpx::find_here(), ec);
```

Note: The error information is transferred back to the invocation site even if it is executed on a different *locality*.

This example show how an error can be handled without having to resolve to exceptions and that the returned `hpx::error_code` instance can be used in a very similar way as the `hpx::exception` type above. Simply pass it to the `hpx::diagnostic_information`, which retrieves all available diagnostic information from the error code instance as a formatted string.

As for handling exceptions, when working with error codes, under certain circumstances it is desirable to output only some of the diagnostics, or to output those using different formatting. For this case, *HPX* exposes a set of lower-level functions usable with error codes as demonstrated in the following code snippet:

```
// the exception.
hpx::cout << "diagnostic information:"
    << hpx::diagnostic_information(ec) << "\n";
}

hpx::cout << std::flush;
//]
}

// Detailed error reporting using error code
{
    // [error_handling_diagnostic_elements
    hpx::cout << "Detailed error reporting using error code\n";

    // Create a new error_code instance.
    hpx::error_code ec;

    // If an instance of an error_code is passed as the last argument while
    // invoking the action, the function will not throw in case of an error
    // but store the error information in this error_code instance instead.
    raise_exception_action do_it;
    do_it(hpx::find_here(), ec);

    if (ec)
    {
        // Print the elements of the diagnostic information separately.
        hpx::cout << "{what}: " << hpx::get_error_what(ec) << "\n";
        hpx::cout << "{locality-id}: " << hpx::get_error_locality_id(ec)
            << "\n";
        hpx::cout << "{hostname}: " << hpx::get_error_host_name(ec)
            << "\n";
        hpx::cout << "{pid}: " << hpx::get_error_process_id(ec) << "\n";
        hpx::cout << "{function}: " << hpx::get_error_function_name(ec)
    }
}
```

For more information please refer to the documentation of `hpx::get_error_what`, `hpx::get_error_locality_id`, `hpx::get_error_host_name`, `hpx::get_error_process_id`, `hpx::get_error_function_name`, `hpx::get_error_file_name`, `hpx::get_error_line_number`, `hpx::get_error_os_thread`, `hpx::get_error_thread_id`, `hpx::get_error_thread_description`, `hpx::get_error_backtrace`, `hpx::get_error_env`, and `hpx::get_error_state`.

Lightweight error codes

Sometimes it is not desirable to collect all the ambient information about the error at the point where it happened as this might impose too much overhead for simple scenarios. In this case, *HPX* provides a lightweight error code facility that will hold the error code only. The following snippet demonstrates its use:

```
    hpx::cout << "{thread-id}: " << std::hex
                  << hpx::get_error_thread_id(ec) << "\n";
    hpx::cout << "{thread-description}: "
                  << hpx::get_error_thread_description(ec) << "\n\n";
    hpx::cout << "{state}: " << std::hex << hpx::get_error_state(ec)
                  << "\n";
    hpx::cout << "{stack-trace}: " << hpx::get_error_backtrace(ec)
                  << "\n";
    hpx::cout << "{env}: " << hpx::get_error_env(ec) << "\n";
}

    hpx::cout << std::flush;
//]
}
```

// Error reporting using lightweight error code

```
{
    //[[lightweight_error_handling_diagnostic_information
    hpx::cout << "Error reporting using an lightweight error code\n";

    // Create a new error_code instance.
```

All functions that retrieve other diagnostic elements from the `hpx::error_code` will fail if called with a lightweight error_code instance.

Utilities in *HPX*

In order to ease the burden of programming, *HPX* provides several utilities to users. The following section documents those facilities.

Checkpoint

See *checkpoint*.

The *HPX* I/O-streams component

The *HPX* I/O-streams subsystem extends the standard C++ output streams `std::cout` and `std::cerr` to work in the distributed setting of an *HPX* application. All of the output streamed to `hpx::cout` will be dispatched to `std::cout` on the console *locality*. Likewise, all output generated from `hpx::cerr` will be dispatched to `std::cerr` on the console *locality*.

Note: All existing standard manipulators can be used in conjunction with `hpx::cout` and `hpx::cerr`.

In order to use either `hpx::cout` or `hpx::cerr`, application codes need to `#include <hpx/include/iostreams.hpp>`. For an example, please see the following ‘Hello world’ program:

```

// Copyright (c) 2007-2012 Hartmut Kaiser
//
// SPDX-License-Identifier: BSL-1.0
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

///////////////////////////////
// The purpose of this example is to execute a HPX-thread printing
// "Hello World!" once. That's all.

///[hello_world_1_getting_started
// Including 'hpx/hpx_main.hpp' instead of the usual 'hpx/hpx_init.hpp' enables
// to use the plain C-main below as the direct main HPX entry point.
#include <hpx/hpx_main.hpp>
#include <hpx/iostream.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << std::flush;
    return 0;
}
//]

```

Additionally, those applications need to link with the iostreams component. When using CMake this can be achieved by using the COMPONENT_DEPENDENCIES parameter; for instance:

```

include(HPX_AddExecutable)

add_hpx_executable(
    hello_world
    SOURCES hello_world.cpp
    COMPONENT_DEPENDENCIES iostreams
)

```

Note: The `hpx::cout` and `hpx::cerr` streams buffer all output locally until a `std::endl` or `std::flush` is encountered. That means that no output will appear on the console as long as either of these is explicitly used.

2.3.15 Troubleshooting

Common issues

This section contains commonly encountered problems when compiling or using HPX.

See also the closed issues on [GitHub¹⁴¹](#) to find out how other people resolved a similar problem. If nothing of that works, you can also open a new issue on [GitHub¹⁴²](#) or contact us using one the options found in [Support for deploying and using HPX¹⁴³](#).

¹⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues?q=is%3Aissue+is%3Aclosed>

¹⁴² <https://github.com/STELLAR-GROUP/hpx/issues>

¹⁴³ <https://github.com/STELLAR-GROUP/hpx/blob/master/.github/SUPPORT.md>

Undefined reference to `hpx::cout`

You may see a linker error message that looks a bit like this:

```
hello_world.cpp:(.text+0x5aa): undefined reference to `hpx::cout'
```

This usually happens if you are trying to use *HPX* iostreams functionality such as `hpx::cout` but are not linking against it. The iostreams functionality is not part of the core *HPX* library, and must be linked to explicitly. Typically this can be solved by adding `COMPONENT_DEPENDENCIES iostreams` to a call to `add_hpx_library/` `add_hpx_executable/hpx_setup_target` if using CMake. See [Creating HPX projects](#) for more details.

Fail compiling for examples with `hpx::future` and `co_await`

You may see an error message that looks a bit like this:

```
error: coroutines require a traits template; cannot find 'std::coroutine_traits'
```

This can be resolved by using `-DHPX_WITH_CXX_STANDARD=20` to the cmake command line. Note that a compiler that supports C++20 is needed.

See also the corresponding closed Issue #5784¹⁴⁴.

Build fails with ASIO error

You may see an error message that looks a bit like this:

```
Cannot open include file: asio/io_context.hpp
```

This can be resolved by using `-DHPX_WITH_FETCH_ASIO=ON` to the cmake command line.

See also the corresponding closed Issue #5404¹⁴⁵ for more information.

Build fails with TCMalloc error

You may see an error message that looks a bit like this:

```
Could NOT find TCMalloc (missing: TCMALLOC_LIBRARY TCMALLOC_INCLUDE_DIR)
ERROR: HPX_WITH_MALLOC was set to tcmalloc, but tcmalloc could not be
found. Valid options for HPX_WITH_MALLOC are: system, tcmalloc, jemalloc,
mimalloc, tbbmalloc, and custom
```

This can be resolved either by defining `HPX_WITH_MALLOC=system` or by installing TCMalloc. This error occurs when users don't specify an option for `HPX_WITH_MALLOC`; in that case, *HPX* will be looking `tcmalloc`, which is the default value.

¹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5784>

¹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5404>

Useful suggestions

Reducing compilation time

If you want to significantly reduce compilation time, you can just use the local part of *HPX* for parallelism by disabling the distributed functionality. Moreover, you can avoid compiling examples. These can be done with the following flags:

```
-DHPX_WITH_NETWORKING=OFF
-DHPX_WITH_DISTRIBUTED_RUNTIME=OFF
-DHPX_WITH_EXAMPLES=OFF
-DHPX_WITH_TESTS=OFF
```

Linking *HPX* to your application

If you want to avoid installing and linking *HPX*, you can just build *HPX* and then use the following flag on your *HPX* application CMake configuration:

```
-DHPX_DIR=<build_dir>/lib/cmake/HPX
```

Note: For this to work you need not to specify `-DCMAKE_INSTALL_PREFIX` when building *HPX*.

HPX-application build type conformance

Your application's build type should align with the *HPX* build type. For example, if you specified `-DCMAKE_BUILD_TYPE=Debug` during the *HPX* compilation, then your application needs to be compiled with the same flag. We recommend keeping a separate build folder for different build types and just point accordingly to the type you want by using `-DHPX_DIR=<build_dir>/lib/cmake/HPX`.

2.4 Terminology

This section gives definitions for some of the terms used throughout the *HPX* documentation and source code.

Locality A locality in *HPX* describes a synchronous domain of execution, or the domain of bounded upper response time. This normally is just a single node in a cluster or a NUMA domain in a SMP machine.

Active Global Address Space

AGAS *HPX* incorporates a global address space. Any executing thread can access any object within the domain of the parallel application with the caveat that it must have appropriate access privileges. The model does not assume that global addresses are cache coherent; all loads and stores will deal directly with the site of the target object. All global addresses within a Synchronous Domain are assumed to be cache coherent for those processor cores that incorporate transparent caches. The Active Global Address Space used by *HPX* differs from research PGAS¹⁴⁶ models. Partitioned Global Address Space is passive in their means of address translation. Copy semantics, distributed compound operations, and affinity relationships are some of the global functionality supported by AGAS.

¹⁴⁶ <https://www.pgas.org/>

Process The concept of the “process” in *HPX* is extended beyond that of either sequential execution or communicating sequential processes. While the notion of process suggests action (as do “function” or “subroutine”) it has a further responsibility of context, that is, the logical container of program state. It is this aspect of operation that process is employed in *HPX*. Furthermore, referring to “parallel processes” in *HPX* designates the presence of parallelism within the context of a given process, as well as the coarse grained parallelism achieved through concurrency of multiple processes of an executing user job. *HPX* processes provide a hierarchical name space within the framework of the active global address space and support multiple means of internal state access from external sources.

Parcel The Parcel is a component in *HPX* that communicates data, invokes an action at a distance, and distributes flow-control through the migration of continuations. Parcels bridge the gap of asynchrony between synchronous domains while maintaining symmetry of semantics between local and global execution. Parcels enable message-driven computation and may be seen as a form of “active messages”. Other important forms of message-driven computation predating active messages include [dataflow tokens](#)¹⁴⁷, the J-machine’s¹⁴⁸ support for remote method instantiation, and at the coarse grained variations of Unix remote procedure calls, among others. This enables work to be moved to the data as well as performing the more common action of bringing data to the work. A parcel can cause actions to occur remotely and asynchronously, among which are the creation of threads at different system nodes or synchronous domains.

Local Control Object

Lightweight Control Object

LCO A local control object (sometimes called a lightweight control object) is a general term for the synchronization mechanisms used in *HPX*. Any object implementing a certain concept can be seen as an LCO. This concepts encapsulates the ability to be triggered by one or more events which when taking the object into a predefined state will cause a thread to be executed. This could either create a new thread or resume an existing thread.

The LCO is a family of synchronization functions potentially representing many classes of synchronization constructs, each with many possible variations and multiple instances. The LCO is sufficiently general that it can subsume the functionality of conventional synchronization primitives such as spinlocks, mutexes, semaphores, and global barriers. However due to the rich concept an LCO can represent powerful synchronization and control functionality not widely employed, such as dataflow and futures (among others), which open up enormous opportunities for rich diversity of distributed control and operation.

See [Using LCOs](#) for more details on how to use LCOs in *HPX*.

Action An action is a function that can be invoked remotely. In *HPX* a plain function can be made into an action using a macro. See [Applying actions](#) for details on how to use actions in *HPX*.

Component A component is a C++ object which can be accessed remotely. A component can also contain member functions which can be invoked remotely. These are referred to as component actions. See [Writing components](#) for details on how to use components in *HPX*.

2.5 Why *HPX*?

Current advances in high performance computing (HPC) continue to suffer from the issues plaguing parallel computation. These issues include, but are not limited to, ease of programming, inability to handle dynamically changing workloads, scalability, and efficient utilization of system resources. Emerging technological trends such as multi-core processors further highlight limitations of existing parallel computation models. To mitigate the aforementioned problems, it is necessary to rethink the approach to parallelization models. ParalleX contains mechanisms such as multi-threading, *parcels*, *global name space* support, percolation and *local control objects (LCO)*. By design, ParalleX overcomes limitations of current models of parallelism by alleviating contention, latency, overhead and starvation. With ParalleX, it is further possible to increase performance by at least an order of magnitude on challenging

¹⁴⁷ http://en.wikipedia.org/wiki/Dataflow_architecture

¹⁴⁸ <http://en.wikipedia.org/wiki/J%2E2%80%93Machine>

parallel algorithms, e.g., dynamic directed graph algorithms and adaptive mesh refinement methods for astrophysics. An additional benefit of ParalleX is fine-grained control of power usage, enabling reductions in power consumption.

2.5.1 ParalleX—a new execution model for future architectures

ParalleX is a new parallel execution model that offers an alternative to the conventional computation models, such as message passing. ParalleX distinguishes itself by:

- Split-phase transaction model
- Message-driven
- Distributed shared memory (not cache coherent)
- Multi-threaded
- Futures synchronization
- *Local Control Objects (LCOs)*
- Synchronization for anonymous producer-consumer scenarios
- Percolation (pre-staging of task data)

The ParalleX model is intrinsically latency hiding, delivering an abundance of variable-grained parallelism within a hierarchical namespace environment. The goal of this innovative strategy is to enable future systems delivering very high efficiency, increased scalability and ease of programming. ParalleX can contribute to significant improvements in the design of all levels of computing systems and their usage from application algorithms and their programming languages to system architecture and hardware design together with their supporting compilers and operating system software.

2.5.2 What is HPX?

High Performance ParalleX (*HPX*) is the first runtime system implementation of the ParalleX execution model. The *HPX* runtime software package is a modular, feature-complete, and performance-oriented representation of the ParalleX execution model targeted at conventional parallel computing architectures, such as SMP nodes and commodity clusters. It is academically developed and freely available under an open source license. We provide *HPX* to the community for experimentation and application to achieve high efficiency and scalability for dynamic adaptive and irregular computational problems. *HPX* is a C++ library that supports a set of critical mechanisms for dynamic adaptive resource management and lightweight task scheduling within the context of a global address space. It is solidly based on many years of experience in writing highly parallel applications for HPC systems.

The two-decade success of the communicating sequential processes (CSP) execution model and its message passing interface (MPI) programming model have been seriously eroded by challenges of power, processor core complexity, multi-core sockets, and heterogeneous structures of GPUs. Both efficiency and scalability for some current (strong scaled) applications and future Exascale applications demand new techniques to expose new sources of algorithm parallelism and exploit unused resources through adaptive use of runtime information.

The ParalleX execution model replaces CSP to provide a new computing paradigm embodying the governing principles for organizing and conducting highly efficient scalable computations greatly exceeding the capabilities of today's problems. *HPX* is the first practical, reliable, and performance-oriented runtime system incorporating the principal concepts of the ParalleX model publicly provided in open source release form.

HPX is designed by the STE||AR¹⁴⁹ Group (Systems Technology, Emergent Parallelism, and Algorithm Research) at Louisiana State University (LSU)¹⁵⁰'s Center for Computation and Technology (CCT)¹⁵¹ to enable developers to

¹⁴⁹ <https://stellar-group.org>

¹⁵⁰ <https://www.lsu.edu>

¹⁵¹ <https://www.cct.lsu.edu>

exploit the full processing power of many-core systems with an unprecedented degree of parallelism. STE||AR¹⁵² is a research group focusing on system software solutions and scientific application development for hybrid and many-core hardware architectures.

For more information about the STE||AR¹⁵³ Group, see *People*.

2.5.3 What makes our systems slow?

Estimates say that we currently run our computers at well below 100% efficiency. The theoretical peak performance (usually measured in FLOPS¹⁵⁴—floating point operations per second) is much higher than any practical peak performance reached by any application. This is particularly true for highly parallel hardware. The more hardware parallelism we provide to an application, the better the application must scale in order to efficiently use all the resources of the machine. Roughly speaking, we distinguish two forms of scalability: strong scaling (see Amdahl’s Law¹⁵⁵) and weak scaling (see Gustafson’s Law¹⁵⁶). Strong scaling is defined as how the solution time varies with the number of processors for a fixed **total** problem size. It gives an estimate of how much faster we can solve a particular problem by throwing more resources at it. Weak scaling is defined as how the solution time varies with the number of processors for a fixed problem size **per processor**. In other words, it defines how much more data can we process by using more hardware resources.

In order to utilize as much hardware parallelism as possible an application must exhibit excellent strong and weak scaling characteristics, which requires a high percentage of work executed in parallel, i.e., using multiple threads of execution. Optimally, if you execute an application on a hardware resource with N processors it either runs N times faster or it can handle N times more data. Both cases imply 100% of the work is executed on all available processors in parallel. However, this is just a theoretical limit. Unfortunately, there are more things that limit scalability, mostly inherent to the hardware architectures and the programming models we use. We break these limitations into four fundamental factors that make our systems *SLOW*:

- Starvation occurs when there is insufficient concurrent work available to maintain high utilization of all resources.
- Latencies are imposed by the time-distance delay intrinsic to accessing remote resources and services.
- Overhead is work required for the management of parallel actions and resources on the critical execution path, which is not necessary in a sequential variant.
- Waiting for contention resolution is the delay due to the lack of availability of oversubscribed shared resources.

Each of those four factors manifests itself in multiple and different ways; each of the hardware architectures and programming models expose specific forms. However, the interesting part is that all of them are limiting the scalability of applications no matter what part of the hardware jungle we look at. Hand-helds, PCs, supercomputers, or the cloud, all suffer from the reign of the 4 horsemen: Starvation, Latency, Overhead, and Contention. This realization is very important as it allows us to derive the criteria for solutions to the scalability problem from first principles, and it allows us to focus our analysis on very concrete patterns and measurable metrics. Moreover, any derived results will be applicable to a wide variety of targets.

¹⁵² <https://stellar-group.org>

¹⁵³ <https://stellar-group.org>

¹⁵⁴ <http://en.wikipedia.org/wiki/FLOPS>

¹⁵⁵ http://en.wikipedia.org/wiki/Amdahl%27s_law

¹⁵⁶ http://en.wikipedia.org/wiki/Gustafson%27s_law

2.5.4 Technology demands new response

Today's computer systems are designed based on the initial ideas of [John von Neumann](#)¹⁵⁷, as published back in 1945, and later extended by the [Harvard architecture](#)¹⁵⁸. These ideas form the foundation, the execution model, of computer systems we use currently. However, a new response is required in the light of the demands created by today's technology.

So, what are the overarching objectives for designing systems allowing for applications to scale as they should? In our opinion, the main objectives are:

- Performance: as previously mentioned, scalability and efficiency are the main criteria people are interested in.
- Fault tolerance: the low expected mean time between failures ([MTBF](#)¹⁵⁹) of future systems requires embracing faults, not trying to avoid them.
- Power: minimizing energy consumption is a must as it is one of the major cost factors today, and will continue to rise in the future.
- Generality: any system should be usable for a broad set of use cases.
- Programmability: for programmer this is a very important objective, ensuring long term platform stability and portability.

What needs to be done to meet those objectives, to make applications scale better on tomorrow's architectures? Well, the answer is almost obvious: we need to devise a new execution model—a set of governing principles for the holistic design of future systems—targeted at minimizing the effect of the outlined **SLOW** factors. Everything we create for future systems, every design decision we make, every criteria we apply, have to be validated against this single, uniform metric. This includes changes in the hardware architecture we prevalently use today, and it certainly involves new ways of writing software, starting from the operating system, runtime system, compilers, and at the application level. However, the key point is that all those layers have to be co-designed; they are interdependent and cannot be seen as separate facets. The systems we have today have been evolving for over 50 years now. All layers function in a certain way, relying on the other layers to do so. But we do not have the time to wait another 50 years for a new coherent system to evolve. The new paradigms are needed now—therefore, co-design is the key.

2.5.5 Governing principles applied while developing HPX

As it turns out, we do not have to start from scratch. Not everything has to be invented and designed anew. Many of the ideas needed to combat the 4 horsemen already exist, many for more than 30 years. All it takes is to gather them into a coherent approach. We'll highlight some of the derived principles we think to be crucial for defeating **SLOW**. Some of those are focused on high-performance computing, others are more general.

Focus on latency hiding instead of latency avoidance

It is impossible to design a system exposing zero latencies. In an effort to come as close as possible to this goal many optimizations are mainly targeted towards minimizing latencies. Examples for this can be seen everywhere, such as low latency network technologies like [InfiniBand](#)¹⁶⁰, caching memory hierarchies in all modern processors, the constant optimization of existing [MPI](#)¹⁶¹ implementations to reduce related latencies, or the data transfer latencies intrinsic to the way we use [GPGPUs](#)¹⁶² today. It is important to note that existing latencies are often tightly related to some resource having to wait for the operation to be completed. At the same time it would be perfectly fine to do some other, unrelated work in the meantime, allowing the system to hide the latencies by filling the idle-time with

¹⁵⁷ <http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf>

¹⁵⁸ http://en.wikipedia.org/wiki/Harvard_architecture

¹⁵⁹ http://en.wikipedia.org/wiki/Mean_time_between_failures

¹⁶⁰ <http://en.wikipedia.org/wiki/InfiniBand>

¹⁶¹ https://en.wikipedia.org/wiki/Message_Passing_Interface

¹⁶² <http://en.wikipedia.org/wiki/GPGPU>

useful work. Modern systems already employ similar techniques (pipelined instruction execution in the processor cores, asynchronous input/output operations, and many more). What we propose is to go beyond anything we know today and to make latency hiding an intrinsic concept of the operation of the whole system stack.

Embrace fine-grained parallelism instead of heavyweight threads

If we plan to hide latencies even for very short operations, such as fetching the contents of a memory cell from main memory (if it is not already cached), we need to have very lightweight threads with extremely short context switching times, optimally executable within one cycle. Granted, for mainstream architectures, this is not possible today (even if we already have special machines supporting this mode of operation, such as the Cray XMT¹⁶³). For conventional systems, however, the smaller the overhead of a context switch and the finer the granularity of the threading system, the better will be the overall system utilization and its efficiency. For today's architectures we already see a flurry of libraries providing exactly this type of functionality: non-pre-emptive, task-queue based parallelization solutions, such as Intel Threading Building Blocks (TBB)¹⁶⁴, Microsoft Parallel Patterns Library (PPL)¹⁶⁵, Cilk++¹⁶⁶, and many others. The possibility to suspend a current task if some preconditions for its execution are not met (such as waiting for I/O or the result of a different task), seamlessly switching to any other task which can continue, and to reschedule the initial task after the required result has been calculated, which makes the implementation of latency hiding almost trivial.

Rediscover constraint-based synchronization to replace global barriers

The code we write today is riddled with implicit (and explicit) global barriers. By “global barriers,” we mean the synchronization of the control flow between several (very often all) threads (when using OpenMP¹⁶⁷) or processes (MPI¹⁶⁸). For instance, an implicit global barrier is inserted after each loop parallelized using OpenMP¹⁶⁹ as the system synchronizes the threads used to execute the different iterations in parallel. In MPI¹⁷⁰ each of the communication steps imposes an explicit barrier onto the execution flow as (often all) nodes have to be synchronized. Each of those barriers is like the eye of a needle the overall execution is forced to be squeezed through. Even minimal fluctuations in the execution times of the parallel threads (jobs) causes them to wait. Additionally, it is often only one of the executing threads that performs the actual reduce operation, which further impedes parallelism. A closer analysis of a couple of key algorithms used in science applications reveals that these global barriers are not always necessary. In many cases it is sufficient to synchronize a small subset of the threads. Any operation should proceed whenever the preconditions for its execution are met, and only those. Usually there is no need to wait for iterations of a loop to finish before you can continue calculating other things; all you need is to complete the iterations that produce the required results for the next operation. Good bye global barriers, hello constraint based synchronization! People have been trying to build this type of computing (and even computers) since the 1970s. The theory behind what they did is based on ideas around static and dynamic dataflow. There are certain attempts today to get back to those ideas and to incorporate them with modern architectures. For instance, a lot of work is being done in the area of constructing dataflow-oriented execution trees. Our results show that employing dataflow techniques in combination with the other ideas, as outlined herein, considerably improves scalability for many problems.

¹⁶³ http://en.wikipedia.org/wiki/Cray_XMT

¹⁶⁴ <https://www.threadingbuildingblocks.org/>

¹⁶⁵ <https://msdn.microsoft.com/en-us/library/dd492418.aspx>

¹⁶⁶ <https://software.intel.com/en-us/articles/intel-cilk-plus/>

¹⁶⁷ <https://openmp.org/wp/>

¹⁶⁸ https://en.wikipedia.org/wiki/Message_Passing_Interface

¹⁶⁹ <https://openmp.org/wp/>

¹⁷⁰ https://en.wikipedia.org/wiki/Message_Passing_Interface

Adaptive locality control instead of static data distribution

While this principle seems to be a given for single desktop or laptop computers (the operating system is your friend), it is everything but ubiquitous on modern supercomputers, which are usually built from a large number of separate nodes (i.e., Beowulf clusters), tightly interconnected by a high-bandwidth, low-latency network. Today's prevalent programming model for those is MPI, which does not directly help with proper data distribution, leaving it to the programmer to decompose the data to all of the nodes the application is running on. There are a couple of specialized languages and programming environments based on PGAS^{[171](#)} (Partitioned Global Address Space) designed to overcome this limitation, such as Chapel^{[172](#)}, X10^{[173](#)}, UPC^{[174](#)}, or Fortress^{[175](#)}. However, all systems based on PGAS rely on static data distribution. This works fine as long as this static data distribution does not result in heterogeneous workload distributions or other resource utilization imbalances. In a distributed system these imbalances can be mitigated by migrating part of the application data to different localities (nodes). The only framework supporting (limited) migration today is Charm++^{[176](#)}. The first attempts towards solving related problem go back decades as well, a good example is the Linda coordination language^{[177](#)}. Nevertheless, none of the other mentioned systems support data migration today, which forces the users to either rely on static data distribution and live with the related performance hits or to implement everything themselves, which is very tedious and difficult. We believe that the only viable way to flexibly support dynamic and adaptive *locality* control is to provide a global, uniform address space to the applications, even on distributed systems.

Prefer moving work to the data over moving data to the work

For the best performance it seems obvious to minimize the amount of bytes transferred from one part of the system to another. This is true on all levels. At the lowest level we try to take advantage of processor memory caches, thus, minimizing memory latencies. Similarly, we try to amortize the data transfer time to and from GPGPUs^{[178](#)} as much as possible. At high levels we try to minimize data transfer between different nodes of a cluster or between different virtual machines on the cloud. Our experience (well, it's almost common wisdom) shows that the amount of bytes necessary to encode a certain operation is very often much smaller than the amount of bytes encoding the data the operation is performed upon. Nevertheless, we still often transfer the data to a particular place where we execute the operation just to bring the data back to where it came from afterwards. As an example let's look at the way we usually write our applications for clusters using MPI. This programming model is all about data transfer between nodes. MPI is the prevalent programming model for clusters, and it is fairly straightforward to understand and to use. Therefore, we often write applications in a way that accommodates this model, centered around data transfer. These applications usually work well for smaller problem sizes and for regular data structures. The larger the amount of data we have to churn and the more irregular the problem domain becomes, the worse the overall machine utilization and the (strong) scaling characteristics become. While it is not impossible to implement more dynamic, data driven, and asynchronous applications using MPI, it is somewhat difficult to do so. At the same time, if we look at applications that prefer to execute the code close to the *locality* where the data was placed, i.e., utilizing active messages (for instance based on Charm++^{[179](#)}), we see better asynchrony, simpler application codes, and improved scaling.

¹⁷¹ <https://www.pgas.org/>

¹⁷² <https://chapel.cray.com/>

¹⁷³ <https://x10-lang.org/>

¹⁷⁴ <https://upc.lbl.gov/>

¹⁷⁵ <https://labs.oracle.com/projects/plrg/Publications/index.html>

¹⁷⁶ <https://charm.cs.uiuc.edu/>

¹⁷⁷ [http://en.wikipedia.org/wiki/Linda_\(coordination_language\)](http://en.wikipedia.org/wiki/Linda_(coordination_language))

¹⁷⁸ <http://en.wikipedia.org/wiki/GPGPU>

¹⁷⁹ <https://charm.cs.uiuc.edu/>

Favor message driven computation over message passing

Today's prevalently used programming model on parallel (multi-node) systems is MPI. It is based on message passing, as the name implies, which means that the receiver has to be aware of a message about to come in. Both codes, the sender and the receiver, have to synchronize in order to perform the communication step. Even the newer, asynchronous interfaces require explicitly coding the algorithms around the required communication scheme. As a result, everything but the most trivial MPI applications spends a considerable amount of time waiting for incoming messages, thus, causing starvation and latencies to impede full resource utilization. The more complex and more dynamic the data structures and algorithms become, the larger the adverse effects. The community discovered message-driven and data-driven methods of implementing algorithms a long time ago, and systems such as [Charm++¹⁸⁰](#) have already integrated active messages demonstrating the validity of the concept. Message-driven computation allows for sending messages without requiring the receiver to actively wait for them. Any incoming message is handled asynchronously and triggers the encoded action by passing along arguments and—possibly—continuations. *HPX* combines this scheme with work-queue based scheduling as described above, which allows the system to almost completely overlap any communication with useful work, thereby minimizing latencies.

2.6 Additional material

- 2-day workshop held at CSCS in 2016
 - Recorded lectures¹⁸¹
 - Slides¹⁸²
- Tutorials repository¹⁸³
- STELLAR Group blog posts¹⁸⁴
- Basic *HPX* recipes
 - Exporting a free function from a shared library which lives in a namespace, to use as Action¹⁸⁵
 - Turning a struct or class into a component and use it's methods¹⁸⁶
 - Creating and referencing components in hpx¹⁸⁷

2.7 Overview

HPX is organized into different sub-libraries and those in turn into modules. The libraries and modules are independent, with clear dependencies and no cycles. As an end-user, the use of these libraries is completely transparent. If you use e.g. `add_hpx_executable` to create a target in your project you will automatically get all modules as dependencies. See below for a list of the available libraries and modules. Currently these are nothing more than an internal grouping and do not affect usage. They cannot be consumed individually at the moment.

Note: There is a dependency report that displays useful information about the structure of the code. It is available for each commit at [HPX Dependency report](#).

¹⁸⁰ <https://charm.cs.uiuc.edu/>

¹⁸¹ <https://www.youtube.com/playlist?list=PL1tk5lGm7zvSXfS-sqOOmIJ0lFNjKze18>

¹⁸² <https://github.com/STELLAR-GROUP/tutorials/tree/master/cscs2016>

¹⁸³ <https://github.com/STELLAR-GROUP/tutorials>

¹⁸⁴ <http://stellar-group.org/blog/>

¹⁸⁵ <https://gitlab.com/-/snippets/1821389>

¹⁸⁶ <https://gitlab.com/-/snippets/1822983>

¹⁸⁷ <https://gitlab.com/-/snippets/1828131>

2.7.1 Core modules

affinity

The affinity module contains helper functionality for mapping worker threads to hardware resources.

See the API reference of the module for more details.

algorithms

The algorithms module exposes the full set of algorithms defined by the C++ standard. There is also partial support for C++ ranges.

See the [API reference](#) of the module for more details.

allocator_support

This module provides utilities for allocators. It contains `hpx::util::internal_allocator` which directly forwards allocation calls to `jemalloc`. This utility is mainly useful on Windows.

See the API reference of the module for more details.

asio

The asio module is a thin wrapper around the Boost.ASIO library, providing a few additional helper functions.

See the [API reference](#) of the module for more details.

assertion

The assertion library implements the macros `HPX_ASSERT` and `HPX_ASSERT_MSG`. Those two macros can be used to implement assertions which are turned off during a release build.

By default, the location and function where the assert has been called from are displayed when the assertion fires. This behavior can be modified by using `hpx::assertion::set_assertion_handler`. When HPX initializes, it uses this function to specify a more elaborate assertion handler. If your application needs to customize this, it needs to do so before calling `hpx::hpx_init`, `hpx::hpx_main` or using the C-main wrappers.

See the [API reference](#) of the module for more details.

async_base

The `async_base` module defines the basic functionality for spawning tasks on thread pools. This module does not implement any functionality on its own, but is extended by `async_local` and `modules_async_distributed` with implementations for the local and distributed cases.

See the [API reference](#) of this module for more details.

async_combinators

This module contains combinators for futures. The `when_*` functions allow you to turn multiple futures into a single future which is ready when all, any, some, or each of the given futures are ready. The `wait_*` combinators are equivalent to the `when_*` functions except that they do not return a future. Those wait for all futures to become ready before returning to the user. Note that the `wait_*` functions will rethrow one of the exceptions from exceptional futures. The `wait_*_nothrow` combinators are equivalent to the `wait_*` functions exception that they do not throw if one of the futures has become exceptional.

The `split_future` combinator takes a single future of a container (e.g. `tuple`) and turns it into a container of futures.

See [lcos_local](#), [synchronization](#), and [async](#) for other synchronization facilities.

See the [API reference](#) of this module for more details.

async_cuda

This library adds a simple API that enables the user to retrieve a future from a cuda stream. Typically, a user may launch one or more kernels and then get a future from the stream that will become ready when those kernels have completed. It is important to note that multiple kernels may be launched without fetching a future, and multiple futures may be obtained from the helper. Please refer to the unit tests and examples for further examples.

See the [API reference](#) of this module for more details.

async_local

This module extends [async_base](#) to provide local implementations of `hpx::async`, `hpx::apply`, `hpx::sync`, and `hpx::dataflow`.

See the API reference of this module for more details.

async_mpi

The MPI library is intended to simplify the process of integrating MPI based codes with the *HPX* runtime. Any MPI function that is asynchronous and uses an `MPI_Request` may be converted into an `hpx::future`. The syntax is designed to allow a simple replacement of the MPI call with a futurized `async` version that accepts an executor instead of a communicator, and returns a future instead of assigning a request. Typically, an MPI call of the form

```
int MPI_Isend(buf, count, datatype, rank, tag, comm, request);
```

becomes

```
hpx::future<int> f = hpx::async(executor, MPI_Isend, buf, count, datatype, rank, tag);
```

When the MPI operation is complete, the future will become ready. This allows communication to integrated cleanly with the rest of HPX, in particular the continuation style of programming may be used to build up more complex code. Consider the following example, that chains user processing, sends and receives using continuations...

```
// create an executor for MPI dispatch
hpx::mpi::experimental::executor exec(MPI_COMM_WORLD);

// post an asynchronous receive using MPI_Irecv
hpx::future<int> f_recv = hpx::async(
    exec, MPI_Irecv, &data, rank, MPI_INT, rank_from, i);
```

(continues on next page)

(continued from previous page)

```
// attach a continuation to run when the recv completes,
f_recv.then([=, &tokens, &counter] (auto&&)
{
    // call an application specific function
    msg_recv(rank, size, rank_to, rank_from, tokens[i], i);

    // send a new message
    hpx::future<int> f_send = hpx::async(
        exec, MPI_Isend, &tokens[i], 1, MPI_INT, rank_to, i);

    // when that send completes
    f_send.then([=, &tokens, &counter] (auto&&)
    {
        // call an application specific function
        msg_send(rank, size, rank_to, rank_from, tokens[i], i);
    });
}
```

The example above makes use of `MPI_Isend` and `MPI_Irecv`, but *any* MPI function that uses requests may be futurized in this manner. The following is a (non exhaustive) list of MPI functions that *should* be supported, though not all have been tested at the time of writing (please report any problems to the issue tracker).

```
int MPI_Isend(...);
int MPI_Ibsend(...);
int MPI_Issend(...);
int MPI_Irsend(...);
int MPI_Irecv(...);
int MPI_Imrecv(...);
int MPI_Ibarrier(...);
int MPI_Ibcast(...);
int MPI_Igather(...);
int MPI_Igatherv(...);
int MPI_Iscatter(...);
int MPI_Iscatterv(...);
int MPI_Iallgather(...);
int MPI_Iallgatherv(...);
int MPI_Ialltoall(...);
int MPI_Ialltoallv(...);
int MPI_Ialltoallw(...);
int MPI_Ireduce(...);
int MPI_Iallreduce(...);
int MPI_Ireduce_scatter(...);
int MPI_Ireduce_scatter_block(...);
int MPI_Iscan(...);
int MPI_Iexscan(...);
int MPI_Ineighbor_allgather(...);
int MPI_Ineighbor_allgatherv(...);
int MPI_Ineighbor_alltoall(...);
int MPI_Ineighbor_alltoallv(...);
int MPI_Ineighbor_alltoallw(...);
```

Note that the `HPX mpi` futurization wrapper should work with *any* asynchronous `MPI` call, as long as the function signature has the last two arguments `MPI_xxx(..., MPI_Comm comm, MPI_Request *request)` - internally these two parameters will be substituted by the executor and future data parameters that are supplied by template instantiations inside the `hpx::mpi` code.

See the API reference of this module for more details.

batch_environments

This module allows for the detection of execution as batch jobs, a series of programs executed without user intervention. All data is preselected and will be executed according to preset parameters, such as date or completion of another task. Batch environments are especially useful for executing repetitive tasks.

HPX supports the creation of batch jobs through the Portable Batch System (PBS) and SLURM.

For more information on batch environments, see [Running on batch systems](#) and the API reference for the module.

cache

This module provides two cache data structures:

- `hpx::util::cache::local_cache`
- `hpx::util::cache::lru_cache`

See the [API reference](#) of the module for more details.

command_line_handling_local

TODO: High-level description of the module.

See the API reference of this module for more details.

concepts

This module provides helpers for emulating concepts. It provides the following macros:

- `HPX_CONCEPT_REQUIRES`
- `HPX_HAS_MEMBER_XXX_TRAIT_DEF`
- `HPX_HAS_XXX_TRAIT_DEF`

See the API reference of the module for more details.

concurrency

This module provides concurrency primitives useful for multi-threaded programming such as:

- `hpx::util::barrier`
- `hpx::util::cache_line_data` and `hpx::util::cache_aligned_data`: wrappers for aligning and padding data to cache lines.
- various lockfree queue data structures

See the API reference of the module for more details.

config

The config module contains various configuration options, typically hidden behind macros that choose the correct implementation based on the compiler and other available options.

See the [API reference](#) of the module for more details.

config_registry

The config_registry module is a low level module providing helper functionality for registering configuration entries to a global registry from other modules. The `hpx::config_registry::add_module_config` function is used to add configuration options, and `hpx::config_registry::get_module_configs` can be used to retrieve configuration entries registered so far. `add_module_config_helper` can be used to register configuration entries through static global options.

See the API reference of this module for more details.

coroutines

The coroutines module provides coroutine (user-space thread) implementations for different platforms.

See the [API reference](#) of the module for more details.

datastructures

The datastructures module provides basic data structures (typically provided for compatibility with older C++ standards):

- `hpx::detail::small_vector`
- `hpx::util::basic_any`
- `hpx::util::member_pack`
- `hpx::optional`
- `hpx::util::tuple`
- `hpx::variant`

See the API reference of the module for more details.

debugging

This module provides helpers for demangling symbol names.

See the [API reference](#) of the module for more details.

errors

This module provides support for exceptions and error codes:

- `hpx::exception`
- `hpx::error_code`
- `hpx::error`

See the [API reference](#) of the module for more details.

execution

This library implements executors and execution policies for use with parallel algorithms and other facilities related to managing the execution of tasks.

See the [API reference](#) of the module for more details.

execution_base

The basic execution module is the main entry point to implement parallel and concurrent operations. It is modeled after P0443¹⁸⁸ with some additions and implementations for the described concepts. Most notably, it provides an abstraction for execution resources, execution contexts and execution agents in such a way, that it provides customization points that those aforementioned concepts can be replaced and combined with ease.

For that purpose, three virtual base classes are provided to be able to provide implementations with different properties:

- **resource_base**: This is the abstraction for execution resources, that is for example CPU cores or an accelerator.
- **context_base**: An execution context uses execution resources and is able to spawn new execution agents, as new threads of executions on the available resources.
- **agent_base**: The execution agent represents the thread of execution, and can be used to yield, suspend, resume or abort a thread of execution.

executors

The executors module exposes executors and execution policies. Most importantly, it exposes the following classes and constants:

- `hpx::execution::sequenced_executor`
- `hpx::execution::parallel_executor`
- `hpx::execution::sequenced_policy`
- `hpx::execution::parallel_policy`
- `hpx::execution::parallel_unsequenced_policy`
- `hpx::execution::sequenced_task_policy`
- `hpx::execution::parallel_task_policy`
- `hpx::execution::seq`
- `hpx::execution::par`

¹⁸⁸ <http://wg21.link/p0443>

- `hpx::execution::par_unseq`
- `hpx::execution::task`

See the [API reference](#) of this module for more details.

filesystem

This module provides a compatibility layer for the C++17 filesystem library. If the filesystem library is available this module will simply forward its contents into the `hpx::filesystem` namespace. If the library is not available it will fall back to Boost.Filesystem instead.

See the [API reference](#) of the module for more details.

format

The format module exposes the `format` and `format_to` functions for formatting strings.

See the API reference of the module for more details.

functional

This module provides function wrappers and helpers for managing functions and their arguments.

- `hpx::function`
- `hpx::function_ref_`
- `hpx::move_only_function`
- `hpx::bind`
- `hpx::bind_back`
- `hpx::bind_front`
- `hpx::util::deferred_call`
- `hpx::util::invoke`
- `hpx::util::invoke_fused`
- `hpx::util::mem_fn`
- `hpx::util::one_shot`
- `hpx::util::protect`
- `hpx::util::result_of`
- `hpx::placeholders::_1`
- `hpx::placeholders::_2`
- `...`
- `hpx::placeholders::_9`

See the [API reference](#) of the module for more details.

futures

This module defines the `hpx::future` and `hpx::shared_future` classes corresponding to the C++ standard library classes `std::future` and `std::shared_future`. Note that the specializations of `hpx::future::then` for executors and execution policies are defined in the [execution](#) module.

See the [API reference](#) of this module for more details.

hardware

The hardware module abstracts away hardware specific details of timestamps and CPU features.

See the API reference of the module for more details.

hashing

The hashing module provides two hashing implementations:

- `hpx::util::fibhash`
- `hpx::util::jenkins_hash`

See the API reference of the module for more details.

include_local

This module provides no functionality in itself. Instead it provides headers that group together other headers that often appear together. This module provides local-only headers.

See the API reference of this module for more details.

ini

TODO: High-level description of the module.

See the API reference of this module for more details.

init_runtime_local

TODO: High-level description of the module.

See the API reference of this module for more details.

io_service

This module provides an abstraction over Boost.ASIO, combining multiple `asio::io_contexts` into a single pool. `hpx::util::io_service_pool` provides a simple pool of `asio::io_contexts` with an API similar to `asio::io_context`. `hpx::threads::detail::io_service_thread_pool` wraps `hpx::util::io_service_pool` into an interface derived from `hpx::threads::detail::thread_pool_base`.

See the [API reference](#) of this module for more details.

iterator_support

This module provides helpers for iterators. It provides `hpx::util::iterator_facade` and `hpx::util::iterator_adaptor` for creating new iterators, and the trait `hpx::util::is_iterator` along with more specific iterator traits.

See the API reference of the module for more details.

itt_notify

This module provides support for profiling with Intel VTune¹⁸⁹.

See the API reference of this module for more details.

lcos_local

This module provides the following local *LCOs*:

- `hpx::lcos::local::and_gate`
- `hpx::lcos::local::channel`
- `hpx::lcos::local::one_element_channel`
- `hpx::lcos::local::receive_channel`
- `hpx::lcos::local::send_channel`
- `hpx::lcos::local::guard`
- `hpx::lcos::local::guard_set`
- `hpx::lcos::local::run_guarded`
- `hpx::lcos::local::conditional_trigger`
- `hpx::packaged_task`
- `hpx::promise`
- `hpx::lcos::local::receive_buffer`
- `hpx::lcos::local::trigger`

See `lcos_distributed` for distributed LCOs. Basic synchronization primitives for use in HPX threads can be found in `synchronization`. `async_combinators` contains useful utility functions for combining futures.

See the *API reference* of this module for more details.

lock_registration

This module contains functionality for registering locks to detect when they are locked and unlocked on different threads.

See the API reference of this module for more details.

¹⁸⁹ <https://software.intel.com/content/www/us/en/develop/tools/vtune-profiler.html>

logging

This module provides useful macros for logging information.

See the API reference of the module for more details.

memory

Part of this module is a forked version of boost::intrusive_ptr from Boost.SmartPtr.

See the API reference of the module for more details.

mpi_base

This module provides helper functionality for detecting MPI environments.

See the API reference of this module for more details.

pack_traversal

This module exposes the basic functionality for traversing various packs, both synchronously and asynchronously: `hpx::util::traverse_pack` and `hpx::util::traverse_pack_async`. It also exposes the higher level functionality of unwrapping nested futures: `hpx::util::unwrap` and its function object form `hpx::util::functional::unwrap`.

See the [API reference](#) of this module for more details.

plugin

This module provides base utilities for creating plugins.

See the API reference of the module for more details.

prefix

This module provides utilities for handling the prefix of an *HPX* application, i.e. the paths used for searching components and plugins.

See the API reference of this module for more details.

preprocessor

This library contains useful preprocessor macros:

- `HPX_PP_CAT`
- `HPX_PP_EXPAND`
- `HPX_PP_NARGS`
- `HPX_PP_STRINGIZE`
- `HPX_PP_STRIP_PARENS`

See the [API reference](#) of the module for more details.

program_options

The module `program_options` is a direct fork of the Boost.ProgramOptions library (Boost V1.70.0). For more information about this library please see [here¹⁹⁰](#). In order to be included as an *HPX* module, the Boost.ProgramOptions library has been moved to the namespace `hpx::program_options`. We have also replaced all Boost facilities the library depends on with either the equivalent facilities from the standard library or from *HPX*. As a result, the *HPX* `program_options` module is fully interface compatible with Boost.ProgramOptions (sans the `hpx` namespace and the `#include <hpx/modules/program_options.hpp>` changes that need to be applied to all code relying on this library).

All credit goes to Vladimir Prus, the author of the excellent Boost.ProgramOptions library. All bugs have been introduced by us.

See the API reference of the module for more details.

properties

This module implements the `prefer` customization point for properties in terms of [P2220¹⁹¹](#). This differs from [P1393¹⁹²](#) in that it relies fully on `tag_invoke` overloads and fewer base customization points. Actual properties are defined in modules. All functionality is experimental and can be accessed through the `hpx::experimental` namespace.

See the API reference of this module for more details.

resiliency

In *HPX*, a program failure is a manifestation of a failing task. This module exposes several APIs that allow users to manage failing tasks in a convenient way by either replaying a failed task or by replicating a specific task.

Task replay is analogous to the Checkpoint/Rollback mechanism found in conventional execution models. The key difference being localized fault detection. When the runtime detects an error, it replays the failing task as opposed to completely rolling back the entire program to the previous checkpoint.

Task replication is designed to provide reliability enhancements by replicating a set of tasks and evaluating their results to determine a consensus among them. This technique is most effective in situations where there are few tasks in the critical path of the DAG which leaves the system underutilized or where hardware or software failures may result in an incorrect result instead of an error. However, the drawback of this method is the additional computational cost incurred by repeating a task multiple times.

The following API functions are exposed:

- `hpx::resiliency::experimental::async_replay`: This version of task replay will catch user-defined exceptions and automatically reschedule the task N times before throwing an `hpx::resiliency::experimental::abort_replay_exception` if no task is able to complete execution without an exception.
- `hpx::resiliency::experimental::async_replay_validate`: This version of replay adds an argument to `async replay` which receives a user-provided validation function to test the result of the task against. If the task's output is validated, the result is returned. If the output fails the check or an exception is thrown, the task is replayed until no errors are encountered or the number of specified retries has been exceeded.
- `hpx::resiliency::experimental::async_replicate`: This is the most basic implementation of the task replication. The API returns the first result that runs without detecting any errors.

¹⁹⁰ https://www.boost.org/doc/libs/1_70_0/doc/html/program_options.html

¹⁹¹ <https://wg21.link/p2220>

¹⁹² [http://wg21.link/p1393](https://wg21.link/p1393)

- `hpx::resiliency::experimental::async_replicate_validate`: This API additionally takes a validation function which evaluates the return values produced by the threads. The first task to compute a valid result is returned.
- `hpx::resiliency::experimental::async_replicate_vote`: This API adds a vote function to the basic replicate function. Many hardware or software failures are silent errors which do not interrupt program flow. In order to detect errors of this kind, it is necessary to run the task several times and compare the values returned by every version of the task. In order to determine which return value is “correct”, the API allows the user to provide a custom consensus function to properly form a consensus. This voting function then returns the “correct” answer.
- `hpx::resiliency::experimental::async_replicate_vote_validate`: This combines the features of the previously discussed replicate set. Replicate vote validate allows a user to provide a validation function to filter results. Additionally, as described in replicate vote, the user can provide a “voting function” which returns the consensus formed by the voting logic.
- `hpx::resiliency::experimental::dataflow_replay`: This version of dataflow replay will catch user-defined exceptions and automatically reschedules the task N times before throwing an `hpx::resiliency::experimental::abort_replay_exception` if no task is able to complete execution without an exception. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.
- `hpx::resiliency::experimental::dataflow_replay_validate`: This version of replay adds an argument to dataflow replay which receives a user-provided validation function to test the result of the task against. If the task’s output is validated, the result is returned. If the output fails the check or an exception is thrown, the task is replayed until no errors are encountered or the number of specified retries have been exceeded. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.
- `hpx::resiliency::experimental::dataflow_replicate`: This is the most basic implementation of the task replication. The API returns the first result that runs without detecting any errors. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.
- `hpx::resiliency::experimental::dataflow_replicate_validate`: This API additionally takes a validation function which evaluates the return values produced by the threads. The first task to compute a valid result is returned. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.
- `hpx::resiliency::experimental::dataflow_replicate_vote`: This API adds a vote function to the basic replicate function. Many hardware or software failures are silent errors which do not interrupt program flow. In order to detect errors of this kind, it is necessary to run the task several times and compare the values returned by every version of the task. In order to determine which return value is “correct”, the API allows the user to provide a custom consensus function to properly form a consensus. This voting function then returns the “correct” answer. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.
- `hpx::resiliency::experimental::dataflow_replicate_vote_validate`: This combines the features of the previously discussed replicate set. Replicate vote validate allows a user to provide a validation function to filter results. Additionally, as described in replicate vote, the user can provide a “voting function” which returns the consensus formed by the voting logic. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

See the [API reference](#) of the module for more details.

resource_partitioner

The `resource_partitioner` module defines `hpx::resource::partitioner`, the class used by the runtime and users to partition available hardware resources into thread pools. See [Using the resource partitioner](#) for more details on using the resource partitioner in applications.

See the API reference of this module for more details.

runtime_configuration

This module handles the configuration options required by the runtime.

See the [API reference](#) of this module for more details.

runtime_local

TODO: High-level description of the library.

See the [API reference](#) of this module for more details.

schedulers

This module provides schedulers used by thread pools in the `thread_pools` module. There are currently three main schedulers:

- `hpx::threads::policies::local_priority_queue_scheduler`
- `hpx::threads::policies::static_priority_queue_scheduler`
- `hpx::threads::policies::shared_priority_queue_scheduler`

Other schedulers are specializations or variations of the above schedulers. See the examples of the `resource_partitioner` module for examples of specifying a custom scheduler for a thread pool.

See the API reference of this module for more details.

serialization

This module provides serialization primitives and support for all built-in types as well as all C++ Standard Library collection and utility types. This list is extended by *HPX* vocabulary types with proper support for global reference counting. *HPX*'s mode of serialization is derived from [Boost's serialization model](#)¹⁹³ and, as such, is mostly interface compatible with its Boost counterpart.

The purest form of serializing data is to copy the content of the payload bit by bit; however, this method is impractical for generic C++ types, which might be composed of more than just regular built-in types. Instead, *HPX*'s approach to serialization is derived from the Boost Serialization library, and is geared towards allowing the programmer of a given class explicit control and syntax of what to serialize. It is based on operator overloading of two special archive types that hold a buffer or stream to store the serialized data and is responsible for dispatching the serialization mechanism to the intrusive or non-intrusive version. The serialization process is recursive. Each member that needs to be serialized must be specified explicitly. The advantage of this approach is that the serialization code is written in C++ and leverages all necessary programming techniques. The generic, user-facing interface allows for effective application of the serialization process without obstructing the algorithms that need special code for packing and unpacking. It also allows for optimizations in the implementation of the archives.

See the [API reference](#) of the module for more details.

¹⁹³ https://www.boost.org/doc/libs/1_72_0/libs/serialization/doc/index.html

static_reinit

This module provides a simple wrapper around static variables that can be reinitialized.

See the [API reference](#) of this module for more details.

string_util

This module contains string utilities inspired by the Boost string algorithms library.

See the API reference of this module for more details.

synchronization

This module provides synchronization primitives that should be used rather than the C++ standard ones in HPX threads:

- `hpx::barrier`
- `hpx::binary_semaphore`
- `hpx::call_once`
- `hpx::condition_variable`
- `hpx::condition_variable_any`
- `hpx::counting_semaphore`
- `hpx::lcos::local::event`
- `hpx::latch`
- `hpx::mutex`
- `hpx::no_mutex`
- `hpx::once_flag`
- `hpx::recursive_mutex`
- `hpx::shared_mutex`
- `hpx::sliding_semaphore`
- `hpx::spinlock` (`std::mutex` compatible spinlock)
- `hpx::spinlock_no_backoff` (`boost::mutex` compatible spinlock)
- `hpx::spinlock_pool`
- `hpx::stop_callback`
- `hpx::stop_source`
- `hpx::stop_token`
- `hpx::in_place_stop_token`
- `hpx::timed_mutex`
- `hpx::upgrade_to_unique_lock`
- `hpx::upgrade_lock`

See `lcos_local`, `async_combinators`, and `async` for higher level synchronization facilities.

See the [API reference](#) of this module for more details.

tag_invoke

TODO: High-level description of the module.

See the API reference of this module for more details.

testing

The testing module contains useful macros for testing. The results of tests can be printed with `hpx::util::report_errors`. The following macros are provided:

- `HPX_TEST`
- `HPX_TEST_MSG`
- `HPX_TEST_EQ`
- `HPX_TEST_NEQ`
- `HPX_TEST_LT`
- `HPX_TEST_LTE`
- `HPX_TEST_RANGE`
- `HPX_TEST_EQ_MSG`
- `HPX_TEST_NEQ_MSG`
- `HPX_SANITY`
- `HPX_SANITY_MSG`
- `HPX_SANITY_EQ`
- `HPX_SANITY_NEQ`
- `HPX_SANITY_LT`
- `HPX_SANITY_LTE`
- `HPX_SANITY_RANGE`
- `HPX_SANITY_EQ_MSG`

See the API reference of the module for more details.

thread_pool_util

This module contains helper functions for asynchronously suspending and resuming thread pools and their worker threads.

See the [API reference](#) of this module for more details.

thread_pools

This module defines the thread pools and utilities used by the *HPX* runtime. The only thread pool implementation provided by this module is `hpx::threads::detail::scheduled_thread_pool`, which is derived from `hpx::threads::detail::thread_pool_base` defined in the *threading_base* module.

See the API reference of this module for more details.

thread_support

This module provides miscellaneous utilities for threading and concurrency.

See the API reference of the module for more details.

threading

This module provides the equivalents of `std::thread` and `std::jthread` for lightweight *HPX* threads:

- `hpx::thread`
- `hpx::jthread`

See the API reference of this module for more details.

threading_base

This module contains the base class definition required for threads. The base class `hpx::threads::thread_data` is inherited by two specializations for stackful and stackless threads: `hpx::threads::thread_data_stackful` and `hpx::threads::thread_data_stackless`. In addition, the module defines the base classes for schedulers and thread pools: `hpx::threads::policies::scheduler_base` and `hpx::threads::thread_pool_base`.

See the API reference of this module for more details.

thread_manager

This module defines the `hpx::threads::threadmanager` class. This is used by the runtime to manage the creation and destruction of thread pools. The *resource_partitioner* module handles the partitioning of resources into thread pools, but not the creation of thread pools.

See the API reference of this module for more details.

timed_execution

This module provides extensions to the executor interfaces defined in the *execution* module that allow timed submission of tasks on thread pools (at or after a specified time).

See the *API reference* of this module for more details.

timing

This module provides the timing utilities (clocks and timers).

See the API reference of the module for more details.

topology

This module provides the class `hpx::threads::topology` which represents the hardware resources available on a node. The class is a light wrapper around the Portable Hardware Locality (HWLOC)¹⁹⁴ library. The `hpx::threads::cpu_mask` is a small companion class that represents a set of resources on a node.

See the *API reference* of the module for more details.

type_support

This module provides helper facilities related to types.

See the API reference of the module for more details.

util

The util module provides miscellaneous standalone utilities.

See the *API reference* of the module for more details.

version

This module macros and functions for accessing version information about *HPX* and its dependencies.

See the *API reference* of this module for more details.

2.7.2 Main HPX modules

actions

TODO: High-level description of the library.

See the *API reference* of this module for more details.

actions_base

TODO: High-level description of the library.

See the *API reference* of this module for more details.

¹⁹⁴ <https://www.open-mpi.org/projects/hwloc/>

agas

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

agas_base

This module holds the implementation of the four AGAS services: primary namespace, locality namespace, component namespace, and symbol namespace.

See the [API reference](#) of this module for more details.

async_colocated

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

async

This module contains functionality for asynchronously launching work on remote localities: `hpx::async`, `hpx::apply`. This module extends the local-only functions in `libs_async_local`.

See the API reference of this module for more details.

checkpoint

A common need of users is to periodically backup an application. This practice provides resiliency and potential restart points in code. *HPX* utilizes the concept of a `checkpoint` to support this use case.

Found in `hpx/util/checkpoint.hpp`, checkpoints are defined as objects that hold a serialized version of an object or set of objects at a particular moment in time. This representation can be stored in memory for later use or it can be written to disk for storage and/or recovery at a later point. In order to create and fill this object with data, users must use a function called `save_checkpoint`. In code the function looks like this:

```
hpx::future<hpx::util::checkpoint> hpx::util::save_checkpoint(a, b, c, ...);
```

`save_checkpoint` takes arbitrary data containers, such as `int`, `double`, `float`, `vector`, and `future`, and serializes them into a newly created `checkpoint` object. This function returns a `future` to a `checkpoint` containing the data. Here's an example of a simple use case:

```
using hpx::util::checkpoint;
using hpx::util::save_checkpoint;

std::vector<int> vec{1, 2, 3, 4, 5};
hpx::future<checkpoint> save_checkpoint(vec);
```

Once the future is ready, the `checkpoint` object will contain the `vector` `vec` and its five elements.

`prepare_checkpoint` takes arbitrary data containers (same as for `save_checkpoint`), , such as `int`, `double`, `float`, `vector`, and `future`, and calculates the necessary buffer space for the `checkpoint` that would be created if `save_checkpoint` was called with the same arguments. This function returns a `future` to a `checkpoint` that is appropriately initialized. Here's an example of a simple use case:

```
using hpx::util::checkpoint;
using hpx::util::prepare_checkpoint;

std::vector<int> vec{1,2,3,4,5};
hpx::future<checkpoint> prepare_checkpoint(vec);
```

Once the future is ready, the checkpoint object will be initialized with an appropriately sized internal buffer.

It is also possible to modify the launch policy used by `save_checkpoint`. This is accomplished by passing a launch policy as the first argument. It is important to note that passing `hpx::launch::sync` will cause `save_checkpoint` to return a `checkpoint` instead of a future to a `checkpoint`. All other policies passed to `save_checkpoint` will return a future to a `checkpoint`.

Sometimes `checkpoint`s must be declared before they are used. `save_checkpoint` allows users to move pre-created `checkpoint`s into the function as long as they are the first container passing into the function (In the case where a launch policy is used, the `checkpoint` will immediately follow the launch policy). An example of these features can be found below:

```
char character = 'd';
int integer = 10;
float flt = 10.01f;
bool boolean = true;
std::string str = "I am a string of characters";
std::vector<char> vec(str.begin(), str.end());
checkpoint archive;

// Test 1
// test basic functionality
hpx::shared_future<checkpoint> f_archive = save_checkpoint(
    std::move(archive), character, integer, flt, boolean, str, vec);
```

Once users can create `checkpoints` they must now be able to restore the objects they contain into memory. This is accomplished by the function `restore_checkpoint`. This function takes a `checkpoint` and fills its data into the containers it is provided. It is important to remember that the containers must be ordered in the same way they were placed into the `checkpoint`. For clarity see the example below:

```
char character2;
int integer2;
float flt2;
bool boolean2;
std::string str2;
std::vector<char> vec2;

restore_checkpoint(data, character2, integer2, flt2, boolean2, str2, vec2);
```

The core utility of `checkpoint` is in its ability to make certain data persistent. Often, this means that the data needs to be stored in an object, such as a file, for later use. *HPX* has two solutions for these issues: stream operator overloads and access iterators.

HPX contains two stream overloads, `operator<<` and `operator>>`, to stream data out of and into `checkpoint`. Here is an example of the overloads in use below:

```
double a9 = 1.0, b9 = 1.1, c9 = 1.2;
std::ofstream test_file_9("test_file_9.txt");
hpx::future<checkpoint> f_9 = save_checkpoint(a9, b9, c9);
test_file_9 << f_9.get();
test_file_9.close();
```

(continues on next page)

(continued from previous page)

```
double a9_1, b9_1, c9_1;
std::ifstream test_file_9_1("test_file_9.txt");
checkpoint archive9;
test_file_9_1 >> archive9;
restore_checkpoint(archive9, a9_1, b9_1, c9_1);
```

This is the primary way to move data into and out of a checkpoint. It is important to note, however, that users should be cautious when using a stream operator to load data and another function to remove it (or vice versa). Both `operator<<` and `operator>>` rely on a `.write()` and a `.read()` function respectively. In order to know how much data to read from the `std::istream`, the `operator<<` will write the size of the `checkpoint` before writing the `checkpoint` data. Correspondingly, the `operator>>` will read the size of the stored data before reading the data into a new instance of `checkpoint`. As long as the user employs the `operator<<` and `operator>>` to stream the data, this detail can be ignored.

Important: Be careful when mixing `operator<<` and `operator>>` with other facilities to read and write to a `checkpoint`. `operator<<` writes an extra variable, and `operator>>` reads this variable back separately. Used together the user will not encounter any issues and can safely ignore this detail.

Users may also move the data into and out of a `checkpoint` using the exposed `.begin()` and `.end()` iterators. An example of this use case is illustrated below.

```
std::ofstream test_file_7("checkpoint_test_file.txt");
std::vector<float> vec7{1.02f, 1.03f, 1.04f, 1.05f};
hpx::future<checkpoint> fut_7 = save_checkpoint(vec7);
checkpoint archive7 = fut_7.get();
std::copy(archive7.begin(),           // Write data to ofstream
          archive7.end(),          // ie. the file
          std::ostream_iterator<char>(test_file_7));
test_file_7.close();

std::vector<float> vec7_1;
std::vector<char> char_vec;
std::ifstream test_file_7_1("checkpoint_test_file.txt");
if (test_file_7_1)
{
    test_file_7_1.seekg(0, test_file_7_1.end);
    auto length = test_file_7_1.tellg();
    test_file_7_1.seekg(0, test_file_7_1.beg);
    char_vec.resize(length);
    test_file_7_1.read(char_vec.data(), length);
}
checkpoint archive7_1(std::move(char_vec));      // Write data to checkpoint
restore_checkpoint(archive7_1, vec7_1);
```

Checkpointing components

`save_checkpoint` and `restore_checkpoint` are also able to store components inside checkpoints. This can be done in one of two ways. First a client of the component can be passed to `save_checkpoint`. When the user wishes to resurrect the component she can pass a client instance to `restore_checkpoint`.

This technique is demonstrated below:

```
// Try to checkpoint and restore a component with a client
std::vector<int> vec3{10, 10, 10, 10, 10};

// Create a component instance through client constructor
data_client D(hpx::find_here(), std::move(vec3));
hpx::future<checkpoint> f3 = save_checkpoint(D);

// Create a new client
data_client E;

// Restore server inside client instance
restore_checkpoint(f3.get(), E);
```

The second way a user can save a component is by passing a `shared_ptr` to the component to `save_checkpoint`. This component can be resurrected by creating a new instance of the component type and passing a `shared_ptr` to the new instance to `restore_checkpoint`.

This technique is demonstrated below:

```
// test checkpoint a component using a shared_ptr
std::vector<int> vec{1, 2, 3, 4, 5};
data_client A(hpx::find_here(), std::move(vec));

// Checkpoint Server
hpx::id_type old_id = A.get_id();

hpx::future<std::shared_ptr<data_server>> f_a_ptr =
    hpx::get_ptr<data_server>(A.get_id());
std::shared_ptr<data_server> a_ptr = f_a_ptr.get();
hpx::future<checkpoint> f = save_checkpoint(a_ptr);
auto&& data = f.get();

// test prepare_checkpoint API
checkpoint c = prepare_checkpoint(hpx::launch::sync, a_ptr);
HPX_TEST(c.size() == data.size());

// Restore Server
// Create a new server instance
std::shared_ptr<data_server> b_server;
restore_checkpoint(data, b_server);
```

checkpoint_base

The checkpoint_base module contains lower level facilities that wrap simple check-pointing capabilities. This module does not implement special handling for futures or components, but simply serializes all arguments to or from a given container.

This module exposes the `hpx::util::save_checkpoint_data`, `hpx::util::restore_checkpoint_data`, and `hpx::util::prepare_checkpoint_data` APIs. These functions encapsulate the basic serialization functionalities necessary to save/restore a variadic list of arguments to/from a given data container.

See the [API reference](#) of this module for more details.

collectives

The collectives module exposes a set of distributed collective operations. Those can be used to exchange data between participating sites in a coordinated way. At this point the module exposes the following collective primitives:

- `hpx::collectives::all_gather`: receives a set of values from all participating sites.
- `hpx::collectives::all_reduce`: performs a reduction on data from each participating site to each participating site.
- `hpx::collectives::all_to_all`: each participating site provides its element of the data to collect while all participating sites receive the data from every other site.
- `hpx::collectives::broadcast_to` and `hpx::collectives::broadcast_from`: performs a broadcast operation from a root site to all participating sites.
- **cpp:func:hpx::collectives::exclusive_scan** performs an exclusive scan operation on a set of values received from all call sites operating on the given base name.
- `hpx::collectives::gather_here` and `hpx::collectives::gather_there`: gathers values from all participating sites.
- **cpp:func:hpx::collectives::inclusive_scan** performs an inclusive scan operation on a set of values received from all call sites operating on the given base name.
- `hpx::collectives::reduce_here` and `hpx::collectives::reduce_there`: performs a reduction on data from each participating site to a root site.
- `hpx::collectives::scatter_to` and `hpx::collectives::scatter_from`: receives an element of a set of values operating on the given base name.
- `hpx::lcos::broadcast`: performs a given action on all given global identifiers.
- `hpx::distributed::barrier`: distributed barrier.
- `hpx::lcos::fold`: performs a fold with a given action on all given global identifiers.
- `hpx::distributed::latch`: distributed latch.
- `hpx::lcos::reduce`: performs a reduction on data from each given global identifiers.
- `hpx::lcos::spmd_block`: performs the same operation on a local image while providing handles to the other images.

See the [API reference](#) of the module for more details.

command_line_handling

The command_line_handling module defines and handles the command-line options required by the *HPX* runtime, combining them with configuration options defined by the *runtime_configuration* module. The actual parsing of command line options is handled by the *program_options* module.

See the API reference of the module for more details.

components

TODO: High-level description of the module.

See the *API reference* of this module for more details.

components_base

TODO: High-level description of the library.

See the *API reference* of this module for more details.

compute

The compute module provides utilities for handling task and memory affinity on host systems.

See the *API reference* of the module for more details.

distribution_policies

TODO: High-level description of the module.

See the *API reference* of this module for more details.

executors_distributed

This module provides the executor `hpx::parallel::execution::distribution_policy_executor`. It allows one to create work that is implicitly distributed over multiple localities.

See the *API reference* of this module for more details.

include

This module provides no functionality in itself. Instead it provides headers that group together other headers that often appear together.

See the API reference of this module for more details.

init_runtime

TODO: High-level description of the library.

See the [API reference](#) of this module for more details.

lcos_distributed

This module contains distributed *LCOs*. Currently the only LCO provided is :cpp:class::*hpx::lcos::channel*, a construct for sending values from one *locality* to another. See `libs_lcos_local` for local LCOs.

See the API reference of this module for more details.

naming

TODO: High-level description of the module.

See the API reference of this module for more details.

naming_base

This module provides a forward declaration of *address_type*, *component_type* and *invalid_locality_id*.

See the [API reference](#) of this module for more details.

parcelport_lci

TODO: High-level description of the module.

See the API reference of this module for more details.

parcelport_libfabric

TODO: High-level description of the module.

See the API reference of this module for more details.

parcelport_mpi

TODO: High-level description of the module.

See the API reference of this module for more details.

parcelport_tcp

TODO: High-level description of the module.

See the API reference of this module for more details.

parcelset

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

parcelset_base

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

performance_counters

This module provides the basic functionality required for defining performance counters. See [Performance counters](#) for more information about performance counters.

See the [API reference](#) of this module for more details.

plugin_factories

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

resiliency_distributed

Software resiliency features of HPX were introduced in the [resiliency module](#). This module extends the APIs to run on distributed-memory systems allowing the user to invoke the failing task on other localities at runtime. This is useful in cases where a node is identified to fail more often (e.g., for certain ALU computes) as the task can now be replayed or replicated among different localities. The API exposed allows for an easy integration with the local only resiliency APIs as well.

Distributed software resilience APIs have a similar function signature and lives under the same namespace of `hpx::resiliency::experimental`. The difference arises in the formal parameters where distributed APIs takes the localities as the first argument, and an action as opposed to a function or a function object. The localities signify the order in which the API will either schedule (in case of Task Replay) tasks in a round robin fashion or replicate the tasks onto the list of localities.

The list of APIs exposed by distributed resiliency modules is the same as those defined in [local resiliency module](#).

See the API reference of this module for more details.

runtime_components

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

runtime_distributed

TODO: High-level description of the module.

See the [API reference](#) of this module for more details.

segmented_algorithms

Segmented algorithms extend the usual parallel [algorithms](#) by providing overloads that work with distributed containers, such as partitioned vectors.

See the [API reference](#) of the module for more details.

statistics

This module provide some statistics utilities like rolling min/max and histogram.

See the API reference of the module for more details.

2.8 API reference

HPX follows a versioning scheme with three numbers: `major.minor.patch`. We guarantee no breaking changes in the API for patch releases. Minor releases may remove or break existing APIs, but only after a deprecation period of at least two minor releases. In rare cases do we outright remove old and unused functionality without a deprecation period.

We do not provide any ABI compatibility guarantees between any versions, debug and release builds, and builds with different C++ standards.

The public API of *HPX* is presented below. Clicking on a name brings you to the full documentation for the class or function. Including the header specified in a heading brings in the features listed under that heading.

Note: Names listed here are guaranteed stable with respect to semantic versioning. However, at the moment the list is incomplete and certain unlisted features are intended to be in the public API. While we work on completing the list, if you're unsure about whether a particular unlisted name is part of the public API you can get into contact with us or open an issue and we'll clarify the situation.

2.8.1 Public API

Our API is semantically conforming; hence, the reader is highly encouraged to refer to the corresponding facility in the [C++ Standard](#)¹⁹⁵ if needed. All names below are also available in the top-level `hpx` namespace unless otherwise noted. The names in `hpx` should be preferred. The names in sub-namespaces will eventually be removed.

¹⁹⁵ <https://en.cppreference.com/w/cpp/header>

hpx/algorithms.hpp

The header `hpx/algorithms.hpp`¹⁹⁶ corresponds to the C++ standard library header `algorithm`¹⁹⁷. See [Using parallel algorithms](#) for more information about the parallel algorithms.

Classes

Table 2.35: Classes of header `hpx/algorithms.hpp`

Class	C++ standard
<code>hpx::experimental::reduction</code>	N4808 ¹⁹⁸
<code>hpx::experimental::induction</code>	N4808 ¹⁹⁹

Functions

Table 2.36: *hpx* functions of header `hpx/algorithms.hpp`

<i>hpx</i> function	C++ standard
<code>hpx::adjacent_find</code>	<code>std::adjacent_find</code> ²⁰⁰
<code>hpx::all_of</code>	<code>std::all_of</code> ²⁰¹
<code>hpx::any_of</code>	<code>std::any_of</code> ²⁰²
<code>hpx::copy</code>	<code>std::copy</code> ²⁰³
<code>hpx::copy_if</code>	<code>std::copy_if</code> ²⁰⁴
<code>hpx::copy_n</code>	<code>std::copy_n</code> ²⁰⁵
<code>hpx::count</code>	<code>std::count</code> ²⁰⁶
<code>hpx::count_if</code>	<code>std::count_if</code> ²⁰⁷
<code>hpx::ends_with</code>	<code>std::ends_with</code> ²⁰⁸
<code>hpx::equal</code>	<code>std::equal</code> ²⁰⁹
<code>hpx::fill</code>	<code>std::fill</code> ²¹⁰
<code>hpx::fill_n</code>	<code>std::fill_n</code> ²¹¹
<code>hpx::find</code>	<code>std::find</code> ²¹²
<code>hpx::find_end</code>	<code>std::find_end</code> ²¹³
<code>hpx::find_first_of</code>	<code>std::find_first_of</code> ²¹⁴
<code>hpx::find_if</code>	<code>std::find_if</code> ²¹⁵
<code>hpx::find_if_not</code>	<code>std::find_if_not</code> ²¹⁶
<code>hpx::for_each</code>	<code>std::for_each</code> ²¹⁷
<code>hpx::for_each_n</code>	<code>std::for_each_n</code> ²¹⁸
<code>hpx::generate</code>	<code>std::generate</code> ²¹⁹
<code>hpx::generate_n</code>	<code>std::generate_n</code> ²²⁰
<code>hpx::includes</code>	<code>std::includes</code> ²²¹
<code>hpx::inplace_merge</code>	<code>std::inplace_merge</code> ²²²
<code>hpx::is_heap</code>	<code>std::is_heap</code> ²²³
<code>hpx::is_heap_until</code>	<code>std::is_heap_until</code> ²²⁴
<code>hpx::is_partitioned</code>	<code>std::is_partitioned</code> ²²⁵
<code>hpx::is_sorted</code>	<code>std::is_sorted</code> ²²⁶

continues on next page

¹⁹⁶ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/algorithms.hpp>

¹⁹⁷ <http://en.cppreference.com/w/cpp/header/algorithm>

¹⁹⁸ <http://wg21.link/n4808>

¹⁹⁹ <http://wg21.link/n4808>

Table 2.36 – continued from previous page

<i>hpx</i> function	C++ standard
<code>hpx::is_sorted_until</code>	<code>std::is_sorted_until</code> ²²⁷
<code>hpx::lexicographical_compare</code>	<code>std::lexicographical_compare</code> ²²⁸
<code>hpx::make_heap</code>	<code>std::make_heap</code> ²²⁹
<code>hpx::max_element</code>	<code>std::max_element</code> ²³⁰
<code>hpx::merge</code>	<code>std::merge</code> ²³¹
<code>hpx::min_element</code>	<code>std::min_element</code> ²³²
<code>hpx::minmax_element</code>	<code>std::minmax_element</code> ²³³
<code>hpx::mismatch</code>	<code>std::mismatch</code> ²³⁴
<code>hpx::move</code>	<code>std::move</code> ²³⁵
<code>hpx::none_of</code>	<code>std::none_of</code> ²³⁶
<code>hpx::nth_element</code>	<code>std::nth_element</code> ²³⁷
<code>hpx::partial_sort</code>	<code>std::partial_sort</code> ²³⁸
<code>hpx::partial_sort_copy</code>	<code>std::partial_sort_copy</code> ²³⁹
<code>hpx::partition</code>	<code>std::partition</code> ²⁴⁰
<code>hpx::partition_copy</code>	<code>std::partition_copy</code> ²⁴¹
<code>hpx::remove</code>	<code>std::remove</code> ²⁴²
<code>hpx::remove_copy</code>	<code>std::remove_copy</code> ²⁴³
<code>hpx::remove_copy_if</code>	<code>std::remove_copy_if</code> ²⁴⁴
<code>hpx::remove_if</code>	<code>std::remove_if</code> ²⁴⁵
<code>hpx::replace</code>	<code>std::replace</code> ²⁴⁶
<code>hpx::replace_copy</code>	<code>std::replace_copy</code> ²⁴⁷
<code>hpx::replace_copy_if</code>	<code>std::replace_copy_if</code> ²⁴⁸
<code>hpx::replace_if</code>	<code>std::replace_if</code> ²⁴⁹
<code>hpx::reverse</code>	<code>std::reverse</code> ²⁵⁰
<code>hpx::reverse_copy</code>	<code>std::reverse_copy</code> ²⁵¹
<code>hpx::rotate</code>	<code>std::rotate</code> ²⁵²
<code>hpx::rotate_copy</code>	<code>std::rotate_copy</code> ²⁵³
<code>hpx::search</code>	<code>std::search</code> ²⁵⁴
<code>hpx::search_n</code>	<code>std::search_n</code> ²⁵⁵
<code>hpx::set_difference</code>	<code>std::set_difference</code> ²⁵⁶
<code>hpx::set_intersection</code>	<code>std::set_intersection</code> ²⁵⁷
<code>hpx::set_symmetric_difference</code>	<code>std::set_symmetric_difference</code> ²⁵⁸
<code>hpx::set_union</code>	<code>std::set_union</code> ²⁵⁹
<code>hpx::shift_left</code>	<code>std::shift_left</code> ²⁶⁰
<code>hpx::shift_right</code>	<code>std::shift_right</code> ²⁶¹
<code>hpx::sort</code>	<code>std::sort</code> ²⁶²
<code>hpx::stable_partition</code>	<code>std::stable_partition</code> ²⁶³
<code>hpx::stable_sort</code>	<code>std::stable_sort</code> ²⁶⁴
<code>hpx::starts_with</code>	<code>std::starts_with</code> ²⁶⁵
<code>hpx::swap_ranges</code>	<code>std::swap_ranges</code> ²⁶⁶
<code>hpx::transform</code>	<code>std::transform</code> ²⁶⁷
<code>hpx::unique</code>	<code>std::unique</code> ²⁶⁸
<code>hpx::unique_copy</code>	<code>std::unique_copy</code> ²⁶⁹
<code>hpx::experimental::for_loop</code>	N4808 ²⁷⁰
<code>hpx::experimental::for_loop_strided</code>	N4808 ²⁷¹
<code>hpx::experimental::for_loop_n</code>	N4808 ²⁷²
<code>hpx::experimental::for_loop_n_strided</code>	N4808 ²⁷³

200 http://en.cppreference.com/w/cpp/algorithm/adjacent_find
 201 http://en.cppreference.com/w/cpp/algorithm/all_any_none_of
 202 http://en.cppreference.com/w/cpp/algorithm/all_any_none_of
 203 <http://en.cppreference.com/w/cpp/algorithm/copy>
 204 <http://en.cppreference.com/w/cpp/algorithm/copy>
 205 http://en.cppreference.com/w/cpp/algorithm/copy_n
 206 <http://en.cppreference.com/w/cpp/algorithm/count>
 207 <http://en.cppreference.com/w/cpp/algorithm/count>
 208 http://en.cppreference.com/w/cpp/algorithm/ranges/ends_with
 209 <http://en.cppreference.com/w/cpp/algorithm/equal>
 210 <http://en.cppreference.com/w/cpp/algorithm/fill>
 211 http://en.cppreference.com/w/cpp/algorithm/fill_n
 212 <http://en.cppreference.com/w/cpp/algorithm/find>
 213 http://en.cppreference.com/w/cpp/algorithm/find_end
 214 http://en.cppreference.com/w/cpp/algorithm/find_first_of
 215 <http://en.cppreference.com/w/cpp/algorithm/find>
 216 <http://en.cppreference.com/w/cpp/algorithm/find>
 217 http://en.cppreference.com/w/cpp/algorithm/for_each
 218 http://en.cppreference.com/w/cpp/algorithm/for_each_n
 219 <http://en.cppreference.com/w/cpp/algorithm/generate>
 220 http://en.cppreference.com/w/cpp/algorithm/generate_n
 221 <http://en.cppreference.com/w/cpp/algorithm/includes>
 222 http://en.cppreference.com/w/cpp/algorithm/inplace_merge
 223 http://en.cppreference.com/w/cpp/algorithm/is_heap
 224 http://en.cppreference.com/w/cpp/algorithm/is_heap_until
 225 http://en.cppreference.com/w/cpp/algorithm/is_partitioned
 226 http://en.cppreference.com/w/cpp/algorithm/is_sorted
 227 http://en.cppreference.com/w/cpp/algorithm/is_sorted_until
 228 http://en.cppreference.com/w/cpp/algorithm/lexicographical_compare
 229 http://en.cppreference.com/w/cpp/algorithm/make_heap
 230 http://en.cppreference.com/w/cpp/algorithm/max_element
 231 <http://en.cppreference.com/w/cpp/algorithm/merge>
 232 http://en.cppreference.com/w/cpp/algorithm/min_element
 233 http://en.cppreference.com/w/cpp/algorithm/minmax_element
 234 <http://en.cppreference.com/w/cpp/algorithm/mismatch>
 235 <http://en.cppreference.com/w/cpp/algorithm/move>
 236 http://en.cppreference.com/w/cpp/algorithm/all_any_none_of
 237 http://en.cppreference.com/w/cpp/algorithm/nth_element
 238 http://en.cppreference.com/w/cpp/algorithm/partial_sort
 239 http://en.cppreference.com/w/cpp/algorithm/partial_sort_copy
 240 <http://en.cppreference.com/w/cpp/algorithm/partition>
 241 http://en.cppreference.com/w/cpp/algorithm/partition_copy
 242 <http://en.cppreference.com/w/cpp/algorithm/remove>
 243 http://en.cppreference.com/w/cpp/algorithm/remove_copy
 244 http://en.cppreference.com/w/cpp/algorithm/remove_copy
 245 <http://en.cppreference.com/w/cpp/algorithm/remove>
 246 <http://en.cppreference.com/w/cpp/algorithm/replace>
 247 http://en.cppreference.com/w/cpp/algorithm/replace_copy
 248 http://en.cppreference.com/w/cpp/algorithm/replace_copy
 249 <http://en.cppreference.com/w/cpp/algorithm/replace>
 250 <http://en.cppreference.com/w/cpp/algorithm/reverse>
 251 http://en.cppreference.com/w/cpp/algorithm/reverse_copy
 252 <http://en.cppreference.com/w/cpp/algorithm/rotate>
 253 http://en.cppreference.com/w/cpp/algorithm/rotate_copy
 254 <http://en.cppreference.com/w/cpp/algorithm/search>
 255 http://en.cppreference.com/w/cpp/algorithm/search_n
 256 http://en.cppreference.com/w/cpp/algorithm/set_difference
 257 http://en.cppreference.com/w/cpp/algorithm/set_intersection
 258 http://en.cppreference.com/w/cpp/algorithm/set_symmetric_difference
 259 http://en.cppreference.com/w/cpp/algorithm/set_union
 260 <http://en.cppreference.com/w/cpp/algorithm/shift>
 261 <http://en.cppreference.com/w/cpp/algorithm/shift>
 262 <http://en.cppreference.com/w/cpp/algorithm/sort>
 263 http://en.cppreference.com/w/cpp/algorithm/stable_partition
 264 http://en.cppreference.com/w/cpp/algorithm/stable_sort
 265 http://en.cppreference.com/w/cpp/algorithm/ranges/starts_with
 266 http://en.cppreference.com/w/cpp/algorithm/swap_ranges
 267 <http://en.cppreference.com/w/cpp/algorithm/transform>
 268 <http://en.cppreference.com/w/cpp/algorithm/unique>
 269 http://en.cppreference.com/w/cpp/algorithm/unique_copy
 270 <http://wg21.link/n4808>
 271 <http://wg21.link/n4808>
 272 <http://wg21.link/n4808>
 273 <http://wg21.link/n4808>

Table 2.37: *hpx::ranges* functions of header *hpx/algorithm.hpp*

<i>hpx::ranges</i> function	C++ standard
<i>hpx::ranges::adjacent_find</i>	<code>std::adjacent_find</code> ²⁷⁴
<i>hpx::ranges::all_of</i>	<code>std::all_of</code> ²⁷⁵
<i>hpx::ranges::any_of</i>	<code>std::any_of</code> ²⁷⁶
<i>hpx::ranges::copy</i>	<code>std::copy</code> ²⁷⁷
<i>hpx::ranges::copy_if</i>	<code>std::copy_if</code> ²⁷⁸
<i>hpx::ranges::copy_n</i>	<code>std::copy_n</code> ²⁷⁹
<i>hpx::ranges::count</i>	<code>std::count</code> ²⁸⁰
<i>hpx::ranges::count_if</i>	<code>std::count_if</code> ²⁸¹
<i>hpx::ranges::ends_with</i>	<code>std::ends_with</code> ²⁸²
<i>hpx::ranges::equal</i>	<code>std::equal</code> ²⁸³
<i>hpx::ranges::fill</i>	<code>std::fill</code> ²⁸⁴
<i>hpx::ranges::fill_n</i>	<code>std::fill_n</code> ²⁸⁵
<i>hpx::ranges::find</i>	<code>std::find</code> ²⁸⁶
<i>hpx::ranges::find_end</i>	<code>std::find_end</code> ²⁸⁷
<i>hpx::ranges::find_first_of</i>	<code>std::find_first_of</code> ²⁸⁸
<i>hpx::ranges::find_if</i>	<code>std::find_if</code> ²⁸⁹
<i>hpx::ranges::find_if_not</i>	<code>std::find_if_not</code> ²⁹⁰
<i>hpx::ranges::for_each</i>	<code>std::for_each</code> ²⁹¹
<i>hpx::ranges::for_each_n</i>	<code>std::for_each_n</code> ²⁹²
<i>hpx::ranges::generate</i>	<code>std::generate</code> ²⁹³
<i>hpx::ranges::generate_n</i>	<code>std::generate_n</code> ²⁹⁴
<i>hpx::ranges::includes</i>	<code>std::includes</code> ²⁹⁵
<i>hpx::ranges::inplace_merge</i>	<code>std::inplace_merge</code> ²⁹⁶
<i>hpx::ranges::is_heap</i>	<code>std::is_heap</code> ²⁹⁷
<i>hpx::ranges::is_heap_until</i>	<code>std::is_heap_until</code> ²⁹⁸
<i>hpx::ranges::is_partitioned</i>	<code>std::is_partitioned</code> ²⁹⁹
<i>hpx::ranges::is_sorted</i>	<code>std::is_sorted</code> ³⁰⁰
<i>hpx::ranges::is_sorted_until</i>	<code>std::is_sorted_until</code> ³⁰¹
<i>hpx::ranges::make_heap</i>	<code>std::make_heap</code> ³⁰²
<i>hpx::ranges::merge</i>	<code>std::merge</code> ³⁰³
<i>hpx::ranges::move</i>	<code>std::move</code> ³⁰⁴
<i>hpx::ranges::none_of</i>	<code>std::none_of</code> ³⁰⁵
<i>hpx::ranges::nth_element</i>	<code>std::nth_element</code> ³⁰⁶
<i>hpx::ranges::partial_sort</i>	<code>std::partial_sort</code> ³⁰⁷
<i>hpx::ranges::partial_sort_copy</i>	<code>std::partial_sort_copy</code> ³⁰⁸
<i>hpx::ranges::partition</i>	<code>std::partition</code> ³⁰⁹
<i>hpx::ranges::partition_copy</i>	<code>std::partition_copy</code> ³¹⁰
<i>hpx::ranges::set_difference</i>	<code>std::set_difference</code> ³¹¹
<i>hpx::ranges::set_intersection</i>	<code>std::set_intersection</code> ³¹²
<i>hpx::ranges::set_symmetric_difference</i>	<code>std::set_symmetric_difference</code> ³¹³
<i>hpx::ranges::set_union</i>	<code>std::set_union</code> ³¹⁴
<i>hpx::ranges::shift_left</i>	P2440 ³¹⁵
<i>hpx::ranges::shift_right</i>	P2440 ³¹⁶
<i>hpx::ranges::sort</i>	<code>std::sort</code> ³¹⁷
<i>hpx::ranges::stable_partition</i>	<code>std::stable_partition</code> ³¹⁸
<i>hpx::ranges::stable_sort</i>	<code>std::stable_sort</code> ³¹⁹
<i>hpx::ranges::starts_with</i>	<code>std::starts_with</code> ³²⁰
<i>hpx::ranges::swap_ranges</i>	<code>std::swap_ranges</code> ³²¹

continues on next page

Table 2.37 – continued from previous page

<i>hpx::ranges</i> function	C++ standard
<i>hpx::ranges::unique</i>	<i>std::unique</i> ³²²
<i>hpx::ranges::unique_copy</i>	<i>std::unique_copy</i> ³²³
<i>hpx::ranges::experimental::for_loop</i>	N4808 ³²⁴
<i>hpx::ranges::experimental::for_loop_str</i>	N4808 ³²⁵

²⁷⁴ http://en.cppreference.com/w/cpp/algorithm/ranges/adjacent_find²⁷⁵ http://en.cppreference.com/w/cpp/algorithm/ranges/all_any_none_of276 http://en.cppreference.com/w/cpp/algorithm/ranges/all_any_none_of277 <http://en.cppreference.com/w/cpp/algorithm/ranges/copy>278 <http://en.cppreference.com/w/cpp/algorithm/ranges/copy>279 http://en.cppreference.com/w/cpp/algorithm/ranges/copy_n280 <http://en.cppreference.com/w/cpp/algorithm/ranges/count>281 <http://en.cppreference.com/w/cpp/algorithm/ranges/count>282 http://en.cppreference.com/w/cpp/algorithm/ranges/ends_with283 <http://en.cppreference.com/w/cpp/algorithm/ranges/equal>284 <http://en.cppreference.com/w/cpp/algorithm/ranges/fill>285 http://en.cppreference.com/w/cpp/algorithm/ranges/fill_n286 <http://en.cppreference.com/w/cpp/algorithm/ranges/find>287 http://en.cppreference.com/w/cpp/algorithm/ranges/find_end288 http://en.cppreference.com/w/cpp/algorithm/ranges/find_first_of289 <http://en.cppreference.com/w/cpp/algorithm/ranges/find>290 <http://en.cppreference.com/w/cpp/algorithm/ranges/find>291 http://en.cppreference.com/w/cpp/algorithm/ranges/for_each292 http://en.cppreference.com/w/cpp/algorithm/ranges/for_each_n293 <http://en.cppreference.com/w/cpp/algorithm/ranges/generate>294 http://en.cppreference.com/w/cpp/algorithm/ranges/generate_n295 <http://en.cppreference.com/w/cpp/algorithm/ranges/includes>296 http://en.cppreference.com/w/cpp/algorithm/ranges/inplace_merge297 http://en.cppreference.com/w/cpp/algorithm/ranges/is_heap298 http://en.cppreference.com/w/cpp/algorithm/ranges/is_heap_until299 http://en.cppreference.com/w/cpp/algorithm/ranges/is_partitioned300 http://en.cppreference.com/w/cpp/algorithm/ranges/is_sorted301 http://en.cppreference.com/w/cpp/algorithm/ranges/is_sorted_until302 http://en.cppreference.com/w/cpp/algorithm/ranges/make_heap303 <http://en.cppreference.com/w/cpp/algorithm/ranges/merge>304 <http://en.cppreference.com/w/cpp/algorithm/ranges/move>305 http://en.cppreference.com/w/cpp/algorithm/ranges/all_any_none_of306 http://en.cppreference.com/w/cpp/algorithm/ranges/nth_element307 http://en.cppreference.com/w/cpp/algorithm/ranges/partial_sort308 http://en.cppreference.com/w/cpp/algorithm/ranges/partial_sort_copy309 <http://en.cppreference.com/w/cpp/algorithm/ranges/partition>310 http://en.cppreference.com/w/cpp/algorithm/ranges/partition_copy311 http://en.cppreference.com/w/cpp/algorithm/ranges/set_difference312 http://en.cppreference.com/w/cpp/algorithm/ranges/set_intersection313 http://en.cppreference.com/w/cpp/algorithm/ranges/set_symmetric_difference314 http://en.cppreference.com/w/cpp/algorithm/ranges/set_union315 <https://wg21.link/p2440>316 <https://wg21.link/p2440>317 <http://en.cppreference.com/w/cpp/algorithm/ranges/sort>318 http://en.cppreference.com/w/cpp/algorithm/ranges/stable_partition319 http://en.cppreference.com/w/cpp/algorithm/ranges/stable_sort320 http://en.cppreference.com/w/cpp/algorithm/ranges/starts_with321 http://en.cppreference.com/w/cpp/algorithm/ranges/swap_ranges322 <http://en.cppreference.com/w/cpp/algorithm/ranges/unique>323 http://en.cppreference.com/w/cpp/algorithm/ranges/unique_copy324 <http://wg21.link/n4808>325 <http://wg21.link/n4808>

hpx/any.hpp

The header `hpx/any.hpp`³²⁶ corresponds to the C++ standard library header `any`³²⁷.

`hpx::any` is compatible with `std::any`.

Classes

Table 2.38: Classes of header `hpx/any.hpp`

Class	C++ standard
<code>hpx::any</code>	<code>std::any</code> ³²⁸
<code>hpx::any_nons</code>	
<code>hpx::bad_any_cast</code>	<code>std::bad_any_cast</code> ³²⁹
<code>hpx::unique_any_nons</code>	

Functions

Table 2.39: Functions of header `hpx/any.hpp`

Function	C++ standard
<code>hpx::any_cast</code>	<code>std::any_cast</code> ³³⁰
<code>hpx::make_any</code>	<code>std::make_any</code> ³³¹
<code>hpx::make_any_nons</code>	
<code>hpx::make_unique_any_nons</code>	

hpx/assert.hpp

The header `hpx/assert.hpp`³³² corresponds to the C++ standard library header `cassert`³³³.

`HPX_ASSERT` is the HPX equivalent to `assert` in `cassert`. `HPX_ASSERT` can also be used in CUDA device code.

Macros

hpx/barrier.hpp

The header `hpx/barrier.hpp`³³⁴ corresponds to the C++ standard library header `barrier`³³⁵ and contains a distributed barrier implementation. This functionality is also exposed through the `hpx::distributed` namespace. The name in `hpx::distributed` should be preferred.

³²⁶ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/any.hpp>

³²⁷ <http://en.cppreference.com/w/cpp/header/any>

³²⁸ <http://en.cppreference.com/w/cpp/utility/any>

³²⁹ http://en.cppreference.com/w/cpp/utility/any/bad_any_cast

³³⁰ http://en.cppreference.com/w/cpp/utility/any/any_cast

³³¹ http://en.cppreference.com/w/cpp/utility/any/make_any

³³² <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/core/assertion/include/hpx/assert.hpp>

³³³ <http://en.cppreference.com/w/cpp/header/cassert>

³³⁴ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/barrier.hpp>

³³⁵ <http://en.cppreference.com/w/cpp/header/barrier>

Classes

Table 2.40: Classes of header `hpx/barrier.hpp`

Class	C++ standard
<code>hpx::barrier</code>	<code>std::barrier</code> ³³⁶

`hpx/channel.hpp`

The header `hpx/channel.hpp`³³⁷ contains a local and a distributed channel implementation. This functionality is also exposed through the `hpx::distributed` namespace. The name in `hpx::distributed` should be preferred.

Classes

`hpx/chrono.hpp`

The header `hpx/chrono.hpp`³³⁸ corresponds to the C++ standard library header `chrono`³³⁹. The following replacements and extensions are provided compared to `chrono`³⁴⁰.

Classes

Table 2.41: Classes of header `hpx/chrono.hpp`

Class	C++ standard
<code>hpx::chrono::high_resolution_clock</code>	<code>std::high_resolution_clock</code> ³⁴¹
<code>hpx::chrono::high_resolution_timer</code>	
<code>hpx::chrono::steady_time_point</code>	

`hpx/condition_variable.hpp`

The header `hpx/condition_variable.hpp`³⁴² corresponds to the C++ standard library header `condition_variable`³⁴³.

³³⁶ <http://en.cppreference.com/w/cpp/thread/barrier>

³³⁷ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/channel.hpp>

³³⁸ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/chrono.hpp>

³³⁹ <http://en.cppreference.com/w/cpp/header/chrono>

³⁴⁰ <http://en.cppreference.com/w/cpp/header/chrono>

³⁴¹ http://en.cppreference.com/w/cpp/chrono/high_resolution_clock

³⁴² http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/condition_variable.hpp

³⁴³ http://en.cppreference.com/w/cpp/header/condition_variable

Classes

Table 2.42: Classes of header `hpx/condition_variable.hpp`

Class	C++ standard
<code>hpx::condition_variable</code>	<code>std::condition_variable</code> ³⁴⁴
<code>hpx::condition_variable_any</code>	<code>std::condition_variable_any</code> ³⁴⁵
<code>hpx::cv_status</code>	<code>std::cv_status</code> ³⁴⁶

`hpx/exception.hpp`

The header `hpx/exception.hpp`³⁴⁷ corresponds to the C++ standard library header `exception`³⁴⁸. `hpx::exception` extends `std::exception` and is the base class for all exceptions thrown in HPX. `HPX_THROW_EXCEPTION` can be used to throw HPX exceptions with file and line information attached to the exception.

Macros

- `HPX_THROW_EXCEPTION`

Classes

Table 2.43: Classes of header `hpx/exception.hpp`

Class	C++ standard
<code>hpx::exception</code>	<code>std::exception</code> ³⁴⁹

`hpx/execution.hpp`

The header `hpx/execution.hpp`³⁵⁰ corresponds to the C++ standard library header `execution`³⁵¹. See *High level parallel facilities*, *Using parallel algorithms* and *Executor parameters and executor parameter traits* for more information about execution policies and executor parameters.

Note: These names are only available in the `hpx::execution` namespace, not in the top-level `hpx` namespace.

³⁴⁴ http://en.cppreference.com/w/cpp/thread/condition_variable

³⁴⁵ http://en.cppreference.com/w/cpp/thread/condition_variable_any

³⁴⁶ http://en.cppreference.com/w/cpp/thread/cv_status

³⁴⁷ <https://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/exception.hpp>

³⁴⁸ <http://en.cppreference.com/w/cpp/header/exception>

³⁴⁹ <http://en.cppreference.com/w/cpp/error/exception>

³⁵⁰ <https://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/execution.hpp>

³⁵¹ <http://en.cppreference.com/w/cpp/header/execution>

Constants

Table 2.44: Constants of header `hpx/execution.hpp`

Constant	C++ standard
<code>hpx::execution::seq</code>	<code>std::execution_policy_tag</code> ³⁵²
<code>hpx::execution::par</code>	<code>std::execution_policy_tag</code> ³⁵³
<code>hpx::execution::par_unseq</code>	<code>std::execution_policy_tag</code> ³⁵⁴
<code>hpx::execution::task</code>	

Classes

Table 2.45: Classes of header `hpx/execution.hpp`

Class	C++ standard
<code>hpx::execution::sequenced_policy</code>	<code>std::execution_policy_tag_t</code> ³⁵⁵
<code>hpx::execution::parallel_policy</code>	<code>std::execution_policy_tag_t</code> ³⁵⁶
<code>hpx::execution::parallel_unsequenced_policy</code>	<code>std::execution_policy_tag_t</code> ³⁵⁷
<code>hpx::execution::sequenced_task_policy</code>	
<code>hpx::execution::parallel_task_policy</code>	
<code>hpx::execution::auto_chunk_size</code>	
<code>hpx::execution::dynamic_chunk_size</code>	
<code>hpx::execution::guided_chunk_size</code>	
<code>hpx::execution::persistent_auto_chunk_size</code>	
<code>hpx::execution::static_chunk_size</code>	

`hpx/functional.hpp`

The header `hpx/functional.hpp`³⁵⁸ corresponds to the C++ standard library header `functional`³⁵⁹. `hpx::function` is a more efficient and serializable replacement for `std::function`.

Constants

The following constants correspond to the C++ standard `std::placeholders`³⁶⁰

³⁵² http://en.cppreference.com/w/cpp/algorithm/execution_policy_tag

³⁵³ http://en.cppreference.com/w/cpp/algorithm/execution_policy_tag

³⁵⁴ http://en.cppreference.com/w/cpp/algorithm/execution_policy_tag

³⁵⁵ http://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t

³⁵⁶ http://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t

³⁵⁷ http://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t

³⁵⁸ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/functional.hpp>

³⁵⁹ <http://en.cppreference.com/w/cpp/header/functional>

³⁶⁰ <http://en.cppreference.com/w/cpp/utility/functional/placeholders>

Classes

Table 2.46: Classes of header `hpx/functional.hpp`

Class	C++ standard
<code>hpx::function</code>	<code>std::function</code> ³⁶¹
<code>hpx::function_ref</code>	
<code>hpx::move_only_function</code>	<code>std::move_only_function</code> ³⁶²
<code>hpx::traits::is_bind_expression</code>	<code>std::is_bind_expression</code> ³⁶³
<code>hpx::traits::is_placeholder</code>	<code>std::is_placeholder</code> ³⁶⁴
<code>hpx::scoped_annotation</code>	

Functions

Table 2.47: Functions of header `hpx/functional.hpp`

Function	C++ standard
<code>hpx::annotated_function</code>	
<code>hpx::bind</code>	<code>std::bind</code> ³⁶⁵
<code>hpx::experimental::bind_back</code>	
<code>hpx::bind_front</code>	<code>std::bind_front</code> ³⁶⁶
<code>hpx::invoke</code>	<code>std::invoke</code> ³⁶⁷
<code>hpx::util::invoke_fused</code>	
<code>hpx::mem_fn</code>	<code>std::mem_fn</code> ³⁶⁸

`hpx/future.hpp`

The header `hpx/future.hpp`³⁶⁹ corresponds to the C++ standard library header `future`³⁷⁰. See *Extended facilities for futures* for more information about extensions to futures compared to the C++ standard library.

This header file also contains overloads of `hpx::async`, `hpx::apply`, `hpx::sync`, and `hpx::dataflow` that can be used with actions. See *Action invocation* for more information about invoking actions.

³⁶¹ <http://en.cppreference.com/w/cpp/utility/functional/function>

³⁶² http://en.cppreference.com/w/cpp/utility/functional/move_only_function

³⁶³ http://en.cppreference.com/w/cpp/utility/functional/is_bind_expression

³⁶⁴ http://en.cppreference.com/w/cpp/utility/functional/is_placeholder

³⁶⁵ <http://en.cppreference.com/w/cpp/utility/functional/bind>

³⁶⁶ http://en.cppreference.com/w/cpp/utility/functional/bind_front

³⁶⁷ <http://en.cppreference.com/w/cpp/utility/functional/invoke>

³⁶⁸ http://en.cppreference.com/w/cpp/utility/functional/mem_fn

³⁶⁹ <http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/future.hpp>

³⁷⁰ <http://en.cppreference.com/w/cpp/header/future>

Classes

Table 2.48: Classes of header `hpx/future.hpp`

Class	C++ standard
<code>hpx::future</code>	<code>std::future</code> ³⁷¹
<code>hpx::shared_future</code>	<code>std::shared_future</code> ³⁷²
<code>hpx::promise</code>	<code>std::promise</code> ³⁷³
<code>hpx::launch</code>	<code>std::launch</code> ³⁷⁴
<code>hpx::packaged_task</code>	<code>std::packaged_task</code> ³⁷⁵

Note: All names except `hpx::promise` are also available in the top-level `hpx` namespace. `hpx::promise` refers to `hpx::distributed::promise`, a distributed variant of `hpx::promise`, but will eventually refer to `hpx::promise` after a deprecation period.

Functions

Table 2.49: Functions of header `hpx/future.hpp`

Function	C++ standard
<code>hpx::async</code>	<code>std::async</code> ³⁷⁶
<code>hpx::apply</code>	
<code>hpx::sync</code>	
<code>hpx::dataflow</code>	
<code>hpx::make_future</code>	
<code>hpx::make_shared_future</code>	
<code>hpx::make_ready_future</code>	
<code>hpx::make_ready_future_alloc</code>	
<code>hpx::make_ready_future_at</code>	
<code>hpx::make_ready_future_after</code>	
<code>hpx::make_exceptional_future</code>	
<code>hpx::async</code>	
<code>hpx::apply</code>	
<code>hpx::sync</code>	
<code>hpx::dataflow</code>	
<code>hpx::when_all</code>	
<code>hpx::when_any</code>	
<code>hpx::when_some</code>	
<code>hpx::when_each</code>	
<code>hpx::wait_all</code>	
<code>hpx::wait_any</code>	
<code>hpx::wait_some</code>	
<code>hpx::wait_each</code>	

³⁷¹ <http://en.cppreference.com/w/cpp/thread/future>³⁷² http://en.cppreference.com/w/cpp/thread/shared_future³⁷³ <http://en.cppreference.com/w/cpp/thread/promise>³⁷⁴ <http://en.cppreference.com/w/cpp/thread/launch>³⁷⁵ http://en.cppreference.com/w/cpp/thread/packaged_task³⁷⁶ <http://en.cppreference.com/w/cpp/thread/async>

Examples

```
#include <hpx/assert.hpp>
#include <hpx/future.hpp>
#include <hpx/hpx_main.hpp>
#include <hpx/tuple.hpp>

#include <iostream>
#include <utility>

int main()
{
    // Asynchronous execution with futures
    hpx::future<void> f1 = hpx::async(hpx::launch::async, []() { });
    hpx::shared_future<int> f2 =
        hpx::async(hpx::launch::async, []() { return 42; });
    hpx::future<int> f3 =
        f2.then([](hpx::shared_future<int>&& f) { return f.get() * 3; });

    hpx::promise<double> p;
    auto f4 = p.get_future();
    HPX_ASSERT(!f4.is_ready());
    p.set_value(123.45);
    HPX_ASSERT(f4.is_ready());

    hpx::packaged_task<int()> t([]() { return 43; });
    hpx::future<int> f5 = t.get_future();
    HPX_ASSERT(!f5.is_ready());
    t();
    HPX_ASSERT(f5.is_ready());

    // Fire-and-forget
    hpx::apply([]() {
        std::cout << "This will be printed later\n" << std::flush;
    });

    // Synchronous execution
    hpx::sync([]() {
        std::cout << "This will be printed immediately\n" << std::flush;
    });

    // Combinators
    hpx::future<double> f6 = hpx::async([]() { return 3.14; });
    hpx::future<double> f7 = hpx::async([]() { return 42.0; });
    std::cout
        << hpx::when_all(f6, f7)
        .then([](hpx::future<
                    hpx::tuple<hpx::future<double>, hpx::future<double>>>
                    f) {
            hpx::tuple<hpx::future<double>, hpx::future<double>> t =
                f.get();
            double pi = hpx::get<0>(t).get();
            double r = hpx::get<1>(t).get();
            return pi * r * r;
        })
        .get()
    << std::endl;
```

(continues on next page)

(continued from previous page)

```

// Easier continuations with dataflow; it waits for all future or
// shared_future arguments before executing the continuation, and also
// accepts non-future arguments
hpx::future<double> f8 = hpx::async([]() { return 3.14; });
hpx::future<double> f9 = hpx::make_ready_future(42.0);
hpx::shared_future<double> f10 = hpx::async([]() { return 123.45; });
hpx::future<hpx::tuple<double, double>> f11 = hpx::dataflow(
    [] (hpx::future<double> a, hpx::future<double> b,
        hpx::shared_future<double> c, double d) {
        return hpx::make_tuple<>(a.get() + b.get(), c.get() / d);
    },
    f8, f9, f10, -3.9);

// split_future gives a tuple of futures from a future of tuple
hpx::tuple<hpx::future<double>, hpx::future<double>> f12 =
    hpx::split_future(std::move(f11));
std::cout << hpx::get<1>(f12).get() << std::endl;

return 0;
}

```

hpx/init.hpp

The header `hpx/init.hpp`³⁷⁷ contains functionality for starting, stopping, suspending, and resuming the *HPX* runtime. This is the main way to explicitly start the *HPX* runtime. See [Starting the HPX runtime](#) for more details on starting the *HPX* runtime.

Classes

Functions

hpx/latch.hpp

The header `hpx/latch.hpp`³⁷⁸ corresponds to the C++ standard library header `latch`³⁷⁹. It contains a local and a distributed latch implementation. This functionality is also exposed through the `hpx::distributed` namespace. The name in `hpx::distributed` should be preferred.

Classes

Table 2.50: Classes of header `hpx/latch.hpp`

Class	C++ standard
<code>hpx::latch</code>	<code>std::latch</code> ³⁸⁰

³⁷⁷ http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/init_runtime/include/hpx/init.hpp

³⁷⁸ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/latch.hpp>

³⁷⁹ <http://en.cppreference.com/w/cpp/header/latch>

³⁸⁰ <http://en.cppreference.com/w/cpp/thread/latch>

hpx/mutex.hpp

The header `hpx/mutex.hpp`³⁸¹ corresponds to the C++ standard library header `mutex`³⁸².

Classes

Table 2.51: Classes of header `hpx/mutex.hpp`

Class	C++ standard
<code>hpx::mutex</code>	<code>std::mutex</code> ³⁸³
<code>hpx::no_mutex</code>	
<code>hpx::once_flag</code>	<code>std::once_flag</code> ³⁸⁴
<code>hpx::recursive_mutex</code>	<code>std::recursive_mutex</code> ³⁸⁵
<code>hpx::spinlock</code>	
<code>hpx::timed_mutex</code>	<code>std::timed_mutex</code> ³⁸⁶
<code>hpx::unlock_guard</code>	

Functions

Table 2.52: Functions of header `hpx/mutex.hpp`

Function	C++ standard
<code>hpx::call_once</code>	<code>std::call_once</code> ³⁸⁷

hpx/memory.hpp

The header `hpx/memory.hpp`³⁸⁸ corresponds to the C++ standard library header `memory`³⁸⁹. It contains parallel versions of the copy, fill, move, and construct helper functions in `memory`³⁹⁰. See *Using parallel algorithms* for more information about the parallel algorithms.

³⁸¹ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/mutex.hpp>

³⁸² <http://en.cppreference.com/w/cpp/header/mutex>

³⁸³ <http://en.cppreference.com/w/cpp/thread/mutex>

³⁸⁴ http://en.cppreference.com/w/cpp/thread/once_flag

³⁸⁵ http://en.cppreference.com/w/cpp/thread/recursive_mutex

³⁸⁶ http://en.cppreference.com/w/cpp/thread/timed_mutex

³⁸⁷ http://en.cppreference.com/w/cpp/thread/call_once

³⁸⁸ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/memory.hpp>

³⁸⁹ <http://en.cppreference.com/w/cpp/header/memory>

³⁹⁰ <http://en.cppreference.com/w/cpp/header/memory>

Functions

Table 2.53: *hpx* functions of header `hpx/memory.hpp`

<i>hpx</i> function	C++ standard
<code>hpx::uninitialized_copy</code>	<code>std::uninitialized_copy</code> ³⁹¹
<code>hpx::uninitialized_copy_n</code>	<code>std::uninitialized_copy_n</code> ³⁹²
<code>hpx::uninitialized_default_construct</code>	<code>std::uninitialized_default_construct</code> ³⁹³
<code>hpx::uninitialized_default_construct_n</code>	<code>std::uninitialized_default_construct_n</code> ³⁹⁴
<code>hpx::uninitialized_fill</code>	<code>std::uninitialized_fill</code> ³⁹⁵
<code>hpx::uninitialized_fill_n</code>	<code>std::uninitialized_fill_n</code> ³⁹⁶
<code>hpx::uninitialized_move</code>	<code>std::uninitialized_move</code> ³⁹⁷
<code>hpx::uninitialized_move_n</code>	<code>std::uninitialized_move_n</code> ³⁹⁸
<code>hpx::uninitialized_value_construct</code>	<code>std::uninitialized_value_construct</code> ³⁹⁹
<code>hpx::uninitialized_value_construct_n</code>	<code>std::uninitialized_value_construct_n</code> ⁴⁰⁰

Table 2.54: *hpx::ranges* functions of header `hpx/memory.hpp`

<i>hpx::ranges</i> function	C++ standard
<code>hpx::ranges::uninitialized_copy</code>	<code>std::uninitialized_copy</code> ⁴⁰¹
<code>hpx::ranges::uninitialized_copy_n</code>	<code>std::uninitialized_copy_n</code> ⁴⁰²
<code>hpx::ranges::uninitialized_default_construct</code>	<code>std::uninitialized_default_construct</code> ⁴⁰³
<code>hpx::ranges::uninitialized_default_construct_n</code>	<code>std::uninitialized_default_construct_n</code> ⁴⁰⁴
<code>hpx::ranges::uninitialized_fill</code>	<code>std::uninitialized_fill</code> ⁴⁰⁵
<code>hpx::ranges::uninitialized_fill_n</code>	<code>std::uninitialized_fill_n</code> ⁴⁰⁶
<code>hpx::ranges::uninitialized_move</code>	<code>std::uninitialized_move</code> ⁴⁰⁷
<code>hpx::ranges::uninitialized_move_n</code>	<code>std::uninitialized_move_n</code> ⁴⁰⁸
<code>hpx::ranges::uninitialized_value_construct</code>	<code>std::uninitialized_value_construct</code> ⁴⁰⁹
<code>hpx::ranges::uninitialized_value_construct_n</code>	<code>std::uninitialized_value_construct_n</code> ⁴¹⁰

³⁹¹ http://en.cppreference.com/w/cpp/memory/uninitialized_copy³⁹² http://en.cppreference.com/w/cpp/memory/uninitialized_copy_n³⁹³ http://en.cppreference.com/w/cpp/memory/uninitialized_default_construct³⁹⁴ http://en.cppreference.com/w/cpp/memory/uninitialized_default_construct_n³⁹⁵ http://en.cppreference.com/w/cpp/memory/uninitialized_fill³⁹⁶ http://en.cppreference.com/w/cpp/memory/uninitialized_fill_n³⁹⁷ http://en.cppreference.com/w/cpp/memory/uninitialized_move³⁹⁸ http://en.cppreference.com/w/cpp/memory/uninitialized_move_n³⁹⁹ http://en.cppreference.com/w/cpp/memory/uninitialized_value_construct⁴⁰⁰ http://en.cppreference.com/w/cpp/memory/uninitialized_value_construct_n⁴⁰¹ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_copy⁴⁰² http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_copy_n⁴⁰³ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_default_construct⁴⁰⁴ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_default_construct_n⁴⁰⁵ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_fill⁴⁰⁶ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_fill_n⁴⁰⁷ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_move⁴⁰⁸ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_move_n⁴⁰⁹ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_value_construct⁴¹⁰ http://en.cppreference.com/w/cpp/memory/ranges/uninitialized_value_construct_n

hpx/numeric.hpp

The header `hpx/numeric.hpp`⁴¹¹ corresponds to the C++ standard library header `numeric`⁴¹². See [Using parallel algorithms](#) for more information about the parallel algorithms.

Functions

Table 2.55: *hpx* functions of header `hpx/numeric.hpp`

<i>hpx</i> function	C++ standard
<code>hpx::adjacent_difference</code>	<code>std::adjacent_difference</code> ⁴¹³
<code>hpx::exclusive_scan</code>	<code>std::exclusive_scan</code> ⁴¹⁴
<code>hpx::inclusive_scan</code>	<code>std::inclusive_scan</code> ⁴¹⁵
<code>hpx::reduce</code>	<code>std::reduce</code> ⁴¹⁶
<code>hpx::transform_exclusive_scan</code>	<code>std::transform_exclusive_scan</code> ⁴¹⁷
<code>hpx::transform_inclusive_scan</code>	<code>std::transform_inclusive_scan</code> ⁴¹⁸
<code>hpx::transform_reduce</code>	<code>std::transform_reduce</code> ⁴¹⁹

hpx/optional.hpp

The header `hpx/optional.hpp`⁴²⁰ corresponds to the C++ standard library header `optional`⁴²¹. `hpx::optional` is compatible with `std::optional`.

Constants

- `hpx::nullopt`

Classes

Table 2.56: Classes of header `hpx/optional.hpp`

Class	C++ standard
<code>hpx::optional</code>	<code>std::optional</code> ⁴²²
<code>hpx::nullopt_t</code>	<code>std::nullopt_t</code> ⁴²³
<code>hpx::bad_optional_access</code>	<code>std::bad_optional_access</code> ⁴²⁴

⁴¹¹ <http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/numeric.hpp>

⁴¹² <http://en.cppreference.com/w/cpp/header/numeric>

⁴¹³ http://en.cppreference.com/w/cpp/algorithm/adjacent_difference

⁴¹⁴ http://en.cppreference.com/w/cpp/algorithm/exclusive_scan

⁴¹⁵ http://en.cppreference.com/w/cpp/algorithm/inclusive_scan

⁴¹⁶ <http://en.cppreference.com/w/cpp/algorithm/reduce>

⁴¹⁷ http://en.cppreference.com/w/cpp/algorithm/transform_exclusive_scan

⁴¹⁸ http://en.cppreference.com/w/cpp/algorithm/transform_inclusive_scan

⁴¹⁹ http://en.cppreference.com/w/cpp/algorithm/transform_reduce

⁴²⁰ <http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/optional.hpp>

⁴²¹ <http://en.cppreference.com/w/cpp/header/optional>

⁴²² <http://en.cppreference.com/w/cpp/utility/optional>

⁴²³ http://en.cppreference.com/w/cpp/utility/nullopt_t

⁴²⁴ http://en.cppreference.com/w/cpp/utility/optional/bad_optional_access

`hpx/runtime.hpp`

The header `hpx/runtime.hpp`⁴²⁵ contains functions for accessing local and distributed runtime information.

Typedefs

Functions

`hpx/source_location.hpp`

The header `hpx/source_location.hpp`⁴²⁶ corresponds to the C++ standard library header `source_location`⁴²⁷.

Classes

Table 2.57: Classes of header `hpx/system_error.hpp`

Class	C++ standard
<code>hpx::source_location</code>	<code>std::source_location</code> ⁴²⁸

`hpx/system_error.hpp`

The header `hpx/system_error.hpp`⁴²⁹ corresponds to the C++ standard library header `system_error`⁴³⁰.

Classes

Table 2.58: Classes of header `hpx/system_error.hpp`

Class	C++ standard
<code>hpx::error_code</code>	<code>std::error_code</code> ⁴³¹

`hpx/task_block.hpp`

The header `hpx/task_block.hpp`⁴³² corresponds to the `task_block` feature in N4411⁴³³. See [Using task blocks](#) for more details on using task blocks.

⁴²⁵ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/runtime.hpp>

⁴²⁶ http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/source_location.hpp

⁴²⁷ http://en.cppreference.com/w/cpp/header/source_location

⁴²⁸ http://en.cppreference.com/w/cpp/utility/source_location

⁴²⁹ http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/system_error.hpp

⁴³⁰ http://en.cppreference.com/w/cpp/header/system_error

⁴³¹ http://en.cppreference.com/w/cpp/error/error_code

⁴³² http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/task_block.hpp

⁴³³ <http://wg21.link/n4411>

Classes

Functions

`hpx/thread.hpp`

The header `hpx/thread.hpp`⁴³⁴ corresponds to the C++ standard library header `thread`⁴³⁵. The functionality in this header is equivalent to the standard library thread functionality, with the exception that the *HPX* equivalents are implemented on top of lightweight threads and the *HPX* runtime.

Classes

Table 2.59: Classes of header `hpx/thread.hpp`

Class	C++ standard
<code>hpx::thread</code>	<code>std::thread</code> ⁴³⁶
<code>hpx::jthread</code>	<code>std::jthread</code> ⁴³⁷

Functions

`hpx/semaphore.hpp`

The header `hpx/semaphore.hpp`⁴³⁸ corresponds to the C++ standard library header `semaphore`⁴³⁹.

Classes

Table 2.60: Classes of header `hpx/semaphore.hpp`

Class	C++ standard
<code>hpx::binary_semaphore</code>	<code>std::counting_semaphore</code> ⁴⁴⁰
<code>hpx::counting_semaphore</code>	<code>std::counting_semaphore</code> ⁴⁴¹

⁴³⁴ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/thread.hpp>

⁴³⁵ <http://en.cppreference.com/w/cpp/header/thread>

⁴³⁶ <http://en.cppreference.com/w/cpp/thread/thread>

⁴³⁷ <http://en.cppreference.com/w/cpp/thread/jthread>

⁴³⁸ <http://github.com/STELLAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/semaphore.hpp>

⁴³⁹ <http://en.cppreference.com/w/cpp/header/semaphore>

⁴⁴⁰ http://en.cppreference.com/w/cpp/thread/counting_semaphore

⁴⁴¹ http://en.cppreference.com/w/cpp/thread/counting_semaphore

hpx/shared_mutex.hpp

The header `hpx/shared_mutex.hpp`⁴⁴² corresponds to the C++ standard library header `shared_mutex`⁴⁴³.

ClassesTable 2.61: Classes of header `hpx/shared_mutex.hpp`

Class	C++ standard
<code>hpx::shared_mutex</code>	<code>std::shared_mutex</code> ⁴⁴⁴

hpx/stop_token.hpp

The header `hpx/stop_token.hpp`⁴⁴⁵ corresponds to the C++ standard library header `stop_token`⁴⁴⁶.

ConstantsTable 2.62: Constants of header `hpx/stop_token.hpp`

Constant	C++ standard
<code>hpx::nostopstate</code>	<code>std::nostopstate</code> ⁴⁴⁷

ClassesTable 2.63: Classes of header `hpx/stop_token.hpp`

Class	C++ standard
<code>hpx::stop_callback</code>	<code>std::stop_callback</code> ⁴⁴⁸
<code>hpx::stop_source</code>	<code>std::stop_source</code> ⁴⁴⁹
<code>hpx::stop_token</code>	<code>std::stop_token</code> ⁴⁵⁰
<code>hpx::nostopstate_t</code>	<code>std::nostopstate_t</code> ⁴⁵¹

⁴⁴² http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/shared_mutex.hpp

⁴⁴³ http://en.cppreference.com/w/cpp/header/shared_mutex

⁴⁴⁴ http://en.cppreference.com/w/cpp/thread/shared_mutex

⁴⁴⁵ http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/stop_token.hpp

⁴⁴⁶ http://en.cppreference.com/w/cpp/header/stop_token

⁴⁴⁷ http://en.cppreference.com/w/cpp/thread/stop_source/nostopstate

⁴⁴⁸ http://en.cppreference.com/w/cpp/thread/stop_callback

⁴⁴⁹ http://en.cppreference.com/w/cpp/thread/stop_source

⁴⁵⁰ http://en.cppreference.com/w/cpp/thread/stop_token

⁴⁵¹ http://en.cppreference.com/w/cpp/thread/stop_source/nostopstate_t

hpx/tuple.hpp

The header `hpx/tuple.hpp`⁴⁵² corresponds to the C++ standard library header `tuple`⁴⁵³. `hpx::tuple` can be used in CUDA device code, unlike `std::tuple`.

Constants

Table 2.64: Constants of header `hpx/tuple.hpp`

Constant	C++ standard
<code>hpx::ignore</code>	<code>std::ignore</code> ⁴⁵⁴

Classes

Table 2.65: Classes of header `hpx/tuple.hpp`

Class	C++ standard
<code>hpx::tuple</code>	<code>std::tuple</code> ⁴⁵⁵
<code>hpx::tuple_size</code>	<code>std::tuple_size</code> ⁴⁵⁶
<code>hpx::tuple_element</code>	<code>std::tuple_element</code> ⁴⁵⁷

Functions

Table 2.66: Functions of header `hpx/tuple.hpp`

Function	C++ standard
<code>hpx::make_tuple</code>	<code>std::tuple_element</code> ⁴⁵⁸
<code>hpx::tie</code>	<code>std::tie</code> ⁴⁵⁹
<code>hpx::forward_as_tuple</code>	<code>std::forward_as_tuple</code> ⁴⁶⁰
<code>hpx::tuple_cat</code>	<code>std::tuple_cat</code> ⁴⁶¹
<code>hpx::get</code>	<code>std::get</code> ⁴⁶²

⁴⁵² <http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/tuple.hpp>

⁴⁵³ <http://en.cppreference.com/w/cpp/header/tuple>

⁴⁵⁴ <http://en.cppreference.com/w/cpp/utility/tuple/ignore>

⁴⁵⁵ <http://en.cppreference.com/w/cpp/utility/tuple/tuple>

⁴⁵⁶ http://en.cppreference.com/w/cpp/utility/tuple/tuple_size

⁴⁵⁷ http://en.cppreference.com/w/cpp/utility/tuple/tuple_element

⁴⁵⁸ http://en.cppreference.com/w/cpp/utility/tuple/tuple_element

⁴⁵⁹ <http://en.cppreference.com/w/cpp/utility/tuple/tie>

⁴⁶⁰ http://en.cppreference.com/w/cpp/utility/tuple/forward_as_tuple

⁴⁶¹ http://en.cppreference.com/w/cpp/utility/tuple/tuple_cat

⁴⁶² <http://en.cppreference.com/w/cpp/utility/tuple/get>

hpx/type_traits.hpp

The header `hpx/type_traits.hpp`⁴⁶³ corresponds to the C++ standard library header `type_traits`⁴⁶⁴.

Classes

Table 2.67: Classes of header `hpx/type_traits.hpp`

Class	C++ standard
<code>hpx::is_invocable</code>	<code>std::is_invocable</code> ⁴⁶⁵
<code>hpx::is_invocable_r</code>	<code>std::is_invocable</code> ⁴⁶⁶

hpx/unwrap.hpp

The header `hpx/unwrap.hpp`⁴⁶⁷ contains utilities for unwrapping futures.

Classes**Functions****hpx/version.hpp**

The header `hpx/version.hpp`⁴⁶⁸ provides version information about *HPX*.

Macros**Functions****hpx/wrap_main.hpp**

The header `hpx/wrap_main.hpp`⁴⁶⁹ does not provide any direct functionality but is used for implicitly using `main` as the runtime entry point. See *Re-use the main() function as the main HPX entry point* for more details on implicitly starting the *HPX* runtime.

⁴⁶³ http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/type_traits.hpp

⁴⁶⁴ http://en.cppreference.com/w/cpp/header/type_traits

⁴⁶⁵ http://en.cppreference.com/w/cpp/types/is_invocable

⁴⁶⁶ http://en.cppreference.com/w/cpp/types/is_invocable

⁴⁶⁷ <http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/full/include/include/hpx/unwrap.hpp>

⁴⁶⁸ <http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/libs/core/version/include/hpx/version.hpp>

⁴⁶⁹ http://github.com/STEllAR-GROUP/hpx/blob/a018072601b1d55e6802bf6e6c0152af74e9ea14/include/hpx/wrap_main.hpp

2.8.2 Full API

The full API of *HPX* is presented below. The listings for the public API above refer to the full documentation below.

Note: Most names listed in the full API reference are implementation details or considered unstable. They are listed mostly for completeness. If there is a particular feature you think deserves being in the public API we may consider promoting it. In general we prioritize making sure features corresponding to C++ standard library features are stable and complete.

algorithms

See *Public API* for a list of names and headers that are part of the public *HPX* API.

hpx/parallel/task_block.hpp

See *Public API* for a list of names and headers that are part of the public *HPX* API.

```
namespace hpx
```

```
namespace parallel
```

Functions

```
template<typename ExPolicy, typename F>
hpx::future<void> define_task_block(ExPolicy &&policy, F &&f)
    Constructs a task_block, tr, using the given execution policy policy, and invokes the expression f(tr)
    on the user-provided object, f.
```

Postcondition: All tasks spawned from *f* have finished execution. A call to `define_task_block` may return on a different thread than that on which it was called.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the task block may be parallelized.
- `F`: The type of the user defined function to invoke inside the `define_task_block` (deduced). *F* shall be `MoveConstructible`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `f`: The user defined function to invoke inside the task block. Given an lvalue *tr* of type `task_block`, the expression, `(void)f(tr)`, shall be well-formed.

Note It is expected (but not mandated) that *f* will (directly or indirectly) call `tr.run(callable_object)`.

Exceptions

- An: `exception_list`, as specified in Exception Handling.

```
template<typename ExPolicy, typename F>
void define_task_block(ExPolicy &&policy, F &&f)

template<typename F>
```

```
void define_task_block(F &&f)
```

Constructs a `task_block`, *tr*, and invokes the expression *f(tr)* on the user-provided object, *f*. This version uses `parallel_policy` for task scheduling.

Postcondition: All tasks spawned from *f* have finished execution. A call to `define_task_block` may return on a different thread than that on which it was called.

Template Parameters

- *F*: The type of the user defined function to invoke inside the `define_task_block` (deduced). *F* shall be `MoveConstructible`.

Parameters

- *f*: The user defined function to invoke inside the task block. Given an lvalue *tr* of type `task_block`, the expression, `(void)f(tr)`, shall be well-formed.

Note It is expected (but not mandated) that *f* will (directly or indirectly) call *tr.run(callable_object)*.

Exceptions

- An: `exception_list`, as specified in Exception Handling.

```
template<typename ExPolicy, typename F>
util::detail::algorithm_result<ExPolicy>::type define_task_block_restore_thread(ExPolicy
&&pol-
icy,
F
&&f)
```

Constructs a `task_block`, *tr*, and invokes the expression *f(tr)* on the user-provided object, *f*.

Postcondition: All tasks spawned from *f* have finished execution. A call to `define_task_block_restore_thread` always returns on the same thread as that on which it was called.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the task block may be parallelized.
- *F*: The type of the user defined function to invoke inside the `define_task_block` (deduced). *F* shall be `MoveConstructible`.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *f*: The user defined function to invoke inside the `define_task_block`. Given an lvalue *tr* of type `task_block`, the expression, `(void)f(tr)`, shall be well-formed.

Exceptions

- An: `exception_list`, as specified in Exception Handling.

Note It is expected (but not mandated) that *f* will (directly or indirectly) call *tr.run(callable_object)*.

```
template<typename F>
void define_task_block_restore_thread(F &&f)
Constructs a task_block, tr, and invokes the expression f(tr) on the user-provided object, f. This version uses parallel_policy for task scheduling.
```

Postcondition: All tasks spawned from *f* have finished execution. A call to `define_task_block_restore_thread` always returns on the same thread as that on which it was called.

Template Parameters

- *F*: The type of the user defined function to invoke inside the `define_task_block` (deduced). *F* shall be `MoveConstructible`.

Parameters

- *f*: The user defined function to invoke inside the `define_task_block`. Given an lvalue *tr* of type `task_block`, the expression, `(void)f(tr)`, shall be well-formed.

Exceptions

- An: *exception_list*, as specified in Exception Handling.

Note It is expected (but not mandated) that *f* will (directly or indirectly) call *tr.run(callable_object)*.

namespace v2

```
template<typename ExPolicy = hpx::execution::parallel_policy>
class task_block
```

#include <task_block.hpp> The class `task_block` defines an interface for forking and joining parallel tasks. The `define_task_block` and `define_task_block_restore_thread` function templates create an object of type `task_block` and pass a reference to that object to a user-provided callable object.

An object of class `task_block` cannot be constructed, destroyed, copied, or moved except by the implementation of the task region library. Taking the address of a `task_block` object via operator& or addressof is ill formed. The result of obtaining its address by any other means is unspecified.

A `task_block` is active if it was created by the nearest enclosing task block, where “task block” refers to an invocation of `define_task_block` or `define_task_block_restore_thread` and “nearest enclosing” means the most recent invocation that has not yet completed. Code designated for execution in another thread by means other than the facilities in this section (e.g., using `thread` or `async`) are not enclosed in the task region and a `task_block` passed to (or captured by) such code is not active within that code. Performing any operation on a `task_block` that is not active results in undefined behavior.

The `task_block` that is active before a specific call to the `run` member function is not active within the asynchronous function that invoked `run`. (The invoked function should not, therefore, capture the `task_block` from the surrounding block.)

Example:

```
define_task_block([&] (auto& tr) {
    tr.run([&] {
        tr.run([] { f(); });
        // Error: tr is not active
    });
    define_task_block([&] (auto& tr) {
        tr.run(f());
        // Nested task block
        // OK: inner tr is active
    });
    // ...
});
```

Template Parameters

- `ExPolicy`: The execution policy an instance of a `task_block` was created with. This defaults to `parallel_policy`.

Public Types

```
template<>
using execution_policy = ExPolicy
    Refers to the type of the execution policy used to create the task\_block.
```

Public Functions

execution_policy const &get_execution_policy() const
 Return the execution policy instance used to create this [task_block](#)

```
template<typename F, typename ...Ts>
void run(F &&f, Ts&&... ts)
```

Causes the expression f() to be invoked asynchronously. The invocation of f is permitted to run on an unspecified thread in an unordered fashion relative to the sequence of operations following the call to run(f) (the continuation), or indeterminately sequenced within the same thread as the continuation.

The call to *run* synchronizes with the invocation of f. The completion of f() synchronizes with the next invocation of wait on the same [task_block](#) or completion of the nearest enclosing task block (i.e., the *define_task_block* or *define_task_block_restore_thread* that created this task block).

Requires: F shall be MoveConstructible. The expression, (void)f(), shall be well-formed.

Precondition: this shall be the active [task_block](#).

Postconditions: A call to run may return on a different thread than that on which it was called.

Note The call to *run* is sequenced before the continuation as if *run* returns on the same thread.

The invocation of the user-supplied callable object f may be immediate or may be delayed until compute resources are available. *run* might or might not return before invocation of f completes.

Exceptions

- This function may throw [task_canceled_exception](#), as described in Exception Handling.

```
template<typename Executor, typename F, typename ...Ts>
void run(Executor &&exec, F &&f, Ts&&... ts)
```

Causes the expression f() to be invoked asynchronously using the given executor. The invocation of f is permitted to run on an unspecified thread associated with the given executor and in an unordered fashion relative to the sequence of operations following the call to run(exec, f) (the continuation), or indeterminately sequenced within the same thread as the continuation.

The call to *run* synchronizes with the invocation of f. The completion of f() synchronizes with the next invocation of wait on the same [task_block](#) or completion of the nearest enclosing task block (i.e., the *define_task_block* or *define_task_block_restore_thread* that created this task block).

Requires: Executor shall be a type modeling the Executor concept. F shall be MoveConstructible. The expression, (void)f(), shall be well-formed.

Precondition: this shall be the active [task_block](#).

Postconditions: A call to run may return on a different thread than that on which it was called.

Note The call to *run* is sequenced before the continuation as if *run* returns on the same thread. The invocation of the user-supplied callable object *f* may be immediate or may be delayed until compute resources are available. *run* might or might not return before invocation of *f* completes.

Exceptions

- This: function may throw `task_canceled_exception`, as described in Exception Handling. The function will also throw a `exception_list` holding all exceptions that were caught while executing the tasks.

`void wait()`

Blocks until the tasks spawned using this `task_block` have finished.

Precondition: this shall be the active `task_block`.

Postcondition: All tasks spawned by the nearest enclosing task region have finished. A call to *wait* may return on a different thread than that on which it was called.

Example:

```
define_task_block([&] (auto& tr) {
    tr.run([&]{ process(a, w, x); }); // Process a[w] through
→ a[x]
    if (y < x) tr.wait(); // Wait if overlap between [w, x]
→ and [y, z)
    process(a, y, z); // Process a[y] through a[z]
});
```

Note The call to *wait* is sequenced before the continuation as if *wait* returns on the same thread.

Exceptions

- This: function may throw `task_canceled_exception`, as described in Exception Handling. The function will also throw a `exception_list` holding all exceptions that were caught while executing the tasks.

`ExPolicy &policy()`

Returns a reference to the execution policy used to construct this object.

Precondition: this shall be the active `task_block`.

`ExPolicy const &policy() const`

Returns a reference to the execution policy used to construct this object.

Precondition: this shall be the active `task_block`.

Private Members

`hpx::execution::experimental::task_group tasks_`

`threads::thread_id_type id_`

`ExPolicy policy_`

`class task_canceled_exception : public exception`

`#include <task_block.hpp>` The class `task_canceled_exception` defines the type of objects thrown by `task_block::run` or `task_block::wait` if they detect that an exception is pending within the current parallel region.

Public Functions

```
task_canceled_exception()
```

hpx/parallel/task_group.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
namespace experimental
```

```
class task_group
```

Public Functions

```
task_group()
```

```
~task_group()
```

```
template<typename Executor, typename F, typename ...Ts>
void run (Executor &&exec, F &&f, Ts&&... ts)
```

```
template<typename F, typename ...Ts>
void run (F &&f, Ts&&... ts)
```

```
void wait ()
```

```
void add_exception (std::exception_ptr p)
```

Private Types

```
using shared_state_type = lcos::detail::future_data<void>
```

Private Functions

```
void serialize (serialization::input_archive&, unsigned const)
```

```
void serialize (serialization::output_archive&, unsigned const)
```

Private Members

```
hpx::lcos::local::latch latch_
hpx::intrusive_ptr<shared_state_type> state_
hpx::exception_list errors_
std::atomic<bool> has_arrived_
```

Friends

```
friend hpx::execution::experimental::serialization::access
struct on_exit
```

Public Functions

```
on_exit(task_group &tg)
~on_exit()
on_exit(on_exit const &rhs)
on_exit &operator=(on_exit const &rhs)
on_exit(on_exit &&rhs)
on_exit &operator=(on_exit &&rhs)
```

Public Members

```
hpx::lcos::local::latch *latch_
```

hpx/parallel/container_algorithms/adjacent_difference.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter, typename Sent, typename Proj = hpx::parallel::util::projection_identity, typename Pred = hpx::parallel::util::predicate_identity>
FwdIter adjacent_difference(FwdIter first, Sent last, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Searches the range [first, last) for two consecutive identical elements.

Note Complexity: Exactly the smaller of (result - first) + 1 and (last - first) - 1 application of the predicate where *result* is the value returned

Return The *adjacent_find* algorithm returns an iterator to the first of the identical elements. If no such elements are found, *last* is returned.

Template Parameters

- *FwdIter*: The type of the source iterators used for the range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*
- *Pred*: The type of an optional function/function object to use.

Parameters

- *first*: Refers to the beginning of the sequence of elements of the range the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements of the range the algorithm will be applied to.
- *pred*: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The types *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Proj = hpx::parallel::util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type adjacent_find(ExPolicy &&policy,
                                                                     FwdIter first, Sent last,
                                                                     Pred &&pred = Pred(),
                                                                     Proj &&proj = Proj())
```

Searches the range [first, last) for two consecutive identical elements. This version uses the given binary predicate *pred*

The comparison operations in the parallel *adjacent_find* invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly the smaller of (*result* - *first*) + 1 and (*last* - *first*) - 1 application of the predicate where *result* is the value returned

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used for the range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *adjacent_find* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to *std::equal_to<>*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements of the range the algorithm will be applied to.

- `last`: Refers to the end of the sequence of elements of the range the algorithm will be applied to.
- `pred`: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel `adjacent_find` invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `adjacent_find` is available if the user decides to provide their algorithm their own binary predicate `pred`.

Return The `adjacent_find` algorithm returns a `hpx::future<InIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `InIter` otherwise. The `adjacent_find` algorithm returns an iterator to the first of the identical elements. If no such elements are found, `last` is returned.

```
template<typename Rng, typename Proj = hpx::parallel::util::projection_identity, typename Pred = detail::equal_to>
hpx::traits::range_traits<Rng>::iterator_type adjacent_difference(ExPolicy &&policy, Rng
&&rng, Pred &&pred =
Pred(), Proj &&proj =
Proj())
```

Searches the range `rng` for two consecutive identical elements.

Note Complexity: Exactly the smaller of `(result - std::begin(rng)) + 1` and `(std::begin(rng) - std::end(rng)) - 1` applications of the predicate where `result` is the value returned

Return The `adjacent_difference` algorithm returns an iterator to the first of the identical elements. If no such elements are found, `last` is returned.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`
- `Pred`: The type of an optional function/function object to use.

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

```
template<typename ExPolicy, typename Rng, typename Proj = hpx::parallel::util::projection_identity, typename Pred =
```

`util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type adjacent_find`

Searches the range `rng` for two consecutive identical elements.

The comparison operations in the parallel `adjacent_find` invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Exactly the smaller of `(result - std::begin(rng)) + 1` and `(std::begin(rng) - std::end(rng)) - 1` applications of the predicate where `result` is the value returned

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). Thhpx::traits::is_range<Rng>::valuee iterators extracted from this range type must meet the requirements of an forward iterator.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `adjacent_find` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: The binary predicate which returns `true` if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` is invoked.

The comparison operations in the parallel `adjacent_find` invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `adjacent_find` is available if the user decides to provide their algorithm their own binary predicate `pred`.

Return The `adjacent_find` algorithm returns a `hpx::future<InIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `InIter` otherwise. The `adjacent_find` algorithm returns an iterator to the first of the identical elements. If no such elements are found, `last` is returned.

Searches the range `rng` for two consecutive identical elements.

The comparison operations in the parallel `adjacent_find` invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Exactly the smaller of $(\text{result} - \text{std::begin}(\text{rng})) + 1$ and $(\text{std::begin}(\text{rng}) - \text{std::end}(\text{rng})) - 1$ applications of the predicate where `result` is the value returned

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `adjacent_find` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: The binary predicate which returns `true` if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel `adjacent_find` invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `adjacent_find` is available if the user decides to provide their algorithm their own binary predicate `pred`.

Return The `adjacent_find` algorithm returns a `hpx::future<InIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `InIter` otherwise. The `adjacent_find` algorithm returns an iterator to the first of the identical elements. If no such elements are found, `last` is returned.

[hpx/parallel/container_algorithms/adjacent_find.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

template<typename **FwdIter**, typename **Sent**, typename **Proj** = *hpx::parallel::util::projection_identity*, typename **Pred** = *hpx::traits::range_traits<Rng>::iterator_type*> **adjacent_find**(*FwdIter first*, *Sent last*, *Pred* &&*pred* = *Pred()*, *Proj* &&*proj* = *Proj()*)
Searches the range [first, last) for two consecutive identical elements.

Note Complexity: Exactly the smaller of (*result* - *first*) + 1 and (*last* - *first*) - 1 application of the predicate where *result* is the value returned

Return The *adjacent_find* algorithm returns an iterator to the first of the identical elements. If no such elements are found, *last* is returned.

Template Parameters

- **FwdIter**: The type of the source iterators used for the range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **InIter**.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*
- **Pred**: The type of an optional function/function object to use.

Parameters

- **first**: Refers to the beginning of the sequence of elements of the range the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements of the range the algorithm will be applied to.
- **pred**: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have **const &**, but the function must not modify the objects passed to it. The types *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

template<typename **Rng**, typename **Proj** = *hpx::parallel::util::projection_identity*, typename **Pred** = *detail::equal_to<hpx::traits::range_traits<Rng>::iterator_type>*> **adjacent_find**(*ExPolicy &&policy*, *Rng &&rng*, *Pred* &&*pred* = *Pred()*, *Proj* &&*proj* = *Proj()*)

Searches the range *rng* for two consecutive identical elements.

Note Complexity: Exactly the smaller of (*result* - *std::begin(rng)*) + 1 and (*std::begin(rng)* - *std::end(rng)*) - 1 applications of the predicate where *result* is the value returned

Return The *adjacent_find* algorithm returns an iterator to the first of the identical elements. If no such elements are found, *last* is returned.

Template Parameters

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*
- **Pred**: The type of an optional function/function object to use.

Parameters

- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **pred**: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

hpx/parallel/container_algorithms/all_any_none.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type none_of(ExPolicy &&policy, Rng &&rng, F
&&f, Proj &&proj = Proj())
```

Checks if unary predicate *f* returns true for no elements in the range *rng*.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: At most `std::distance(begin(rng), end(rng))` applications of the predicate *f*

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `none_of` requires *F* to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *none_of* algorithm returns a `hpx::future<bool>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `bool` otherwise. The *none_of* algorithm returns true if the unary predicate *f* returns true for no elements in the range, false otherwise. It returns true if the range is empty.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type any_of (ExPolicy &&policy, Rng &&rng, F
&&f, Proj &&proj = Proj())
```

Checks if unary predicate *f* returns true for at least one element in the range *rng*.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most `std::distance(begin(rng), end(rng))` applications of the predicate *f*

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Rng:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **F:** The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *none_of* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj:** The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *any_of* algorithm returns a `hpx::future<bool>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `bool` otherwise. The *any_of* algorithm returns true if the unary predicate *f* returns true for at least one element in the range, false otherwise. It returns false if the range is empty.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type all_of (ExPolicy &&policy, Rng &&rng, F
&&f, Proj &&proj = Proj())
```

Checks if unary predicate *f* returns true for all elements in the range *rng*.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most `std::distance(begin(rng), end(rng))` applications of the predicate *f*

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *none_of* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *all_of* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *all_of* algorithm returns true if the unary predicate *f* returns true for all elements in the range, false otherwise. It returns true if the range is empty.

hpx/parallel/container_algorithms/copy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename FwdIter>
hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::ranges::copy_result<Iter1, Iter>>::type copy(ExPolicy
    &&policy,
    Iter1
    iter,
    Sent1
    sent,
    FwdIter
    dest)
```

Copies the elements in the range, defined by [first, last), to another range beginning at *dest*.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter1**: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent1**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **FwdIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **iter**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **sent**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *copy* algorithm returns a `hpx::future<ranges::copy_result<FwdIter1, FwdIter>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `ranges::copy_result<FwdIter1, FwdIter>` otherwise. The *copy* algorithm returns the pair of the input iterator *last* and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng, typename FwdIter>
hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::ranges::copy_result<typename hpx::traits::range_traits<Rng>::
```

Copies the elements in the range *rng* to another range beginning at *dest*.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly `std::distance(begin(rng), end(rng))` assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **FwdIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.

The assignments in the parallel `copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `copy` algorithm returns a `hpx::future<ranges::copy_result<iterator_t<Rng>, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::copy_result<iterator_t<Rng>, FwdIter2>` otherwise. The `copy` algorithm returns the pair of the input iterator `last` and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Size, typename FwdIter2>
hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::ranges::copy_n_result<FwdIter1, FwdIter2>>::type copy_n(ExPolicy policy, FwdIter1 first, Size count, FwdIter2 dest)
```

Copies the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest.

The assignments in the parallel `copy_n` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly `count` assignments, if `count > 0`, no assignments otherwise.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Size`: The type of the argument specifying the number of elements to apply f to.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.

The assignments in the parallel `copy_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `copy_n` algorithm returns a `hpx::future<ranges::copy_n_result<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::copy_n_result<FwdIter1, FwdIter2>` otherwise. The `copy` algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter, typename F, typename P>
```

`hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::ranges::copy_if_result<typename hpx::traits::range_traits<Rng>`

Copies the elements in the range, defined by [first, last) to another range beginning at *dest*. Copies only the elements for which the predicate *f* returns true. The order of the elements that are not removed is preserved.

The assignments in the parallel *copy_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than std::distance(begin(rng), end(rng)) assignments, exactly std::distance(begin(rng), end(rng)) applications of the predicate *f*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent1**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for FwdIter1.
- **FwdIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *copy_if* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **iter**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **sent**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *copy_if* algorithm invoked with an execution policy object of type

parallel_policy or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *copy_if* algorithm returns a *hpx::future<ranges::copy_if_result<iterator_t<Rng>, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *ranges::copy_if_result<iterator_t<Rng>, FwdIter2>* otherwise. The *copy_if* algorithm returns the pair of the input iterator *last* and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng, typename OutIter, typename F, typename Proj = hpx::parallel::util::hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::ranges::copy_if_result<typename hpx::traits::range_traits<Rng>, FwdIter2>>>
```

Copies the elements in the range *rng* to another range beginning at *dest*. Copies only the elements for which the predicate *f* returns true. The order of the elements that are not removed is preserved.

The assignments in the parallel *copy_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than `std::distance(begin(rng), end(rng))` assignments, exactly `std::distance(begin(rng), end(rng))` applications of the predicate *f*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *copy_if* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects

passed to it. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *copy_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *copy_if* algorithm returns a *hpx::future<ranges::copy_if_result<iterator_t<Rng>, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *ranges::copy_if_result<iterator_t<Rng>, FwdIter2>* otherwise. The *copy_if* algorithm returns the pair of the input iterator *last* and the output iterator to the element in the destination range, one past the last element copied.

hpx/parallel/container_algorithms/count.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename ExPolicy, typename Rng, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<typename hpx::traits::range_traits<Rng>::iterator
```

Returns the number of elements in the range [first, last) satisfying a specific criteria. This version counts the elements that are equal to the given *value*.

The comparisons in the parallel *count* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* comparisons.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the comparisons.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *T*: The type of the value to search for (deduced).

- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `value`: The value to search for.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *f* is invoked.

Note The comparisons in the parallel `count` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `count` algorithm returns a `hpx::future<difference_type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `difference_type` otherwise (where `difference_type` is defined by `std::iterator_traits<FwdIter>::difference_type`). The `count` algorithm returns the number of elements satisfying the given criteria.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<typename hpx::traits::range_traits<Rng>::iterator>
```

Returns the number of elements in the range [first, last) satisfying a specific criteria. This version counts elements for which predicate *f* returns true.

Note Complexity: Performs exactly *last - first* applications of the predicate.

Note The assignments in the parallel `count_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note The assignments in the parallel `count_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `count_if` algorithm returns `hpx::future<difference_type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `difference_type` otherwise (where `difference_type` is defined by `std::iterator_traits<FwdIter>::difference_type`). The `count` algorithm returns the number of elements satisfying the given criteria.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the comparisons.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `count_if` requires *F* to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements

in the sequence specified by [first, last). This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have **const&**, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

hpx/parallel/container_algorithms/destroy.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy>
util::detail::algorithm_result<ExPolicy, typename traits::range_iterator<Rng>::type>::type destroy (ExPolicy
&&policy,
Rng
&&rng)
```

Destroys objects of type *typename iterator_traits<ForwardIt>::value_type* in the range [first, last).

The operations in the parallel *destroy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* operations.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.

The operations in the parallel *destroy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *destroy* algorithm returns a *hpx::future<void>*, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *void* otherwise.

```
template<typename ExPolicy, typename FwdIter, typename Size>
```

```
util::detail::algorithm_result<ExPolicy, FwdIter>::type destroy_n(ExPolicy &&policy, FwdIter  
first, Size count)  
Destroys objects of type typename iterator_traits<ForwardIt>::value_type in the range [first, first +  
count).
```

The operations in the parallel *destroy_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* operations, if *count* > 0, no assignments otherwise.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Size*: The type of the argument specifying the number of elements to apply this algorithm to.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *count*: Refers to the number of elements starting at *first* the algorithm will be applied to.

The operations in the parallel *destroy_n* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *destroy_n* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *destroy_n* algorithm returns the iterator to the element in the source range, one past the last element constructed.

hpx/parallel/container_algorithms/ends_with.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2, typename Pred, typename Proj1  
bool ends_with(FwdIter1 first1, Sent1 last1, FwdIter2 first2, Sent2 last2, Pred &&pred, Proj1  
&&proj1, Proj2 &&proj2)
```

Checks whether the second range defined by [first1, last1) matches the suffix of the first range defined by [first2, last2)

The assignments in the parallel *ends_with* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear: at most min(N1, N2) applications of the predicate and both projections.

Template Parameters

- *Iter1*: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an input iterator.

- **Sent1**: The type of the end source iterators used(deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Iter2**: The type of the begin destination iterators used deduced). This iterator type must meet the requirements of a input iterator.
- **Sent2**: The type of the end destination iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter2.
- **Pred**: The binary predicate that compares the projected elements.
- **Proj1**: The type of an optional projection function for the source range. This defaults to *util::projection_identity*
- **Proj1**: The type of an optional projection function for the destination range. This defaults to *util::projection_identity*

Parameters

- **first1**: Refers to the beginning of the source range.
- **last1**: Sentinel value referring to the end of the source range.
- **first2**: Refers to the beginning of the destination range.
- **last2**: Sentinel value referring to the end of the destination range.
- **pred**: Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the in two ranges projected by proj1 and proj2 respectively.
- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

Return The *ends_with* algorithm returns *bool*. The *ends_with* algorithm returns a boolean with the value true if the second range matches the suffix of the first range, false otherwise.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2, typename hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type ends_with(ExPolicy    &&policy,
                                                               FwdIter1
                                                               first1, Sent1 last1,
                                                               FwdIter2
                                                               first2,
                                                               Sent2 last2, Pred
                                                               &&pred, Proj1
                                                               &&proj1, Proj2
                                                               &&proj2)
```

Checks whether the second range defined by [first1, last1) matches the suffix of the first range defined by [first2, last2)

The assignments in the parallel *ends_with* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear: at most min(N1, N2) applications of the predicate and both projections.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent1**: The type of the end source iterators used(deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **FwdIter2**: The type of the begin destination iterators used deduced). This iterator type must meet the requirements of a forward iterator.
- **Sent2**: The type of the end destination iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter2.

- `Pred`: The binary predicate that compares the projected elements.
- `Proj1`: The type of an optional projection function for the source range. This defaults to `util::projection_identity`
- `Proj1`: The type of an optional projection function for the destination range. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the source range.
- `last1`: Sentinel value referring to the end of the source range.
- `first2`: Refers to the beginning of the destination range.
- `last2`: Sentinel value referring to the end of the destination range.
- `pred`: Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the two ranges projected by `proj1` and `proj2` respectively.
- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel `ends_with` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `ends_with` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `ends_with` algorithm returns a boolean with the value true if the second range matches the suffix of the first range, false otherwise.

```
template<typename Rng1, typename Rng2, typename Pred, typename Proj1, typename Proj2>
bool ends_with(Rng1 &&rng1, Rng2 &&rng2, Pred &&pred, Proj1 &&proj1, Proj2 &&proj2)
```

Checks whether the second range `rng2` matches the suffix of the first range `rng1`.

The assignments in the parallel `ends_with` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear: at most $\min(N_1, N_2)$ applications of the predicate and both projections.

Template Parameters

- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Rng2`: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Pred`: The binary predicate that compares the projected elements.
- `Proj1`: The type of an optional projection function for the source range. This defaults to `util::projection_identity`
- `Proj1`: The type of an optional projection function for the destination range. This defaults to `util::projection_identity`

Parameters

- `rng1`: Refers to the source range.
- `rng2`: Refers to the destination range.
- `pred`: Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the two ranges projected by `proj1` and `proj2` respectively.
- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

Return The *ends_with* algorithm returns *bool*. The *ends_with* algorithm returns a boolean with the value true if the second range matches the suffix of the first range, false otherwise.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred, typename Proj1, typename Proj2>
hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type ends_with(ExPolicy    &&policy,
                                                                           Rng1      &&rng1,
                                                                           Rng2      &&rng2,
                                                                           Pred       &&pred,
                                                                           Proj1     &&proj1,
                                                                           Proj2     &&proj2)
```

Checks whether the second range *rng2* matches the suffix of the first range *rng1*.

The assignments in the parallel *ends_with* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear: at most min(N1, N2) applications of the predicate and both projections.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Rng2*: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The binary predicate that compares the projected elements.
- *Proj1*: The type of an optional projection function for the source range. This defaults to *util::projection_identity*
- *Proj1*: The type of an optional projection function for the destination range. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the source range.
- *rng2*: Refers to the destination range.
- *pred*: Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the two ranges projected by *proj1* and *proj2* respectively.
- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *ends_with* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *ends_with* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *ends_with* algorithm returns a boolean with the value true if the second range matches the suffix of the first range, false otherwise.

hpx/parallel/container_algorithms/equal.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Pred>
util::detail::algorithm_result<ExPolicy, bool>::type equal(ExPolicy &&policy, Iter1 first1, Sent1
last1, Iter2 first2, Sent2 last2, Pred
&&op = Pred(), Proj1 &&proj1 =
Proj1(), Proj2 &&proj2 = Proj2())
```

Returns true if the range [first1, last1) is equal to the range [first2, last2), and false otherwise.

The comparison operations in the parallel *equal* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most $\min(\text{last1} - \text{first1}, \text{last2} - \text{first2})$ applications of the predicate *f*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent1**: The type of the source iterators used for the end of the first range (deduced).
- **Iter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent2**: The type of the source iterators used for the end of the second range (deduced).
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *equal* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- **Proj1**: The type of an optional projection function applied to the first range. This defaults to `util::projection_identity`
- **Proj2**: The type of an optional projection function applied to the second range. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **op**: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and *FwdIter2* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *equal* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The two ranges are considered equal if, for every iterator *i* in the range [first1, last1), **i* equals **(first2 + (i - first1))*. This overload of *equal* uses operator== to determine if two elements are equal.

Return The *equal* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *equal* algorithm returns true if the elements in the two ranges are equal, otherwise it returns false. If the length of the range [first1, last1) does not equal the length of the range [first2, last2), it returns false.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = ranges::equal_to, typename Proj1
util::detail::algorithm_result<ExPolicy, bool>::type equal(ExPolicy &&policy, Rng1 &&rng1,
Rng2 &&rng2, Pred &&op = Pred(),
Proj1 &&proj1 = Proj1(), Proj2
&&proj2 = Proj2())
```

Returns true if the range [first1, last1) is equal to the range starting at first2, and false otherwise.

The comparison operations in the parallel *equal* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most *last1 - first1* applications of the predicate *f*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the first source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Rng2*: The type of the second source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *equal* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to *std::equal_to<>*
- *Proj1*: The type of an optional projection function applied to the first range. This defaults to *util::projection_identity*
- *Proj2*: The type of an optional projection function applied to the second range. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the first sequence of elements the algorithm will be applied to.
- *rng2*: Refers to the second sequence of elements the algorithm will be applied to.
- *op*: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

<code>bool pred(const Type1 &a, const Type2 &b);</code>

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.

- proj1: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
 - proj2: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *equal* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The two ranges are considered equal if, for every iterator i in the range $[first1, last1)$, $*i$ equals $*(first2 + (i - first1))$. This overload of `equal` uses operator`==` to determine if two elements are equal.

Return The *equal* algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The *equal* algorithm returns true if the elements in the two ranges are equal, otherwise it returns false.

hpx/parallel/container algorithms/exclusive scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

template<typename **InIter**, typename **Sent**, typename **OutIter**, typename **T**>
exclusive_scan_result<InIter, OutIter> exclusive_scan (InIter first, Sent last, OutIter dest, T init)
Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED NONCOMMUTATIVE SUM($+$, init, $*first, \dots, *(first + (i - result) - 1)$)

The reduce operations in the parallel *exclusive_scan* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(\text{last} - \text{first})$ applications of the predicate `std::plus<T>`.

Template Parameters

- `InIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
 - `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter`.
 - `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
 - `T`: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
 - `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
 - `dest`: Refers to the beginning of the destination range.
 - `init`: The initial value for the generalized sum.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input element in the *i*th sum.

Return The *exclusive_scan* algorithm returns *util::in_out_result<InIter, OutIter>*. The *exclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename T>
util::detail::algorithm_result<ExPolicy, exclusive_scan_result<FwdIter1, FwdIter2>>::type exclusive_scan(ExPolicy &&policy, FwdIter1 first, Sent last, FwdIter2 dest, T init)
```

Assigns through each iterator *i* in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, init, *first, ..., *(first + (i - result) - 1))

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *std::plus<T>*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *init*: The initial value for the generalized sum.

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input

element in the i th sum.

Return The `exclusive_scan` algorithm returns a `hpx::future<util::in_out_result<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `util::in_out_result<FwdIter1, FwdIter2>` otherwise. The `exclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename InIter, typename Sent, typename OutIter, typename T, typename Op>
exclusive_scan_result<InIter, OutIter> exclusive_scan(InIter first, Sent last, OutIter dest, T
init, Op &&op)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, *first, ..., *(first + (i - result) - 1)).

The reduce operations in the parallel `exclusive_scan` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate `op`.

Template Parameters

- `InIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter`.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `dest`: Refers to the beginning of the destination range.
- `init`: The initial value for the generalized sum.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The difference between `exclusive_scan` and `inclusive_scan` is that `inclusive_scan` includes the i th input element in the i th sum. If `op` is not mathematically associative, the behavior of `inclusive_scan` may be non-deterministic.

Return The `exclusive_scan` algorithm returns `util::in_out_result<InIter, OutIter>`. The `exclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(`op`, a1, ..., aN) is defined as:

- a1 when N is 1

- `op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN))` where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename T, typename Op>
util::detail::algorithm_result<ExPolicy, exclusive_scan_result<FwdIter1, FwdIter2>>::type exclusive_scan(ExPolicy
    &&policy,
    FwdIter1
    first,
    Sent
    last,
    FwdIter2
    dest,
    T
    init,
    Op
    &&op)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of `GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, *first, ..., *(first + (i - result) - 1))`.

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter1`.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `dest`: Refers to the beginning of the destination range.
- `init`: The initial value for the generalized sum.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum. If op is not mathematically associative, the behavior of *inclusive_scan* may be non-deterministic.

Return The *exclusive_scan* algorithm returns a $hpx::future<util::in_out_result<FwdIter1, FwdIter2>>$ if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns $util::in_out_result<FwdIter1, FwdIter2>$ otherwise. The *exclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_N) is defined as:

- a_1 when N is 1
- $op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_K), GENERALIZED_NONCOMMUTATIVE_SUM(op, a_{M+1}, \dots, a_N))$ where $1 < K+1 = M \leq N$.

```
template<typename Rng, typename O, typename T>
exclusive_scan_result<traits::range_iterator_t<Rng>, O> exclusive_scan(Rng &&rng, O dest,
T init)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, init, *first, ..., *(first + (i - result) - 1))

The reduce operations in the parallel *exclusive_scan* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate $std::plus<T>$.

Template Parameters

- Rng : The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- O : The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- T : The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- rng : Refers to the sequence of elements the algorithm will be applied to.
- $dest$: Refers to the beginning of the destination range.
- $init$: The initial value for the generalized sum.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum.

Return The *exclusive_scan* algorithm returns $util::in_out_result<traits::range_iterator_t<Rng>, O>$. The *exclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a_1, \dots, a_N) is defined as:

- a_1 when N is 1
- $GENERALIZED_NONCOMMUTATIVE_SUM(+, a_1, \dots, a_K)$
 - $GENERALIZED_NONCOMMUTATIVE_SUM(+, a_{M+1}, \dots, a_N)$ where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng, typename O, typename T>
```

`util::detail::algorithm_result<ExPolicy, exclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type exclusive_scan`

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, init, *first, ..., *(first + (i - result) - 1))

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate `std::plus<T>`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `O`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `init`: The initial value for the generalized sum.

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum.

Return The *exclusive_scan* algorithm returns a `hpx::future<util::in_out_result<traits::range_iterator_t<Rng>, O>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `util::in_out_result<traits::range_iterator_t<Rng>, O>` otherwise. The *exclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename Rng, typename O, typename T, typename Op>
exclusive_scan_result<traits::range_iterator_t<Rng>, O> exclusive_scan(Rng &&rng, O dest,
                                                               T init, Op &&op)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, init, *first, ..., *(first + (i - result) - 1))

The reduce operations in the parallel *exclusive_scan* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate `std::plus<T>`.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `O`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `init`: The initial value for the generalized sum.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input element in the *i*th sum.

Return The *exclusive_scan* algorithm returns `util::in_out_result<traits::range_iterator_t<Rng>, O>`. The *exclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng, typename O, typename T, typename Op>
util::detail::algorithm_result<ExPolicy, exclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type exclusive_scan
```

Assigns through each iterator *i* in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, init, *first, ..., *(first + (i - result) - 1))

The reduce operations in the parallel *exclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate `std::plus<T>`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `O`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `init`: The initial value for the generalized sum.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel `exclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `exclusive_scan` and `inclusive_scan` is that `inclusive_scan` includes the i th input element in the i th sum.

Return The `exclusive_scan` algorithm returns a `hpx::future<util::in_out_result<traits::range_iterator_t<Rng>, O>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `util::in_out_result<traits::range_iterator_t<Rng>, O>` otherwise. The `exclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note `GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN)` is defined as:

- $a1$ when N is 1
- $\text{GENERALIZED_NONCOMMUTATIVE_SUM}(+, a1, \dots, aK)$
 - $\text{GENERALIZED_NONCOMMUTATIVE_SUM}(+, aM, \dots, aN)$ where $1 < K+1 = M \leq N$.

hpx/parallel/container_algorithms/fill.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename ExPolicy, typename Rng, typename T>
util::detail::algorithm_result<ExPolicy>::type fill(ExPolicy &&policy, Rng &&rng, T const &value)
```

Assigns the given value to the elements in the range [first, last).

The comparisons in the parallel *fill* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **T**: The type of the value to be assigned (deduced).

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **value**: The value to be assigned.

The comparisons in the parallel *fill* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *fill* algorithm returns a *hpx::future<void>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *difference_type* otherwise (where *difference_type* is defined by *void*).

```
template<typename ExPolicy, typename Iterator, typename Size, typename T>
util::detail::algorithm_result<ExPolicy, Iterator>::type fill_n(ExPolicy &&policy, Iterator first,
Size count, T const &value)
```

Assigns the given value *value* to the first *count* elements in the range beginning at *first* if *count* > 0. Does nothing otherwise.

The comparisons in the parallel *fill_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* assignments, for *count* > 0.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iterator**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Size**: The type of the argument specifying the number of elements to apply *f* to.
- **T**: The type of the value to be assigned (deduced).

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **count**: Refers to the number of elements starting at *first* the algorithm will be applied to.
- **value**: The value to be assigned.

The comparisons in the parallel *fill_n* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *fill_n* algorithm returns a *hpx::future<void>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *difference_type* otherwise (where *difference_type* is defined by *void*).

hpx/parallel/container_algorithms/find.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter, typename Sent, typename T, typename Proj = util::projection_identity<util::detail::algorithm_result<ExPolicy, Iter>>::type find(ExPolicy &&policy, Iter first, Sent last, T const &val, Proj &&proj = Proj())
```

Returns the first element in the range [first, last) that is equal to value

The comparison operations in the parallel *find* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most last - first applications of the operator==().

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter:** The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent:** The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter.
- **T:** The type of the value to find (deduced).
- **Proj:** The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **first:** Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last:** Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **val:** the value to compare the elements to
- **proj:** Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *find* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *find* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *find* algorithm returns the first element in the range [first,last) that is equal to *val*. If no such element in the range of [first,last) is equal to *val*, then the algorithm returns *last*.

```
template<typename ExPolicy, typename Rng, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, Iter>::type find(ExPolicy &&policy, Rng &&rng, T
                                                       const &val, Proj &&proj = Proj())
```

Returns the first element in the range [first, last) that is equal to value

The comparison operations in the parallel *find* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most last - first applications of the operator==().

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *T*: The type of the value to find (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *val*: the value to compare the elements to
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *find* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *find* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *find* algorithm returns the first element in the range [first,last) that is equal to *val*. If no such element in the range of [first,last) is equal to *val*, then the algorithm returns *last*.

```
template<typename ExPolicy, typename FwdIter, typename F>
util::detail::algorithm_result<ExPolicy, FwdIter>::type find_if(ExPolicy &&policy, Iter first, Iter
                                                               first, Sent last, Pred &&pred, Proj
                                                               &&proj = Proj())
```

Returns the first element in the range [first, last) for which predicate *pred* returns true

The comparison operations in the parallel *find_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most last - first applications of the predicate.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Iter*: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of a forward iterator.

- **Sent**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for *Iter*.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **pred**: The unary predicate which returns true for the required element. The signature of the predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have **const** **&**, but the function must not modify the objects passed to it. The type *Type* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *f* is invoked.

The comparison operations in the parallel *find_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *find_if* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *find_if* algorithm returns the first element in the range [first, last) that satisfies the predicate *f*. If no such element exists that satisfies the predicate *f*, the algorithm returns *last*.

```
template<typename ExPolicy, typename FwdIter, typename F>
util::detail::algorithm_result<ExPolicy, FwdIter>::type find_if (ExPolicy &&policy, Rng &&rng,
Pred &&pred, Proj &&proj =
Proj())
```

Returns the first element in the range *rng* for which predicate *pred* returns true

The comparison operations in the parallel *find_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most *last - first* applications of the predicate.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **pred**: The unary predicate which returns true for the required element. The signature of the predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type *Type* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *find_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *find_if* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *find_if* algorithm returns the first element in the range [first, last) that satisfies the predicate *f*. If no such element exists that satisfies the predicate *f*, the algorithm returns *last*.

```
template<typename ExPolicy, typename FwdIter, typename F>
util::detail::algorithm_result<ExPolicy, FwdIter>::type find_if_not(ExPolicy &&policy, Iter
first, Sent last, Pred
&&pred, Proj &&proj =
Proj())
```

Returns the first element in the range [first, last) for which predicate *f* returns false

The comparison operations in the parallel *find_if_not* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most *last - first* applications of the predicate.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Iter*: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent*: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for *Iter*.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- *pred*: The unary predicate which returns false for the required element. The signature of the predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type *Type* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *find_if_not* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *find_if_not* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *find_if_not* algorithm returns the first element in the range [first, last) that does **not** satisfy the predicate *f*. If no such element exists that does not satisfy the predicate *f*, the algorithm returns *last*.

```
template<typename ExPolicy, typename FwdIter, typename F>
util::detail::algorithm_result<ExPolicy, Iter>::type find_if_not (ExPolicy &&policy, Rng
&&rng, Pred &&pred, Proj
&&proj = Proj())
```

Returns the first element in the range *rng* for which predicate *f* returns false

The comparison operations in the parallel *find_if_not* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most last - first applications of the predicate.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: The unary predicate which returns false for the required element. The signature of the predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The type *Type* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *find_if_not* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *find_if_not* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *find_if_not* algorithm returns the first element in the range [first, last) that does **not** satisfy the predicate *f*. If no such element exists that does not satisfy the predicate *f*, the algorithm returns *last*.

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Proj
&&proj = Proj())
```

```
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng1>::type>::type find_end(ExPolicy  
&&policy,  
Iter1  
first1,  
Sent1  
last1,  
Iter2  
first2,  
Sent2  
last2,  
Pred  
&&op  
=  
Pred(),  
Proj1  
&&proj1  
=  
Proj10,  
Proj2  
&&proj2  
=  
Proj20)
```

Returns the last subsequence of elements [first2, last2) found in the range [first1, last1) using the given predicate *f* to compare elements.

The comparison operations in the parallel *find_end* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: at most $S^*(N-S+1)$ comparisons where $S = \text{distance}(\text{first2}, \text{last2})$ and $N = \text{distance}(\text{first1}, \text{last1})$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Iter1*: The type of the begin source iterators for the first sequence used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent1*: The type of the end source iterators for the first sequence used (deduced). This iterator type must meet the requirements of an sentinel for *Iter1*.
- *Iter2*: The type of the begin source iterators for the second sequence used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent2*: The type of the end source iterators for the second sequence used (deduced). This iterator type must meet the requirements of an sentinel for *Iter2*.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *replace* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first1*: Refers to the beginning of the first sequence of elements the algorithm will be applied to.

- `last1`: Refers to the end of the first sequence of elements the algorithm will be applied to.
- `first2`: Refers to the beginning of the second sequence of elements the algorithm will be applied to.
- `last2`: Refers to the end of the second sequence of elements the algorithm will be applied to.
- `op`: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `iterator_t<Rng>` and `iterator_t<Rng2>` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first range of type dereferenced `iterator_t<Rng1>` as a projection operation before the function `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second range of type dereferenced `iterator_t<Rng2>` as a projection operation before the function `op` is invoked.

The comparison operations in the parallel `find_end` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `find_end` is available if the user decides to provide the algorithm their own predicate `op`.

Return The `find_end` algorithm returns a `hpx::future<iterator_t<Rng>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `iterator_t<Rng>` otherwise. The `find_end` algorithm returns an iterator to the beginning of the last subsequence `rng2` in range `rng`. If the length of the subsequence `rng2` is greater than the length of the range `rng`, `end(rng)` is returned. Additionally if the size of the subsequence is empty or no subsequence is found, `end(rng)` is also returned.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = ranges::equal_to, typename Proj1
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng1>::type>::type find_end(ExPolicy
&&policy,
Rng1
&&rng,
Rng2
&&rng2,
Pred
&&op
=
Pred(),
Proj1
&&proj1
=
Proj1(),
Proj2
&&proj2
=
Proj2())
```

Returns the last subsequence of elements `rng2` found in the range `rng` using the given predicate `f` to compare elements.

The comparison operations in the parallel *find_end* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: at most $S^*(N-S+1)$ comparisons where $S = \text{distance}(\text{begin}(\text{rng2}), \text{end}(\text{rng2}))$ and $N = \text{distance}(\text{begin}(\text{rng}), \text{end}(\text{rng}))$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the first source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- *Rng2*: The type of the second source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *replace* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the first sequence of elements the algorithm will be applied to.
- *rng2*: Refers to the second sequence of elements the algorithm will be applied to.
- *op*: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *iterator_t<Rng>* and *iterator_t<Rng2>* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively.

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of the first range of type dereferenced *iterator_t<Rng1>* as a projection operation before the function *op* is invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of the second range of type dereferenced *iterator_t<Rng2>* as a projection operation before the function *op* is invoked.

The comparison operations in the parallel *find_end* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of *find_end* is available if the user decides to provide the algorithm their own predicate *op*.

Return The *find_end* algorithm returns a `hpx::future<iterator_t<Rng>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *iterator_t<Rng>* otherwise. The *find_end* algorithm returns an iterator to the beginning of the last subsequence *rng2* in range *rng*. If the length of the subsequence *rng2* is greater than the length of the range *rng*, *end(rng)* is returned. Additionally if the size of the subsequence is empty or no subsequence is found, *end(rng)* is also returned.

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Proj>
```

Searches the range [first1, last1) for any elements in the range [first2, last2). Uses binary predicate *p* to compare elements

The comparison operations in the parallel `find_first_of` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: at most (S^*N) comparisons where $S = \text{distance(first2, last2)}$ and $N = \text{distance(first1, last1)}$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
 - **Iter1**: The type of the begin source iterators for the first sequence used (deduced). This iterator type must meet the requirements of an forward iterator.
 - **Sent1**: The type of the end source iterators for the first sequence used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
 - **Iter2**: The type of the begin source iterators for the second sequence used (deduced). This iterator type must meet the requirements of an forward iterator.
 - **Sent2**: The type of the end source iterators for the second sequence used (deduced). This iterator type must meet the requirements of an sentinel for Iter2.
 - **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *replace* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
 - **Proj1**: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements in *rng1*.
 - **Proj2**: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements in *rng2*.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
 - **first1**: Refers to the beginning of the first sequence of elements the algorithm will be applied to.

- `last1`: Refers to the end of the first sequence of elements the algorithm will be applied to.
 - `first2`: Refers to the beginning of the second sequence of elements the algorithm will be applied to.
 - `last2`: Refers to the end of the second sequence of elements the algorithm will be applied to.
 - `op`: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `iterator_t<Rng1>` and `iterator_t<Rng2>` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.

- proj1: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced $iterator_t < Rng1 >$ before the function op is invoked.
 - proj2: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced $iterator_t < Rng2 >$ before the function op is invoked.

The comparison operations in the parallel `find_first_of` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `find_first_of` is available if the user decides to provide the algorithm their own predicate `op`.

Return The `find_end` algorithm returns a `hpx::future<iterator_t<Rng1>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `iterator_t<Rng1>` otherwise. The `find_first_of` algorithm returns an iterator to the first element in the range `rng1` that is equal to an element from the range `rng2`. If the length of the subsequence `rng2` is greater than the length of the range `rng1`, `end(rng1)` is returned. Additionally if the size of the subsequence is empty or no subsequence is found, `end(rng1)` is also returned.

Searches the range $rng1$ for any elements in the range $rng2$. Uses binary predicate p to compare elements.

The comparison operations in the parallel *find_first_of* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: at most (S^*N) comparisons where $S = \text{distance}(\text{begin}(\text{rng2}), \text{end}(\text{rng2}))$ and $N = \text{distance}(\text{begin}(\text{rng1}), \text{end}(\text{rng1}))$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the first source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- *Rng2*: The type of the second source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *replace* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj1*: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements in *rng1*.
- *Proj2*: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements in *rng2*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the first sequence of elements the algorithm will be applied to.
- *rng2*: Refers to the second sequence of elements the algorithm will be applied to.
- *op*: The binary predicate which returns *true* if the elements should be treated as equal. The signature should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *iterator_t<Rng1>* and *iterator_t<Rng2>* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively.

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced *iterator_t<Rng1>* before the function *op* is invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced *iterator_t<Rng2>* before the function *op* is invoked.

The comparison operations in the parallel *find_first_of* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of *find_first_of* is available if the user decides to provide the algorithm their own predicate *op*.

Return The *find_end* algorithm returns a `hpx::future<iterator_t<Rng1>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *iterator_t<Rng1>* otherwise. The *find_first_of* algorithm returns an iterator to the first element in the range *rng1* that is equal to an element from the range *rng2*. If the length of the subsequence *rng2* is greater than the length of the range *rng1*, *end(rng1)* is returned. Additionally if the size of the subsequence is empty or no subsequence is found, *end(rng1)* is also returned.

hpx/parallel/container_algorithms/for_each.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename InIter, typename Sent, typename F, typename Proj = util::projection_identity>
hpx::ranges::for_each_result<InIter, F> for_each (InIter first, Sent last, F &&f, Proj &&proj =
    Proj())
```

Applies *f* to the result of dereferencing every iterator in the range [first, last).

If *f* returns a result, the result is ignored.

Note Complexity: Applies *f* exactly *last - first* times.

If the type of *first* satisfies the requirements of a mutable iterator, *f* may apply non-constant functions through the dereferenced iterator.

Applies *f* to the result of dereferencing every iterator in the range [first, first + count), starting from first and proceeding to first + count - 1.

Return {last, HPX_MOVE(f)} where last is the iterator corresponding to the input sentinel last.

Template Parameters

- *InIter*: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *for_each* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *f*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have *const&*. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

If *f* returns a result, the result is ignored.

Note Complexity: Applies *f* exactly *last - first* times.

If the type of *first* satisfies the requirements of a mutable iterator, *f* may apply non-constant functions through the dereferenced iterator.

Return {first + count, HPX_MOVE(f)}

Template Parameters

- **InIter**: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an input iterator.
- **Size**: The type of the argument specifying the number of elements to apply *f* to.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *for_each* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **count**: Refers to the number of elements starting at *first* the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have **const&**. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename F, typename Proj = util::projection_id  
FwdIter for_each(ExPolicy &&policy, FwdIter first, Sent last, F &&f, Proj &&proj = Proj())
```

Applies *f* to the result of dereferencing every iterator in the range [first, last).

If *f* returns a result, the result is ignored.

Note Complexity: Applies *f* exactly *last - first* times.

If the type of *first* satisfies the requirements of a mutable iterator, *f* may apply non-constant functions through the dereferenced iterator.

Unlike its sequential form, the parallel overload of *for_each* does not return a copy of its *Function* parameter, since parallelization may not permit efficient state accumulation.

Return The *for_each* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. It returns *last*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **FwdIter**: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *for_each* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have `const&`. The type `Type` must be such that an object of type `InIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

```
template<typename Rng, typename F, typename Proj = util::projection_identity>
hpx::ranges::for_each_result<typename hpx::traits::range_iterator<Rng>::type, F> for_each(ExPolicy
    &&policy,
    Rng
    &&rng,
    F
    &&f,
    Proj
    &&proj
    =
    Proj())
```

Applies *f* to the result of dereferencing every iterator in the given range *rng*.

If *f* returns a result, the result is ignored.

Note Complexity: Applies *f* exactly $\text{size}(\text{rng})$ times.

If the type of *first* satisfies the requirements of a mutable iterator, *f* may apply non-constant functions through the dereferenced iterator.

Return {`std::end(rng)`, `HPX_MOVE(f)`}

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `for_each` requires `F` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have `const&`. The type `Type` must be such that an object of type `InIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
```

```
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type for_each (ExPolicy
&&policy,
Rng
&&rng,
F
&&f,
Proj
&&proj
=
Proj())
```

Applies *f* to the result of dereferencing every iterator in the given range *rng*.

If *f* returns a result, the result is ignored.

Note Complexity: Applies *f* exactly *size(rng)* times.

If the type of *first* satisfies the requirements of a mutable iterator, *f* may apply non-constant functions through the dereferenced iterator.

Unlike its sequential form, the parallel overload of *for_each* does not return a copy of its *Function* parameter, since parallelization may not permit efficient state accumulation.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *for_each* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have *const&*. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *for_each* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. It returns *last*.

```
template<typename ExPolicy, typename FwdIter, typename Size, typename F, typename Proj = util::projection_id
```

```
util::detail::algorithm_result<ExPolicy, FwdIter>::type for_each_n (ExPolicy &&policy, FwdIter  
first, Size count, F &&f, Proj  
&&proj = Proj())
```

Applies *f* to the result of dereferencing every iterator in the range [first, first + count), starting from first and proceeding to first + count - 1.

If *f* returns a result, the result is ignored.

Note Complexity: Applies *f* exactly *count* times.

If the type of *first* satisfies the requirements of a mutable iterator, *f* may apply non-constant functions through the dereferenced iterator.

Unlike its sequential form, the parallel overload of *for_each* does not return a copy of its *Function* parameter, since parallelization may not permit efficient state accumulation.

Return The *for_each* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. It returns *last*.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **FwdIter:** The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Size:** The type of the argument specifying the number of elements to apply *f* to.
- **F:** The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *for_each* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj:** The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **count:** Refers to the number of elements starting at *first* the algorithm will be applied to.
- **f:** Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have *const&*. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- **proj:** Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

hpx/parallel/container_algorithms/for_loop.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

```
namespace experimental
```

Functions

```
template<typename Iter, typename Sent, typename ...Args>
void for_loop(Iter first, Sent last, Args&&... args)
```

The for_loop implements loop functionality over a range specified by iterator bounds. These algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of for_loop without specifying an execution policy is equivalent to specifying `hpx::execution::seq` as the execution policy.

Requires: *Iter* shall meet the requirements of an input iterator type. The *args* parameter pack shall have at least one element, comprising objects returned by invocations of *reduction* and/or *induction* function templates followed by exactly one element invocable element-access function, *f*. *f* shall meet the requirements of MoveConstructible.

Template Parameters

- *Iter*: The type of the iteration variable (input iterator).
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *Iter*.
- *Args*: A parameter pack, its last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *args*: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Iter const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies *f* to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the *args* parameter pack. The length of the input sequence is *last - first*.

The first element in the input sequence is specified by *first*. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the *args* parameter pack excluding *f*, an additional argument is passed to each application of *f* as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using `advance` and `distance`.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of *f*, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of *f* in the input sequence.

Complexity: Applies f exactly once for each element of the input sequence.

Remarks: If f returns a result, the result is ignored.

```
template<typename ExPolicy, typename Iter, typename Sent, typename ...Args>
util::detail::algorithm_result<ExPolicy>::type for_loop(ExPolicy &&policy, Iter first, Sent
last, Args&&... args)
```

The `for_loop` implements loop functionality over a range specified by iterator bounds. These algorithms resemble `for_each` from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: `Iter` shall meet the requirements of a forward iterator type. The `args` parameter pack shall have at least one element, comprising objects returned by invocations of *reduction* and/or *induction* function templates followed by exactly one element invocable element-access function, f . f shall meet the requirements of `MoveConstructible`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Iter`: The type of the iteration variable (forward iterator).
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `Iter`.
- `Args`: A parameter pack, its last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `args`: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Iter const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the `args` parameter pack. The length of the input sequence is `last - first`.

The first element in the input sequence is specified by `first`. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the `args` parameter pack excluding f , an additional argument is passed to each application of f as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using `advance` and `distance`.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of f , even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an

object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of f in the input sequence.

Complexity: Applies f exactly once for each element of the input sequence.

Remarks: If f returns a result, the result is ignored.

Return The `for_loop` algorithm returns a `hpx::future<void>` if the execution policy is of type `hpx::execution::sequenced_task_policy` or `hpx::execution::parallel_task_policy` and returns `void` otherwise.

```
template<typename Rng, typename ...Args>
void for_loop (Rng &&rng, Args&&... args)
```

The `for_loop` implements loop functionality over a range specified by a range. These algorithms resemble `for_each` from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of `for_loop` without specifying an execution policy is equivalent to specifying `hpx::execution::seq` as the execution policy.

Requires: `Rng::iterator` shall meet the requirements of an input iterator type. The `args` parameter pack shall have at least one element, comprising objects returned by invocations of `reduction` and/or `induction` function templates followed by exactly one element invocable element-access function, f . f shall meet the requirements of `MoveConstructible`.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Args`: A parameter pack, it's last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `args`: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Rng::iterator const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the `args` parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by `first`. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the `args` parameter pack excluding f , an additional argument is passed to each application of f as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using `advance` and `distance`.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of f , even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of f in the input sequence.

Complexity: Applies f exactly once for each element of the input sequence.

Remarks: If f returns a result, the result is ignored.

```
template<typename ExPolicy, typename Rng, typename ...Args>
util::detail::algorithm_result<ExPolicy>::type for_loop(ExPolicy &&policy, Rng &&rng,
                                                       Args&&... args)
```

The for_loop implements loop functionality over a range specified by a range. These algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: $Rng::iterator$ shall meet the requirements of a forward iterator type. The $args$ parameter pack shall have at least one element, comprising objects returned by invocations of *reduction* and/or *induction* function templates followed by exactly one element invocable element-access function, f . f shall meet the requirements of MoveConstructible.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Args**: A parameter pack, its last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **args**: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Rng::iterator const& a, ...);
```

The signature does not need to have const&. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the $args$ parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by *first*. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the $args$ parameter pack excluding f , an additional argument is passed to each application of f as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of f , even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of f in the input sequence.

Complexity: Applies f exactly once for each element of the input sequence.

Remarks: If f returns a result, the result is ignored.

Return The `for_loop` algorithm returns a `hpx::future<void>` if the execution policy is of type `hpx::execution::sequenced_task_policy` or `hpx::execution::parallel_task_policy` and returns `void` otherwise.

```
template<typename Iter, typename Sent, typename S, typename ...Args>
void for_loop_strided(Iter first, Sent last, S stride, Args&&... args)
```

The `for_loop_strided` implements loop functionality over a range specified by iterator bounds. These algorithms resemble `for_each` from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of `for_loop_strided` without specifying an execution policy is equivalent to specifying `hpx::execution::seq` as the execution policy.

Requires: `Iter` shall meet the requirements of an input iterator type. The `args` parameter pack shall have at least one element, comprising objects returned by invocations of `reduction` and/or `induction` function templates followed by exactly one element invocable element-access function, f . f shall meet the requirements of `MoveConstructible`.

Template Parameters

- `Iter`: The type of the iteration variable (input iterator).
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `Iter`.
- `S`: The type of the stride variable. This should be an integral type.
- `Args`: A parameter pack, it's last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `stride`: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if `Iter` meets the requirements a bidirectional iterator.
- `args`: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Iter const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the `args` parameter pack. The length of the input sequence is `last - first`.

The first element in the input sequence is specified by `first`. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the *args* parameter pack excluding *f*, an additional argument is passed to each application of *f* as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of *f*, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of *f* in the input sequence.

Complexity: Applies *f* exactly once for each element of the input sequence.

Remarks: If *f* returns a result, the result is ignored.

template<typename ExPolicy, typename Iter, typename Sent, typename S, typename . . .>

The `for_loop_strided` implements loop functionality over a range specified by iterator bounds. These algorithms resemble `for_each` from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: *Iter* shall meet the requirements of a forward iterator type. The *args* parameter pack shall have at least one element, comprising objects returned by invocations of *reduction* and/or *induction* function templates followed by exactly one element invocable element-access function, *f*. *f* shall meet the requirements of *MoveConstructible*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- *Iter*: The type of the iteration variable (forward iterator).
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *Iter*.
- *S*: The type of the stride variable. This should be an integral type.
- *Args*: A parameter pack, its last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *stride*: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if *Iter* meets the requirements a bidirectional iterator.
- *args*: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Iter const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies *f* to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the *args* parameter pack. The length of the input sequence is

last - first.

The first element in the input sequence is specified by *first*. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the *args* parameter pack excluding *f*, an additional argument is passed to each application of *f* as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of *f*, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of *f* in the input sequence.

Complexity: Applies *f* exactly once for each element of the input sequence.

Remarks: If *f* returns a result, the result is ignored.

Return The *for_loop_strided* algorithm returns a *hpx::future<void>* if the execution policy is of type *hpx::execution::sequenced_task_policy* or *hpx::execution::parallel_task_policy* and returns *void* otherwise.

```
template<typename Rng, typename S, typename ...Args>
void for_loop_strided(Rng &&rng, S stride, Args&&... args)
```

The *for_loop_strided* implements loop functionality over a range specified by a range. These algorithms resemble *for_each* from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of *for_loop_strided* without specifying an execution policy is equivalent to specifying *hpx::execution::seq* as the execution policy.

Requires: *Rng::iterator* shall meet the requirements of an input iterator type. The *args* parameter pack shall have at least one element, comprising objects returned by invocations of *reduction* and/or *induction* function templates followed by exactly one element invocable element-access function, *f*. *f* shall meet the requirements of *MoveConstructible*.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *S*: The type of the stride variable. This should be an integral type.
- *Args*: A parameter pack, it's last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *stride*: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if *Rng::iterator* meets the requirements of a bidirectional iterator.
- *args*: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last] should expose a signature equivalent to:

```
<ignored> pred(Rng::iterator const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies `f` to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the `args` parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by `first`. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the `args` parameter pack excluding `f`, an additional argument is passed to each application of `f` as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using `advance` and `distance`.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of `f`, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of `f` in the input sequence.

Complexity: Applies `f` exactly once for each element of the input sequence.

Remarks: If `f` returns a result, the result is ignored.

```
template<typename ExPolicy, typename Rng, typename S, typename ...Args>
util::detail::algorithm_result<ExPolicy>::type for_loop_strided(ExPolicy      &&policy,
                                                               Rng &&rng, S stride,
                                                               Args&&... args)
```

The `for_loop_strided` implements loop functionality over a range specified by a range. These algorithms resemble `for_each` from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: `Rng::iterator` shall meet the requirements of a forward iterator type. The `args` parameter pack shall have at least one element, comprising objects returned by invocations of `reduction` and/or `induction` function templates followed by exactly one element invocable element-access function, `f`. `f` shall meet the requirements of `MoveConstructible`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `S`: The type of the stride variable. This should be an integral type.
- `Args`: A parameter pack, its last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `stride`: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if `Rng::iterator` meets the requirements of a bidirectional iterator.
- `args`: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction

objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last) should expose a signature equivalent to:

```
<ignored> pred(Rng::iterator const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the `args` parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by `first`. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the `args` parameter pack excluding f , an additional argument is passed to each application of f as follows:

Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using `advance` and `distance`.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of f , even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of f in the input sequence.

Complexity: Applies f exactly once for each element of the input sequence.

Remarks: If f returns a result, the result is ignored.

Return The `for_loop_strided` algorithm returns a `hpx::future<void>` if the execution policy is of type `hpx::execution::sequenced_task_policy` or `hpx::execution::parallel_task_policy` and returns `void` otherwise.

hpx/parallel/algorithms/for_loop_induction.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace experimental
```

Functions

```
template<typename T>
constexpr hpx::parallel::v2::detail::induction_stride_helper<T> induction(T      &&value,
                                                               std::size_t stride)
```

The function template returns an induction object of unspecified type having a value type and encapsulating an initial value `value` of that type and, optionally, a stride.

For each element in the input range, a looping algorithm over input sequence S computes an induction value from an induction variable and ordinal position p within S by the formula $i + p * \text{stride}$ if a stride was specified or $i + p$ otherwise. This induction value is passed to the element access function.

If the `value` argument to `induction` is a non-const lvalue, then that lvalue becomes the live-out object for the returned induction object. For each induction object that has a live-out object, the looping algorithm assigns the value of $i + n * \text{stride}$ to the live-out object upon return, where n is the number of elements in the input range.

Return This returns an induction object with value type T , initial value $value$, and (if specified) stride $stride$. If T is an lvalue of non-const type, $value$ is used as the live-out object for the induction object; otherwise there is no live-out object.

Template Parameters

- T: The value type to be used by the induction object.

Parameters

- `value`: [in] The initial value to use for the induction object
 - `stride`: [in] The (optional) stride to use for the induction object (default: 1)

namespace parallel

Functions

```
template<typename T>hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, "hpx::hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, "hpx::hpx::parallel::v2::HPX_DEPRECATED_V(1, 8,
```

Variables

```
    return hpx::experimental::induction(R, S, T, U);
```

hpx/parallel/algorithms/for_loop_reduction.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

```
namespace experimental
```

Functions

```
template<typename T, typename Op>
constexpr hpx::parallel::v2::detail::reduction_helper<T, std::decay_t<Op>> reduction(T
    &var,
    T
    const
    &iden-
    tity,
    Op
    &&com-
    biner)
```

The function template returns a reduction object of unspecified type having a value type and encapsulating an identity value for the reduction, a combiner function object, and a live-out object from which the initial value is obtained and into which the final value is stored.

A parallel algorithm uses reduction objects by allocating an unspecified number of instances, called views, of the reduction's value type. Each view is initialized with the reduction object's identity value,

except that the live-out object (which was initialized by the caller) comprises one of the views. The algorithm passes a reference to a view to each application of an element-access function, ensuring that no two concurrently-executing invocations share the same view. A view can be shared between two applications that do not execute concurrently, but initialization is performed only once per view.

Modifications to the view by the application of element access functions accumulate as partial results. At some point before the algorithm returns, the partial results are combined, two at a time, using the reduction object's combiner operation until a single value remains, which is then assigned back to the live-out object.

T shall meet the requirements of `CopyConstructible` and `MoveAssignable`. The expression `var = combiner(var, var)` shall be well formed.

Template Parameters

- *T*: The value type to be used by the induction object.
- *Op*: The type of the binary function (object) used to perform the reduction operation.

Parameters

- *var*: [in,out] The life-out value to use for the reduction object. This will hold the reduced value after the algorithm is finished executing.
- *identity*: [in] The identity value to use for the reduction operation.
- *combiner*: [in] The binary function (object) used to perform a pairwise reduction on the elements.

Note In order to produce useful results, modifications to the view should be limited to commutative operations closely related to the combiner operation. For example if the combiner is `plus<T>`, incrementing the view would be consistent with the combiner but doubling it or assigning to it would not.

Return This returns a reduction object of unspecified type having a value type of *T*. When the return value is used by an algorithm, the reference to *var* is used as the live-out object, new views are initialized to a copy of identity, and views are combined by invoking the copy of combiner, passing it the two views to be combined.

```
namespace parallel
```

Functions

```
template<typename T, typename Op>hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, "hpx::hp
template<typename T>hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, "hpx::hpx::parallel:
```

```
template<typename T>hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, "hpx::hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, \"hpx::hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, \"hpx::hpx::parallel::v2::HPX_DEPRECATED_V(1, 8, \"hpx::hpx::parallel::v2::HPX_DEPRECATED_V(1, 8,
```

Variables

hpx/parallel/container_algorithms/generate.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename ExPolicy, typename Rng, typename Futil::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type generate (ExPolicy
    && policy,
    Rng
    && rng,
    F
    && f)
```

Assign each element in range [first, last) a value generated by the given function object f

The assignments in the parallel `generate` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Exactly $distance(first, last)$ invocations of f and assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
 - **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: generator function that will be called. signature of function should be equivalent to the following:

```
Ret fun();
```

The type *Ret* must be such that an object of type *FwdIter* can be dereferenced and assigned a value of type *Ret*.

The assignments in the parallel *generate* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_if* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. It returns *last*.

```
template<typename ExPolicy, typename Iter, typename Sent, typename F>
util::detail::algorithm_result<ExPolicy, Iter>::type generate(ExPolicy &&policy, Iter first, Sent
last, F &&f)
```

Assign each element in range [first, last) a value generated by the given function object *f*

The assignments in the parallel *generate* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly *distance(first, last)* invocations of *f* and assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Iter*: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source end iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *f*: generator function that will be called. signature of function should be equivalent to the following:

```
Ret fun();
```

The type *Ret* must be such that an object of type *FwdIter* can be dereferenced and assigned a value of type *Ret*.

The assignments in the parallel *generate* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_if* algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. It returns *last*.

```
template<typename ExPolicy, typename FwdIter, typename Size, typename F>
util::detail::algorithm_result<ExPolicy, FwdIter>::type generate_n(ExPolicy &&policy, FwdIter
first, Size count, F &&f)
```

Assigns each element in range [first, first+count) a value generated by the given function object *g*.

The assignments in the parallel *generate_n* algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Exactly *count* invocations of *f* and assignments, for *count* > 0.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `count`: Refers to the number of elements in the sequence the algorithm will be applied to.
- `f`: Refers to the generator function object that will be called. The signature of the function should be equivalent to

```
Ret fun();
```

The type *Ret* must be such that an object of type *OutputIt* can be dereferenced and assigned a value of type *Ret*.

The assignments in the parallel *generate_n* algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_if* algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. It returns *last*.

hpx/parallel/container_algorithms/includes.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Pred>
util::detail::algorithm_result<ExPolicy, bool>::type::type includes (ExPolicy &&policy, Iter1
first1, Sent1 last1, Iter2 first2,
Sent2 last2, Pred &&op =
Pred(), Proj1 &&proj1 =
Proj1(), Proj2 &&proj2 =
Proj2())
```

Returns true if every element from the sorted range [first2, last2) is found within the sorted range [first1, last1). Also returns true if [first2, last2) is empty. The version expects both ranges to be sorted with the user supplied binary predicate *f*.

The comparison operations in the parallel *includes* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note At most $2^*(N1+N2-1)$ comparisons, where $N1 = \text{std}::\text{distance}(first1, last1)$ and $N2 = \text{std}::\text{distance}(first2, last2)$.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter1:** The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent1:** The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Iter2:** The type of the source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent2:** The type of the end source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an sentinel for Iter2.
- **Pred:** The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *includes* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to *std::less<*
- **Proj1:** The type of an optional projection function applied to the first sequence. This defaults to *util::projection_identity*
- **Proj2:** The type of an optional projection function applied to the second sequence. This defaults to *util::projection_identity*

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **first1:** Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1:** Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2:** Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2:** Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **op:** The binary predicate which returns true if the elements should be treated as includes. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and

FwdIter2 can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate *op* is invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate *op* is invoked.

The comparison operations in the parallel *includes* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *includes* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *includes* algorithm returns true every element from the sorted range [first2, last2) is found within the sorted range [first1, last1). Also returns true if [first2, last2) is empty.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = detail::less, typename Proj1 = util::util::detail::algorithm_result<ExPolicy, bool>::type::type includes(ExPolicy &&policy, Rng1 &&rng1, Rng2 &&rng2, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Returns true if every element from the sorted range [first2, last2) is found within the sorted range [first1, last1). Also returns true if [first2, last2) is empty. The version expects both ranges to be sorted with the user supplied binary predicate *f*.

The comparison operations in the parallel *includes* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note At most $2^*(N1+N2-1)$ comparisons, where $N1 = \text{std::distance}(\text{first1}, \text{last1})$ and $N2 = \text{std::distance}(\text{first2}, \text{last2})$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *includes* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to *std::less<>*
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to *util::projection_identity*
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the first sequence of elements the algorithm will be applied to.
- *rng2*: Refers to the second sequence of elements the algorithm will be applied to.
- *op*: The binary predicate which returns true if the elements should be treated as includes. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The comparison operations in the parallel `includes` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `includes` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `includes` algorithm returns true every element from the sorted range `[first2, last2)` is found within the sorted range `[first1, last1)`. Also returns true if `[first2, last2)` is empty.

hpx/parallel/container_algorithms/inclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

`template<typename InIter, typename Sent, typename OutIter>`
`inclusive_scan_result<InIter, OutIter> inclusive_scan (InIter first, Sent last, OutIter dest)`

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM($+, *first, \dots, *(first + (i - result))$).

The reduce operations in the parallel `inclusive_scan` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate `op`.

Template Parameters

- `InIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter`.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `dest`: Refers to the beginning of the destination range.

The difference between `exclusive_scan` and `inclusive_scan` is that `inclusive_scan` includes the i th input element in the i th sum.

Return The *inclusive_scan* algorithm returns `util::in_out_result<InIter, OutIter>`. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, inclusive_scan_result<FwdIter1, FwdIter2>>::type inclusive_scan(ExPolicy
&&policy,
FwdIter1
first,
Sent
last,
FwdIter2
dest)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, *first, ..., *(first + (i - result))).

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum.

Return The *inclusive_scan* algorithm returns a `hpx::future<util::in_out_result<FwdIter1, FwdIter2>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `util::in_out_result<FwdIter1, FwdIter2>` otherwise. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename Rng, typename O>
inclusive_scan_result<traits::range_iterator_t<Rng>, O> inclusive_scan(Rng &&rng, O dest)
Assigns through each iterator  $i$  in [result, result + (last - first)) the value of GENERAL-
IZED_NONCOMMUTATIVE_SUM(+, *first, ..., *(first + (i - result))).
```

The reduce operations in the parallel *inclusive_scan* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *O*: The type of the iterator representing the destination range (deduced).

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input element in the *i*th sum.

Return The *inclusive_scan* algorithm returns *util::in_out_result<traits::range_iterator_t<Rng>, O>*. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng, typename O>
util::detail::algorithm_result<ExPolicy, inclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type inclusive_scan
```

Assigns through each iterator i in [result, result + (last - first)) the value of GENERAL-
IZED_NONCOMMUTATIVE_SUM(+, *first, ..., *(first + (i - result))).

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *O*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.

The reduce operations in the parallel `inclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `exclusive_scan` and `inclusive_scan` is that `inclusive_scan` includes the i th input element in the i th sum.

Return The `inclusive_scan` algorithm returns a `hpx::future<util::in_out_result<traits::range_iterator_t<Rng>, O>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `util::in_out_result<traits::range_iterator_t<Rng>, O>` otherwise. The `inclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM($+, a_1, \dots, a_N$) is defined as:

- a_1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM($+, a_1, \dots, a_K$)
 - GENERALIZED_NONCOMMUTATIVE_SUM($+, a_M, \dots, a_N$) where $1 < K+1 = M \leq N$.

```
template<typename InIter, typename Sent, typename OutIter, typename Op>
inclusive_scan_result<InIter, OutIter> inclusive_scan(InIter first, Sent last, OutIter dest, Op
&&op)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM($op, *first, \dots, *(first + (i - result))$).

The reduce operations in the parallel `inclusive_scan` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate op .

Template Parameters

- `InIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter`.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `dest`: Refers to the beginning of the destination range.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input element in the *i*th sum.

Return The *inclusive_scan* algorithm returns *util::in_out_result<InIter, OutIter>*. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename Op>
util::detail::algorithm_result<ExPolicy, inclusive_scan_result<FwdIter1, FwdIter2>>::type inclusive_scan(ExPolicy
  &&policy,
  FwdIter1
  first,
  Sent
  last,
  FwdIter2
  dest,
  Op
  &&op)
```

Assigns through each iterator *i* in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(op, *first, ..., *(first + (i - result))).

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Op*: The type of the binary function object used for the reduction operation.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input element in the *i*th sum.

Return The *inclusive_scan* algorithm returns a *hpx::future<util::in_out_result<FwdIter1, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *util::in_out_result<FwdIter1, FwdIter2>* otherwise. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename Rng, typename O, typename Op>
inclusive_scan_result<traits::range_iterator_t<Rng>, O> inclusive_scan(Rng &&rng, O dest,
                                                               Op &&op)
```

Assigns through each iterator *i* in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(op, *first, ..., *(first + (i - result))).

The reduce operations in the parallel *inclusive_scan* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *O*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Op*: The type of the binary function object used for the reduction operation.

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the *i*th input element in the *i*th sum.

Return The *inclusive_scan* algorithm returns *util::in_out_result<traits::range_iterator_t<Rng>, O>*. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK)

- GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng, typename O, typename Op

```

```
util::detail::algorithm_result<ExPolicy, inclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type inclusive_scan
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM($op, *first, \dots, *(first + (i - result))$).

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate op .

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **O**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Op**: The type of the binary function object used for the reduction operation.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have **const&**, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum.

Return The *inclusive_scan* algorithm returns a *hpx::future<util::in_out_result<*traits*::range_iterator_t<**Rng**>, **O**>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *util::in_out_result<*traits*::range_iterator_t<**Rng**>, **O**>* otherwise. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:

- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK)
 - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where $1 < K+1 = M \leq N$.

```
template<typename InIter, typename Sent, typename OutIter, typename T, typename Op>
inclusive_scan_result<InIter, OutIter> inclusive_scan(InIter first, Sent last, OutIter dest, Op
&&op, T init)
Assigns through each iterator  $i$  in [result, result + (last - first)) the value of GENERAL-
IZED_NONCOMMUTATIVE_SUM(op, init, *first, ..., *(first + (i - result))).
```

The reduce operations in the parallel *inclusive_scan* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *InIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *OutIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Op*: The type of the binary function object used for the reduction operation.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- *init*: The initial value for the generalized sum.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum. If *op* is not mathematically associative, the behavior of *inclusive_scan* may be non-deterministic.

Return The *inclusive_scan* algorithm returns *util::in_out_result<InIter, OutIter>*. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(*op*, a1, ..., aN) is defined as:

- a1 when N is 1
- *op*(GENERALIZED_NONCOMMUTATIVE_SUM(*op*, a1, ..., aK), GENERAL-
IZED_NONCOMMUTATIVE_SUM(*op*, aM, ..., aN)) where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename T, typename Op>
```

```
util::detail::algorithm_result<ExPolicy, inclusive_scan_result<FwdIter1, FwdIter2>>::type inclusive_scan(ExPolicy  
    &&policy,  
    FwdIter1  
    first,  
    Sent  
    last,  
    FwdIter2  
    dest,  
    T  
    init,  
    Op  
    &&op)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM($op, init, *first, \dots, *(first + (i - result))$).

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate op .

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **T**: The type of the value to be used as initial (and intermediate) values (deduced).
- **Op**: The type of the binary function object used for the reduction operation.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **dest**: Refers to the beginning of the destination range.
- **init**: The initial value for the generalized sum.
- **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have **const&**, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input

element in the ith sum. If op is not mathematically associative, the behavior of $inclusive_scan$ may be non-deterministic.

Return The $inclusive_scan$ algorithm returns a $hpx::future<util::in_out_result<FwdIter1, FwdIter2>>$ if the execution policy is of type $sequenced_task_policy$ or $parallel_task_policy$ and returns $util::in_out_result<FwdIter1, FwdIter2>$ otherwise. The $inclusive_scan$ algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_N) is defined as:

- a_1 when N is 1
- $op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_K), GENERALIZED_NONCOMMUTATIVE_SUM(op, a_M, \dots, a_N))$ where $1 < K+1 = M \leq N$.

```
template<typename Rng, typename O, typename Op, typename T>
inclusive_scan_result<traits::range_iterator_t<Rng>, O> inclusive_scan(Rng &&rng, O dest,
Op &&op, T init)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of $GENERALIZED_NONCOMMUTATIVE_SUM(op, init, *first, \dots, *(first + (i - result)))$.

The reduce operations in the parallel $inclusive_scan$ algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate op .

Template Parameters

- Rng : The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- O : The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- Op : The type of the binary function object used for the reduction operation.
- T : The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- rng : Refers to the sequence of elements the algorithm will be applied to.
- $dest$: Refers to the beginning of the destination range.
- op : Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types $Type1$ and Ret must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- $init$: The initial value for the generalized sum.

The difference between $exclusive_scan$ and $inclusive_scan$ is that $inclusive_scan$ includes the i th input element in the i th sum. If op is not mathematically associative, the behavior of $inclusive_scan$ may be non-deterministic.

Return The $inclusive_scan$ algorithm returns $util::in_out_result<traits::range_iterator_t<Rng>, O>$. The $inclusive_scan$ algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_N) is defined as:

- a_1 when N is 1
- $op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_K), GENERALIZED_NONCOMMUTATIVE_SUM(op, a_M, \dots, a_N))$ where $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng, typename O, typename Op, typename T>
```

`util::detail::algorithm_result<ExPolicy, inclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type inclusive_sca`

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM($op, init, *first, \dots, *(first + (i - result))$).

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *O*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).
- *Op*: The type of the binary function object used for the reduction operation.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *init*: The initial value for the generalized sum.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the i th input element in the i th sum. If *op* is not mathematically associative, the behavior of *inclusive_scan* may be non-deterministic.

Return The *inclusive_scan* algorithm returns a `hpx::future<util::in_out_result<traits::range_iterator_t<Rng>, O>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `util::in_out_result<traits::range_iterator_t<Rng>, O>` otherwise. The *inclusive_scan* algorithm returns an input iterator to the point denoted by the

sentinel and an output iterator to the element in the destination range, one past the last element copied

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN) is defined as:

- a1 when N is 1
- op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN)) where $1 < K+1 = M \leq N$.

hpx/parallel/container_algorithms/is_heap.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename ExPolicy, typename Iter, typename Sent, typename Comp = detail::less, typename Proj = util::util::detail::algorithm_result<ExPolicy, bool>::type is_heap(ExPolicy &&policy, Iter first, Sent last, Comp &&comp = Comp(), Proj &&proj = Proj())
```

Returns whether the range is max heap. That is, true if the range is max heap, false otherwise. The function uses the given comparison function object *comp* (defaults to using operator<()).

comp has to induce a strict weak ordering on the values.

Note Complexity: Performs at most N applications of the comparison *comp*, at most $2 * N$ applications of the projection *proj*, where $N = \text{last} - \text{first}$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter**: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Comp**: The type of the function/function object to use (deduced).
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *iter*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *sent*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *comp*: *comp* is a callable object. The return value of the INVOKE operation applied to an object of type *Comp*, when contextually converted to *bool*, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *is_heap* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *is_heap* algorithm returns whether the range is max heap. That is, true if the range is max heap, false otherwise.

```
template<typename ExPolicy, typename Rng, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type is_heap(ExPolicy &&policy, Rng &&rng,
                                                               Comp &&comp = Comp(), Proj
                                                               &&proj = Proj())
```

Returns whether the range is max heap. That is, true if the range is max heap, false otherwise. The function uses the given comparison function object *comp* (defaults to using operator<()).

comp has to induce a strict weak ordering on the values.

Note Complexity: Performs at most N applications of the comparison *comp*, at most 2 * N applications of the projection *proj*, where N = last - first.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- **Comp**: The type of the function/function object to use (deduced).
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *comp*: *comp* is a callable object. The return value of the INVOKE operation applied to an object of type *Comp*, when contextually converted to *bool*, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *is_heap* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *is_heap* algorithm returns whether the range is max heap. That is, true if the range is max heap, false otherwise.

```
template<typename ExPolicy, typename Iter, typename Sent, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, Iter>::type is_heap_until(ExPolicy &&policy, Iter
                                                               first, Sent sent, Comp
                                                               &&comp = Comp(), Proj
                                                               &&proj = Proj())
```

Returns the upper bound of the largest range beginning at *first* which is a max heap. That is, the last iterator *it* for which range [*first*, *it*) is a max heap. The function uses the given comparison function object *comp* (defaults to using operator<()).

comp has to induce a strict weak ordering on the values.

Note Complexity: Performs at most N applications of the comparison *comp*, at most 2 * N applications of the projection *proj*, where N = last - first.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter**: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Comp**: The type of the function/function object to use (deduced).
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **iter**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **sent**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **comp**: *comp* is a callable object. The return value of the INVOKE operation applied to an object of type *Comp*, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *is_heap_until* algorithm returns a `hpx::future<RandIter>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *RandIter* otherwise. The *is_heap_until* algorithm returns the upper bound of the largest range beginning at *first* which is a max heap. That is, the last iterator *it* for which range [*first*, *it*) is a max heap.

```
template<typename ExPolicy, typename Rng, typename Comp = detail::less, typename Proj = util::projection_identity,
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type is_heap_until(ExPo
&&P
icy,
Rng
&&r
Com
&&c
=
Com
Proj
&&p
=
Proj
```

Returns the upper bound of the largest range beginning at *first* which is a max heap. That is, the last iterator *it* for which range [*first*, *it*) is a max heap. The function uses the given comparison function object *comp* (defaults to using operator<()).

comp has to induce a strict weak ordering on the values.

Note Complexity: Performs at most N applications of the comparison *comp*, at most 2 * N applications of the projection *proj*, where N = last - first.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes

the assignments.

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- **Comp**: The type of the function/function object to use (deduced).
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **comp**: *comp* is a callable object. The return value of the INVOKE operation applied to an object of type *Comp*, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *is_heap_until* algorithm returns a `hpx::future<RandIter>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *RandIter* otherwise. The *is_heap_until* algorithm returns the upper bound of the largest range beginning at first which is a max heap. That is, the last iterator *it* for which range [first, *it*) is a max heap.

hpx/parallel/container_algorithms/is_partitioned.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename FwdIter, typename Sent, typename Pred, typename Proj = hpx::parallel::util::projection_identity>
bool is_partitioned(FwdIter first, Sent last, Pred &&pred, Proj &&proj = Proj())
```

Determines if the range [first, last) is partitioned.

Note Complexity: at most (N) predicate evaluations where $N = \text{distance}(\text{first}, \text{last})$.

Return The *is_partitioned* algorithm returns *bool*. The *is_partitioned* algorithm returns true if each element in the sequence for which *pred* returns true precedes those for which *pred* returns false. Otherwise *is_partitioned* returns false. If the range [first, last) contains less than two elements, the function is always true.

Template Parameters

- **FwdIter**: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.

- Proj: The type of an optional projection function. This defaults to `hpx::parallel::util::projection_identity`.
- Pred: The type of the function/function object to use (deduced).

Parameters

- first: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- pred: Refers to the unary predicate which returns true for elements expected to be found in the beginning of the range. The signature of the function should be equivalent to

```
bool pred(const Type &a);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Pred, typename Proj = hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type is_partitioned(ExPolicy &&policy,
                                                               FwdIter first, Sent last,
                                                               Pred &&pred, Proj
                                                               &&proj = Proj())
```

Determines if the range [first, last) is partitioned.

The predicate operations in the parallel *is_partitioned* algorithm invoked with an execution policy object of type *sequenced_policy* executes in sequential order in the calling thread.

Note Complexity: at most (N) predicate evaluations where $N = \text{distance}(\text{first}, \text{last})$.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- Sent: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- Proj: The type of an optional projection function. This defaults to `hpx::parallel::util::projection_identity`.
- Pred: The type of the function/function object to use (deduced). `Pred` must be *CopyConstructible* when using a parallel policy.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- pred: Refers to the unary predicate which returns true for elements expected to be found in the beginning of the range. The signature of the function should be equivalent to

```
bool pred(const Type &a);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` invoked.

The comparison operations in the parallel `is_partitioned` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `is_partitioned` algorithm returns a `hpx::future<bool>` if the execution policy is of type `task_execution_policy` and returns `bool` otherwise. The `is_partitioned` algorithm returns true if each element in the sequence for which `pred` returns true precedes those for which `pred` returns false. Otherwise `is_partitioned` returns false. If the range [first, last) contains less than two elements, the function is always true.

```
template<typename Rng, typename Pred, typename Proj = hpx::parallel::util::projection_identity>
bool is_partitioned(Rng &&rng, Pred &&pred, Proj &&proj = Proj())
```

Determines if the range `rng` is partitioned.

Note Complexity: at most (N) predicate evaluations where $N = \text{std::size}(\text{rng})$.

Return The `is_partitioned` algorithm returns `bool`. The `is_partitioned` algorithm returns true if each element in the sequence for which `pred` returns true precedes those for which `pred` returns false. Otherwise `is_partitioned` returns false. If the range `rng` contains less than two elements, the function is always true.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Proj`: The type of an optional projection function. This defaults to `hpx::parallel::util::projection_identity`.
- `Pred`: The type of the function/function object to use (deduced).

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: Refers to the unary predicate which returns true for elements expected to be found in the beginning of the range. The signature of the function should be equivalent to

```
bool pred(const Type &a);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` invoked.

```
template<typename ExPolicy, typename Rng, typename Pred, typename Proj = hpx::parallel::util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type is_partitioned(ExPolicy &&policy, Rng
&&rng, Pred &&pred,
Proj &&proj = Proj())
```

Determines if the range [first, last) is partitioned.

The predicate operations in the parallel `is_partitioned` algorithm invoked with an execution policy object of type `sequenced_policy` executes in sequential order in the calling thread.

Note Complexity: at most (N) predicate evaluations where $N = \text{std::size}(\text{rng})$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Proj`: The type of an optional projection function. This defaults to `hpx::parallel::util::projection_identity`.
- `Pred`: The type of the function/function object to use (deduced). `Pred` must be `CopyConstructible` when using a parallel policy.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: Refers to the unary predicate which returns true for elements expected to be found in the beginning of the range. The signature of the function should be equivalent to

```
bool pred(const Type &a);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` is invoked.

The comparison operations in the parallel `is_partitioned` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `is_partitioned` algorithm returns a `hpx::future<bool>` if the execution policy is of type `task_execution_policy` and returns `bool` otherwise. The `is_partitioned` algorithm returns true if each element in the sequence for which `pred` returns true precedes those for which `pred` returns false. Otherwise `is_partitioned` returns false. If the range `rng` contains less than two elements, the function is always true.

hpx/parallel/container_algorithms/is_sorted.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

```
template<typename FwdIter, typename Sent, typename Pred = hpx::parallel::v1::detail::less, typename Proj = hpx::parallel::util::projection_identity>
bool is_sorted(FwdIter first, Sent last, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Determines if the range [first, last) is sorted. Uses `pred` to compare elements.

The comparison operations in the parallel `is_sorted` algorithm executes in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{distance}(\text{first}, \text{last})$. $S = \text{number of partitions}$

Template Parameters

- `FwdIter`: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- `Pred`: The type of an optional function/function object to use.

- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `first`: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- `pred`: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` is invoked.

Return The `is_sorted` algorithm returns a `bool`. The `is_sorted` algorithm returns true if each element in the sequence $[first, last)$ satisfies the predicate passed. If the range $[first, last)$ contains less than two elements, the function always returns true.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Pred = hpx::parallel::v1::detail::less, typename hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type is_sorted(ExPolicy &&policy, FwdIter first, Sent last, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Determines if the range $[first, last)$ is sorted. Uses `pred` to compare elements.

The comparison operations in the parallel `is_sorted` algorithm invoked with an execution policy object of type `sequenced_policy` executes in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{distance}(first, last)$. $S = \text{number of partitions}$

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `is_sorted` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- `pred`: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects

passed to it. The type *Type* must be such that objects of types *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *is_sorted* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *is_sorted* algorithm returns a *hpx::future<bool>* if the execution policy is of type *task_execution_policy* and returns *bool* otherwise. The *is_sorted* algorithm returns a *bool* if each element in the sequence [first, last) satisfies the predicate passed. If the range [first, last) contains less than two elements, the function always returns true.

```
template<typename Rng, typename Pred = hpx::parallel::v1::detail::less, typename Proj = hpx::parallel::util::projection>
bool is_sorted(ExPolicy &&policy, Rng &&rng, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Determines if the range *rng* is sorted. Uses *pred* to compare elements.

The comparison operations in the parallel *is_sorted* algorithm executes in sequential order in the calling thread.

Note Complexity: at most ($N+S-1$) comparisons where $N = \text{size}(\text{rng})$. $S = \text{number of partitions}$

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of an optional function/function object to use.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The type *Type* must be such that objects of types *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *is_sorted* algorithm returns a *bool*. The *is_sorted* algorithm returns true if each element in the *rng* satisfies the predicate passed. If the range *rng* contains less than two elements, the function always returns true.

```
template<typename ExPolicy, typename Rng, typename Pred = hpx::parallel::v1::detail::less, typename Proj = hpx::parallel::util::projection>
hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type is_sorted(ExPolicy &&policy, Rng &&rng, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Determines if the range *rng* is sorted. Uses *pred* to compare elements.

The comparison operations in the parallel *is_sorted* algorithm invoked with an execution policy object of type *sequenced_policy* executes in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{size}(\text{rng})$. $S = \text{number of partitions}$

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *is_sorted* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **pred**: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type *Type* must be such that objects of types *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *is_sorted* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *is_sorted* algorithm returns a `hpx::future<bool>` if the execution policy is of type *task_execution_policy* and returns `bool` otherwise. The *is_sorted* algorithm returns a `bool` if each element in the range *rng* satisfies the predicate passed. If the range *rng* contains less than two elements, the function always returns true.

```
template<typename FwdIter, typename Sent, typename Pred = hpx::parallel::v1::detail::less, typename Proj = hpx::parallel::v1::detail::projection_identity>
FwdIter is_sorted_until(FwdIter first, Sent last, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Returns the first element in the range [first, last) that is not sorted. Uses a predicate to compare elements or the less than operator.

The comparison operations in the parallel *is_sorted_until* algorithm execute in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{distance}(\text{first}, \text{last})$. $S = \text{number of partitions}$

Template Parameters

- **FwdIter**: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- **Pred**: The type of an optional function/function object to use.
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **first**: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- **pred**: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` is invoked.

Return The `is_sorted_until` algorithm returns a `FwdIter`. The `is_sorted_until` algorithm returns the first unsorted element. If the sequence has less than two elements or the sequence is sorted, last is returned.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Pred = hpx::parallel::v1::detail::less, type hpx::parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type is_sorted_until(ExPolicy &&policy, FwdIter first, Sent last, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Returns the first element in the range [first, last) that is not sorted. Uses a predicate to compare elements or the less than operator.

The comparison operations in the parallel `is_sorted_until` algorithm invoked with an execution policy object of type `sequenced_policy` executes in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{distance}(\text{first}, \text{last})$. $S = \text{number of partitions}$

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `is_sorted_until` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- `pred`: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel `is_sorted_until` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `is_sorted_until` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `is_sorted_until` algorithm returns the first unsorted element. If the sequence has less than two elements or the sequence is sorted, last is returned.

```
template<typename Rng, typename Pred = hpx::parallel::v1::detail::less, typename Proj = hpx::parallel::util::projection<hpx::traits::range_iterator<Rng>::type> is_sorted_until(ExPolicy &&policy, Rng &&rng,  
                                              Pred &&pred = Pred(), Proj &&proj  
                                              = Proj())
```

Returns the first element in the range `rng` that is not sorted. Uses a predicate to compare elements or the less than operator.

The comparison operations in the parallel `is_sorted_until` algorithm execute in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{size}(\text{rng})$. $S = \text{number of partitions}$

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Pred`: The type of an optional function/function object to use.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The `is_sorted_until` algorithm returns a `FwdIter`. The `is_sorted_until` algorithm returns the first unsorted element. If the sequence has less than two elements or the sequence is sorted, last is returned.

```
template<typename ExPolicy, typename Rng, typename Pred = hpx::parallel::v1::detail::less, typename Proj = hpx::parallel::util::projection<hpx::traits::range_iterator<Rng>::type> is_sorted_until(ExPolicy &&policy, Rng &&rng,  
                                              Pred &&pred = Pred(), Proj &&proj  
                                              = Proj())
```

`hpx::parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type` **is_sorted_until**

Returns the first element in the range `rng` that is not sorted. Uses a predicate to compare elements or the less than operator.

The comparison operations in the parallel `is_sorted_until` algorithm invoked with an execution policy object of type `sequenced_policy` executes in sequential order in the calling thread.

Note Complexity: at most $(N+S-1)$ comparisons where $N = \text{size}(rng)$. $S = \text{number of partitions}$

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `is_sorted_until` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` is invoked.

The comparison operations in the parallel `is_sorted_until` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `is_sorted_until` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `is_sorted_until` algorithm returns the first unsorted element. If the sequence has less than two elements or the sequence is sorted, last is returned.

hpx/parallel/container_algorithms/lexicographical_compare.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename InIter1, typename Sent1, typename InIter2, typename Sent2, typename Proj1 = hpx::parallel::sequenced\_policy
bool lexicographical_compare(InIter1 first1, Sent1 last1, InIter2 first2, Sent2 last2, Pred
                           &&pred = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2
                           = Proj2())
```

Checks if the first range [first1, last1) is lexicographically less than the second range [first2, last2).
uses a provided predicate to compare elements.

The comparison operations in the parallel *lexicographical_compare* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most $2 * \min(N1, N2)$ applications of the comparison operation, where $N1 = \text{std}::\text{distance}(\text{first1}, \text{last1})$ and $N2 = \text{std}::\text{distance}(\text{first2}, \text{last2})$.

Template Parameters

- **InIter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an input iterator.
- **Sent1**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **InIter1**.
- **InIter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an input iterator.
- **Sent2**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **InIter2**.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *lexicographical_compare* requires **Pred** to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- **Proj1**: The type of an optional projection function for **FwdIter1**. This defaults to `util::projection_identity`
- **Proj2**: The type of an optional projection function for **FwdIter2**. This defaults to `util::projection_identity`

Parameters

- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **pred**: Refers to the comparison function that the first and second ranges will be applied to
- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate is invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate is invoked.

The comparison operations in the parallel *lexicographical_compare* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note Lexicographical comparison is an operation with the following properties

- Two ranges are compared element by element
- The first mismatching element defines which range is lexicographically *less* or *greater* than the other
- If one range is a prefix of another, the shorter range is lexicographically *less* than the other
- If two ranges have equivalent elements and are of the same length, then the ranges are lexicographically *equal*
- An empty range is lexicographically *less* than any non-empty range
- Two empty ranges are lexicographically *equal*

Return The *lexicographically_compare* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *lexicographically_compare* algorithm returns true if the first range is lexicographically less, otherwise it returns false. range [first2, last2], it returns false.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2, typename  
parallel::util::detail::algorithm_result<ExPolicy, bool>::type lexicographical_compare (ExPolicy  
    &&pol-  
    icy,  
    FwdIter1  
    first1,  
    Sent1  
    last1,  
    FwdIter2  
    first2,  
    Sent2  
    last2,  
    Pred  
    &&pred  
    =  
    Pred(),  
    Proj1  
    &&proj1  
    =  
    Proj1(),  
    Proj2  
    &&proj2  
    =  
    Proj2())
```

Checks if the first range [first1, last1) is lexicographically less than the second range [first2, last2). uses a provided predicate to compare elements.

The comparison operations in the parallel *lexicographical_compare* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most $2 * \min(N1, N2)$ applications of the comparison operation, where $N1 = \text{std}::\text{distance}(\text{first1}, \text{last})$ and $N2 = \text{std}::\text{distance}(\text{first2}, \text{last2})$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- `FwdIter1`: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent1`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter1`.
- `FwdIter2`: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent2`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter2`.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `lexicographical_compare` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj1`: The type of an optional projection function for `FwdIter1`. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function for `FwdIter2`. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `first2`: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- `last2`: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- `pred`: Refers to the comparison function that the first and second ranges will be applied to
- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel `lexicographical_compare` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note Lexicographical comparison is an operation with the following properties

- Two ranges are compared element by element
- The first mismatching element defines which range is lexicographically *less* or *greater* than the other
- If one range is a prefix of another, the shorter range is lexicographically *less* than the other
- If two ranges have equivalent elements and are of the same length, then the ranges are lexicographically *equal*
- An empty range is lexicographically *less* than any non-empty range
- Two empty ranges are lexicographically *equal*

Return The `lexicographically_compare` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `lexicographically_compare` algorithm returns true if the first range is lexicographically less, otherwise it returns false. range `[first2, last2]`, it returns false.

```
template<typename Rng1, typename Rng2, typename Proj1 = hpx::parallel::util::projection_identity, typename Proj2 = hpx::parallel::util::projection_identity>
bool lexicographical_compare(Rng1 &&rng1, Rng2 &&rng2, Pred &&pred = Pred(),
                           Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Checks if the first range `rng1` is lexicographically less than the second range `rng2`. uses a provided predicate to compare elements.

The comparison operations in the parallel *lexicographical_compare* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: At most $2 * \min(N1, N2)$ applications of the comparison operation, where $N1 = \text{std}::\text{distance}(\text{std}::\text{begin}(\text{rng1}), \text{std}::\text{end}(\text{rng1}))$ and $N2 = \text{std}::\text{distance}(\text{std}::\text{begin}(\text{rng2}), \text{std}::\text{end}(\text{rng2}))$.

Template Parameters

- **Rng1:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Rng2:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Pred:** The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *lexicographical_compare* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- **Proj1:** The type of an optional projection function for elements of the first range. This defaults to `util::projection_identity`
- **Proj2:** The type of an optional projection function for elements of the second range. This defaults to `util::projection_identity`

Parameters

- **rng1:** Refers to the sequence of elements the algorithm will be applied to.
- **rng2:** Refers to the sequence of elements the algorithm will be applied to.
- **pred:** Refers to the comparison function that the first and second ranges will be applied to
- **proj1:** Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
- **proj2:** Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

Note Lexicographical comparison is an operation with the following properties

- Two ranges are compared element by element
- The first mismatching element defines which range is lexicographically *less* or *greater* than the other
- If one range is a prefix of another, the shorter range is lexicographically *less* than the other
- If two ranges have equivalent elements and are of the same length, then the ranges are lexicographically *equal*
- An empty range is lexicographically *less* than any non-empty range
- Two empty ranges are lexicographically *equal*

Return The *lexicographically_compare* algorithm returns *bool*. The *lexicographically_compare* algorithm returns true if the first range is lexicographically less, otherwise it returns false. range [first2, last2), it returns false.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Proj1 = hpx::parallel::util::projection_iden
```

```
parallel::util::detail::algorithm_result<ExPolicy, bool>::type lexicographical_compare (ExPolicy
&&pol-
icy,
Rng1
&&rng1,
Rng2
&&rng2,
Pred
&&pred
=
Pred(),
Proj1
&&proj1
=
Proj1(),
Proj2
&&proj2
=
Proj2())
```

Checks if the first range `rng1` is lexicographically less than the second range `rng2`. uses a provided predicate to compare elements.

The comparison operations in the parallel *lexicographical_compare* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most $2 * \min(N1, N2)$ applications of the comparison operation, where $N1 = \text{std}::distance(\text{std}::begin(rng1), \text{std}::end(rng1))$ and $N2 = \text{std}::distance(\text{std}::begin(rng2), \text{std}::end(rng2))$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Rng2`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *lexicographical_compare* requires `Pred` to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- `Proj1`: The type of an optional projection function for elements of the first range. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function for elements of the second range. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the sequence of elements the algorithm will be applied to.
- `rng2`: Refers to the sequence of elements the algorithm will be applied to.
- `pred`: Refers to the comparison function that the first and second ranges will be applied to
- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *lexicographical_compare* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an

unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note Lexicographical comparison is an operation with the following properties

- Two ranges are compared element by element
- The first mismatching element defines which range is lexicographically *less* or *greater* than the other
- If one range is a prefix of another, the shorter range is lexicographically *less* than the other
- If two ranges have equivalent elements and are of the same length, then the ranges are lexicographically *equal*
- An empty range is lexicographically *less* than any non-empty range
- Two empty ranges are lexicographically *equal*

Return The *lexicographically_compare* algorithm returns a `hpx::future<bool>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `bool` otherwise. The *lexicographically_compare* algorithm returns true if the first range is lexicographically less, otherwise it returns false. range [first2, last2), it returns false.

`hpx/parallel/container_algorithms/make_heap.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

```
template<typename ExPolicy, typename Rng, typename Comp, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type make_heap (ExPolicy
&&policy,
Rng
&&rng,
Comp
&&comp,
Proj
&&proj =
Proj{})
```

Constructs a *max heap* in the range [first, last).

The predicate operations in the parallel *make_heap* algorithm invoked with an execution policy object of type *sequential_execution_policy* executes in sequential order in the calling thread.

Note Complexity: at most (3*N) comparisons where $N = \text{distance}(\text{first}, \text{last})$.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Comp:** The type of the function/function object to use (deduced).

- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `comp`: Refers to the binary predicate which returns true if the first argument should be treated as less than the second. The signature of the function should be equivalent to

```
bool comp(const Type &a, const Type &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that objects of types `RndIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The comparison operations in the parallel `make_heap` algorithm invoked with an execution policy object of type `parallel_execution_policy` or `parallel_task_execution_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `make_heap` algorithm returns a `hpx::future<Iter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `Iter` otherwise. It returns `last`.

```
template<typename ExPolicy, typename Rng, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type make_heap(ExPolicy
&&policy,
Rng
&&rng,
Proj
&&proj
=
Proj{})
```

Constructs a *max heap* in the range [first, last). Uses the operator `<` for comparisons.

The predicate operations in the parallel `make_heap` algorithm invoked with an execution policy object of type `sequential_execution_policy` executes in sequential order in the calling thread.

Note Complexity: at most $(3*N)$ comparisons where $N = \text{distance}(\text{first}, \text{last})$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The comparison operations in the parallel `make_heap` algorithm invoked with an execution policy object of type `parallel_execution_policy` or `parallel_task_execution_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `make_heap` algorithm returns a `hpx::future<void>` if the execution policy is of type `task_execution_policy` and returns `void` otherwise.

hpx/parallel/container_algorithms/merge.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3, typename Proj1, typename Proj2, typename Comp>
util::detail::algorithm_result<ExPolicy, hpx::ranges::merge_result<Iter1, Iter2, Iter3>>::type merge(ExPolicy &&policy, Iter1 first1, Sent1 last1, Iter2 first2, Sent2 last2, Iter3 dest, Comp &&comp = Comp{}, Proj1 &&proj1 = Proj1{}, Proj2 &&proj2 = Proj2{})
```

Merges two sorted ranges [first1, last1) and [first2, last2) into one sorted range beginning at *dest*. The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range. The destination range cannot overlap with either of the input ranges.

The assignments in the parallel *merge* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs $O(\text{std}::\text{distance}(\text{first1}, \text{last1}) + \text{std}::\text{distance}(\text{first2}, \text{last2}))$ applications of the comparison *comp* and the each projection.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter1:** The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an random access iterator.

- `Sent1`: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for `Iter1`.
- `Iter2`: The type of the source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an random access iterator.
- `Sent2`: The type of the end source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an sentinel for `Iter2`.
- `Iter3`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an random access iterator.
- `Comp`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `merge` requires `Comp` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj1`: The type of an optional projection function to be used for elements of the first range. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function to be used for elements of the second range. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `first2`: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- `last2`: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `comp`: `comp` is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

```
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `RandIter1` and `RandIter2` can be dereferenced and then implicitly converted to both `Type1` and `Type2`

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual comparison `comp` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual comparison `comp` is invoked.

The assignments in the parallel `merge` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `merge` algorithm returns a `hpx::future<merge_result<Iter1, Iter2, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `merge_result<Iter1, Iter2, Iter3>` otherwise. The `merge` algorithm returns the tuple of the source iterator `last1`, the source iterator `last2`, the destination iterator to the end of the `dest` range.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename RandIter3, typename Comp = hpx::range
```

`util::detail::algorithm_result<ExPolicy, hpx::ranges::merge_result<typename hpx::traits::range_iterator<Rng1>::type, typename hpx::traits::range_iterator<Rng2>::type, Compare, Proj1, Proj2>`

Merges two sorted ranges [first1, last1) and [first2, last2) into one sorted range beginning at *dest*. The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range. The destination range cannot overlap with either of the input ranges.

The assignments in the parallel *merge* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs $O(\text{std::distance(first1, last1)} + \text{std::distance(first2, last2)})$ applications of the comparison *comp* and the each projection.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the first source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- *Rng2*: The type of the second source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- *RandIter3*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an random access iterator.
- *Comp*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *merge* requires *Comp* to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- *Proj1*: The type of an optional projection function to be used for elements of the first range. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function to be used for elements of the second range. This defaults to `util::projection_identity`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the first range of elements the algorithm will be applied to.
- *rng2*: Refers to the second range of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.

- `comp`: *comp* is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

```
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *RandIter1* and *RandIter2* can be dereferenced and then implicitly converted to both *Type1* and *Type2*.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual comparison *comp* is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual comparison *comp* is invoked.

The assignments in the parallel *merge* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *merge* algorithm returns a *hpx::future<merge_result<RandIter1, RandIter2, RandIter3>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *merge_result<RandIter1, RandIter2, RandIter3>* otherwise. The *merge* algorithm returns the tuple of the source iterator *last1*, the source iterator *last2*, the destination iterator to the end of the *dest* range.

```
template<typename ExPolicy, typename Iter, typename Sent, typename Comp = hpx::ranges::less, typename Proj = util::detail::algorithm_result<ExPolicy, Iter>::type> inline merge(ExPolicy &&policy, Iter first, Iter middle, Sent last,
                                                               Comp &&comp = Comp(),
                                                               Proj &&proj = Proj())
```

Merges two consecutive sorted ranges [first, middle) and [middle, last) into one sorted range [first, last). The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range.

The assignments in the parallel *inplace_merge* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs O(*std::distance(first, last)*) applications of the comparison *comp* and the each projection.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Iter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random access iterator.
- `Sent`: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for `Iter1`.
- `Comp`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *inplace_merge* requires `Comp` to meet the requirements of *CopyConstructible*. This defaults to *std::less<>*
- `Proj`: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the first sorted range the algorithm will be applied to.
- `middle`: Refers to the end of the first sorted range and the beginning of the second sorted

range the algorithm will be applied to.

- *last*: Refers to the end of the second sorted range the algorithm will be applied to.
- *comp*: *comp* is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

```
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *RandIter* can be dereferenced and then implicitly converted to both *Type1* and *Type2*

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *inplace_merge* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *inplace_merge* algorithm returns a *hpx::future<Iter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *Iter* otherwise. The *inplace_merge* algorithm returns the source iterator *last*

```
template<typename ExPolicy, typename Rng, typename RandIter, typename Comp = hpx::ranges::less, typename Proj = util::detail::algorithm_result<ExPolicy, RandIter>::type> inline void inplace_merge(ExPolicy &&policy, Rng &&rng, RandIter middle, Comp &&comp = Comp{}, Proj &&proj = Proj{})
```

Merges two consecutive sorted ranges [first, middle) and [middle, last) into one sorted range [first, last). The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range.

The assignments in the parallel *inplace_merge* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs O(*std::distance(first, last)*) applications of the comparison *comp* and the each projection.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- *RandIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random access iterator.
- *Comp*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *inplace_merge* requires *Comp* to meet the requirements of *CopyConstructible*. This defaults to *std::less<>*
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the range of elements the algorithm will be applied to.
- *middle*: Refers to the end of the first sorted range and the beginning of the second sorted range the algorithm will be applied to.
- *comp*: *comp* is a callable object which returns true if the first argument is less than the second,

and false otherwise. The signature of this comparison should be equivalent to:

```
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `RandIter` can be dereferenced and then implicitly converted to both `Type1` and `Type2`

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel `inplace_merge` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `inplace_merge` algorithm returns a `hpx::future<RandIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `RandIter` otherwise. The `inplace_merge` algorithm returns the source iterator `last`

hpx/parallel/container_algorithms/minmax.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

```
template<typename FwdIter, typename Sent, typename F, typename Proj = hpx::parallel::util::projection_identity>
FwdIter min_element(FwdIter first, Sent last, F &&f = F(), Proj &&proj = Proj())
```

Finds the smallest element in the range [first, last) using the given comparison function *f*.

The comparisons in the parallel `min_element` algorithm execute in sequential order in the calling thread.

Note Complexity: Exactly $\max(N-1, 0)$ comparisons, where $N = \text{std}:\text{distance}(\text{first}, \text{last})$.

Template Parameters

- `FwdIter`: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the end iterators used (deduced). This sentinel type must be a sentinel for `FwdIter`.
- `F`: The type of the function/function object to use (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `f`: The binary predicate which returns true if the the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *min_element* algorithm returns *FwdIter*. The *min_element* algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename Rng, typename F, typename Proj = hpx::parallel::util::projection_identity>
hpx::traits::range_iterator_t<Rng> min_element (Rng &&rng, F &&f = F(), Proj &&proj = Proj())
    Finds the smallest element in the range [first, last) using the given comparison function f.
```

The comparisons in the parallel *min_element* algorithm execute in sequential order in the calling thread.

Note Complexity: Exactly *max(N-1, 0)* comparisons, where *N* = std::distance(first, last).

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *F*: The type of the function/function object to use (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: The binary predicate which returns true if the the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *min_element* algorithm returns a *hpx::traits::range_iterator<Rng>::type* otherwise. The *min_element* algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename F, typename Proj = hpx::parallel::util::projection_identity>
hpx::parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type min_element (ExPolicy &&policy,
    FwdIter first, Sent
    sent, F &&f = F(),
    Proj &&proj = Proj())
    Finds the smallest element in the range [first, last) using the given comparison function f.
```

The comparisons in the parallel *min_element* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly $\max(N-1, 0)$ comparisons, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for **FwdIter**.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *min_element* requires **F** to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: The binary predicate which returns true if the the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have **const &**, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparisons in the parallel *min_element* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *min_element* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *min_element* algorithm returns the iterator to the smallest element in the range [first, last]. If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

template<typename **ExPolicy**, typename **Rng**, typename **F**, typename **Proj** = *hpx::parallel::util::projection_identity*>

```
util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator_t<Rng>>::type min_element (ExPolicy
    &&policy,
    Rng
    &&rng,
    F
    &&f
    =
    F(),
    Proj
    &&proj
    =
    Proj())
```

Finds the smallest element in the range [first, last) using the given comparison function *f*.

The comparisons in the parallel *min_element* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly $\max(N-1, 0)$ comparisons, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *min_element* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: The binary predicate which returns true if the the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparisons in the parallel *min_element* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *min_element* algorithm returns a *hpx::future<hpx::traits::range_iterator<Rng>::type>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter*

otherwise. The *min_element* algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename FwdIter, typename Sent, typename F, typename Proj = hpx::parallel::util::projection_identity>
FwdIter max_element (FwdIter first, Sent sent, F &&f = F(), Proj &&proj = Proj())
```

Finds the greatest element in the range [first, last) using the given comparison function *f*.

The comparisons in the parallel *max_element* algorithm execute in sequential order in the calling thread.

Note Complexity: Exactly $\max(N-1, 0)$ comparisons, where $N = \text{std::distance(first, last)}$.

Template Parameters

- *FwdIter*: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *F*: The type of the function/function object to use (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *f*: The binary predicate which returns true if the This argument is optional and defaults to *std::less*. The left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *max_element* algorithm returns a *FwdIter*. The *max_element* algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename Rng, typename F, typename Proj = hpx::parallel::util::projection_identity>
hpx::traits::range_iterator_t<Rng> max_element (Rng &&rng, F &&f = F(), Proj &&proj = Proj())
```

Finds the greatest element in the range [first, last) using the given comparison function *f*.

The comparisons in the parallel *max_element* algorithm execute in sequential order in the calling thread.

Note Complexity: Exactly $\max(N-1, 0)$ comparisons, where $N = \text{std::distance(first, last)}$.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

- *F*: The type of the function/function object to use (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: The binary predicate which returns true if the This argument is optional and defaults to *std::less*. The left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *max_element* algorithm returns a *hpx::traits::range_iterator<Rng>::type* otherwise. The *max_element* algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename F, typename Proj = hpx::parallel::util::projection_identity<ExPolicy, FwdIter>>::type max_element(ExPolicy &&policy,
FwdIter first, Sent sent, F &&f = F(), Proj &&proj = Proj())
```

Finds the greatest element in the range [first, last) using the given comparison function *f*.

The comparisons in the parallel *max_element* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly *max(N-1, 0)* comparisons, where *N* = *std::distance(first, last)*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *max_element* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.

- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `f`: The binary predicate which returns true if the This argument is optional and defaults to `std::less`. The left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparisons in the parallel `max_element` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `max_element` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `max_element` algorithm returns the iterator to the smallest element in the range [first, last]. If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = hpx::parallel::util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator_t<Rng>>::type max_element(ExPolicy
    &&pol-
    icy,
    Rng
    &&rng,
    F
    &&f
    =
    F(),
    Proj
    &&proj
    =
    Proj())
```

Finds the greatest element in the range [first, last] using the given comparison function *f*.

The comparisons in the parallel `max_element` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Exactly $\max(N-1, 0)$ comparisons, where $N = \text{std::distance(first, last)}$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `max_element` requires *F* to meet the requirements of `CopyConstructible`.

- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **f**: The binary predicate which returns true if the This argument is optional and defaults to `std::less`. The left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparisons in the parallel `max_element` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `max_element` algorithm returns a `hpx::future<hpx::traits::range_iterator<Rng>::type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `max_element` algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template<typename FwdIter, typename Sent, typename F, typename Proj = hpx::parallel::util::projection_identity>
minmax_element_result<FwdIter, FwdIter> minmax_element(FwdIter first, Sent last, F &&f = F(),
                                                       Proj &&proj = Proj())
```

Finds the greatest element in the range [first, last) using the given comparison function *f*.

The assignments in the parallel `minmax_element` algorithm execute in sequential order in the calling thread.

Note Complexity: At most `max(floor(3/2*(N-1)), 0)` applications of the predicate, where `N = std::distance(first, last)`.

Template Parameters

- **FwdIter**: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for `FwdIter`.
- **F**: The type of the function/function object to use (deduced).
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

- *f*: The binary predicate which returns true if the left argument is less than the right element. This argument is optional and defaults to std::less. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *minmax_element* algorithm returns a *minmax_element_result*<*FwdIter*, *FwdIter*>. The *minmax_element* algorithm returns a *min_max_result* consisting of an iterator to the smallest element as the *min* element and an iterator to the greatest element as the *max* element. Returns *minmax_element_result*{*first*, *first*} if the range is empty. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent to the largest element, the iterator to the last such element is returned.

```
template<typename Rng, typename F, typename Proj = hpx::parallel::util::projection_identity>
minmax_element_result<hpx::traits::range_iterator_t<Rng>, hpx::traits::range_iterator_t<Rng>> minmax_element(Rng &&rng,
&&F =
= F(),
Proj &&proj =
Proj()
```

Finds the greatest element in the range [*first*, *last*) using the given comparison function *f*.

The assignments in the parallel *minmax_element* algorithm execute in sequential order in the calling thread.

Note Complexity: At most $\max(\text{floor}(3/2*(N-1)), 0)$ applications of the predicate, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *F*: The type of the function/function object to use (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: The binary predicate which returns true if the left argument is less than the right element. This argument is optional and defaults to std::less. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` invoked.

Return The `minmax_element` algorithm returns a `minmax_element_result<hpx::traits::range_iterator<Rng>::type`, `hpx::traits::range_iterator<Rng>::type`. The `minmax_element` algorithm returns a `min_max_result` consisting of an range iterator to the smallest element as the min element and an range iterator to the greatest element as the max element. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent to the largest element, the iterator to the last such element is returned.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename F, typename Proj = hpx::parallel::util::projection<ExPolicy, minmax_element_result<FwdIter, FwdIter>>::type minmax_element(ExPolicy &&policy, FwdIter first, Sent last, F &&f, Proj &&p = F(), ProjProj() = Proj()
```

Finds the greatest element in the range [first, last) using the given comparison function `f`.

The comparisons in the parallel `minmax_element` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: At most `max(floor(3/2*(N-1)), 0)` applications of the predicate, where `N = std::distance(first, last)`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the end iterators used (deduced). This sentinel type must be a sentinel for `FwdIter`.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `minmax_element` requires `F` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: The binary predicate which returns true if the left argument is less than the right element. This argument is optional and defaults to `std::less`. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *f* is invoked.

The comparisons in the parallel `minmax_element` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `minmax_element` algorithm returns a `minmax_element_result<FwdIter, FwdIter>`. The `minmax_element` algorithm returns a `min_max_result` consisting of an iterator to the smallest element as the `min` element and an iterator to the greatest element as the `max` element. Returns `minmax_element_result{first, first}` if the range is empty. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent to the largest element, the iterator to the last such element is returned.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = hpx::parallel::util::projection_identity>
util::detail::algorithm_result<ExPolicy, minmax_element_result<hpx::traits::range_iterator_t<Rng>, hpx::traits::range_iterator_t<
```

Finds the greatest element in the range [first, last) using the given comparison function *f*.

The comparisons in the parallel `minmax_element` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: At most $\max(\text{floor}(3/2*(N-1)), 0)$ applications of the predicate, where $N = \text{std::distance(first, last)}$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *minmax_element* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *f*: The binary predicate which returns true if the the left argument is less than the right element. This argument is optional and defaults to std::less. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const &*, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *FwdIter* can be dereferenced and then implicitly converted to *Type1*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The comparisons in the parallel *minmax_element* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *minmax_element* algorithm returns a *minmax_element_result<hpx::traits::range_iterator<Rng>::type, hpx::traits::range_iterator<Rng>::type>*. The *minmax_element* algorithm returns a *min_max_result* consisting of an range iterator to the smallest element as the min element and an range iterator to the greatest element as the max element. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent to the largest element, the iterator to the last such element is returned.

hpx/parallel/container_algorithms/mismatch.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Pred,
util::detail::algorithm_result<ExPolicy, ranges::mismatch_result<FwdIter1, FwdIter2>>::type mismatch(ExPolicy
&&policy,
FwdIter1
first1,
FwdIter1
last1,
FwdIter2
first2,
FwdIter2
last2,
Pred
&&op
=
Pred(),
Proj1
&&proj1
=
Proj1(),
Proj2
&&proj2
=
Proj2())
=
```

Returns true if the range [first1, last1) is mismatch to the range [first2, last2), and false otherwise.

The comparison operations in the parallel *mismatch* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most $\min(\text{last1} - \text{first1}, \text{last2} - \text{first2})$ applications of the predicate *f*. If *FwdIter1* and *FwdIter2* meet the requirements of *RandomAccessIterator* and $(\text{last1} - \text{first1}) \neq (\text{last2} - \text{first2})$ then no applications of the predicate *f* are made.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter1:** The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent1:** The type of the source iterators used for the end of the first range (deduced).
- **Iter2:** The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent2:** The type of the source iterators used for the end of the second range (deduced).
- **Pred:** The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *mismatch* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- **Proj1:** The type of an optional projection function applied to the first range. This defaults to `util::projection_identity`
- **Proj2:** The type of an optional projection function applied to the second range. This defaults to `util::projection_identity`

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **first1:** Refers to the beginning of the sequence of elements of the first range the algorithm

will be applied to.

- `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `first2`: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- `last2`: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- `op`: The binary predicate which returns true if the elements should be treated as mismatch. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *mismatch* algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The two ranges are considered mismatch if, for every iterator `i` in the range `[first1, last1)`, `*i mismatchs *(first2 + (i - first1))`. This overload of *mismatch* uses operator`==` to determine if two elements are mismatch.

Return The *mismatch* algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The *mismatch* algorithm returns true if the elements in the two ranges are mismatch, otherwise it returns false. If the length of the range `[first1, last1)` does not mismatch the length of the range `[first2, last2)`, it returns false.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = ranges::equal_to, typename Proj1 = util::detail::algorithm_result<ExPolicy, ranges::mismatch_result<FwdIter1, FwdIter2>>::type mismatch(ExPolicy &&pol-  
ic平,  
Rng1 &&rng1,  
Rng2 &&rng2,  
Pred &&op-  
= Pred(),  
Proj1 &&proj1  
= Proj1(),  
Proj2 &&proj2  
= Proj2())
```

Returns std::pair with iterators to the first two non-equivalent elements.

The comparison operations in the parallel *mismatch* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most *last1 - first1* applications of the predicate *f*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the first source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Rng2*: The type of the second source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *mismatch* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj1*: The type of an optional projection function applied to the first range. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function applied to the second range. This defaults to `util::projection_identity`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the first sequence of elements the algorithm will be applied to.
- *rng2*: Refers to the second sequence of elements the algorithm will be applied to.
- *op*: The binary predicate which returns true if the elements should be treated as mismatch. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and *FwdIter2* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *mismatch* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *mismatch* algorithm returns a `hpx::future<std::pair<FwdIter1, FwdIter2>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `std::pair<FwdIter1, FwdIter2>` otherwise. The *mismatch* algorithm returns the first mismatching pair of elements from two ranges: one defined by $[first1, last1)$ and another defined by $[first2, last2)$.

hpx/parallel/container_algorithms/move.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter>
util::detail::algorithm_result<ExPolicy, ranges::move_result<FwdIter1, FwdIter>>::type move (ExPolicy
&&policy,
FwdIter1
iter,
Sent1
sent,
FwdIter
dest)
```

Moves the elements in the range *rng* to another range beginning at *dest*. After this operation the elements in the moved-from range will still contain valid values of the appropriate type, but not necessarily the same values as before the move.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly std::distance(begin(rng), end(rng)) assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent1*: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for *FwdIter1*.
- *FwdIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *move* algorithm returns a *hpx::future<ranges::move_result<iterator_t<Rng>, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *ranges::move_result<iterator_t<Rng>, FwdIter2>* otherwise. The *move* algorithm returns the pair of the input iterator *last* and the output iterator to the element in the destination range, one past the last element moved.

```
template<typename ExPolicy, typename Rng, typename FwdIter>
util::detail::algorithm_result<ExPolicy, ranges::move_result<typename hpx::traits::range_traits<Rng>::iterator_type, FwdIter>>
```

Moves the elements in the range *rng* to another range beginning at *dest*. After this operation the elements in the moved-from range will still contain valid values of the appropriate type, but not necessarily the same values as before the move.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly std::distance(begin(rng), end(rng)) assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **FwdIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.

The assignments in the parallel *copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *move* algorithm returns a *hpx::future<ranges::move_result<iterator_t<Rng>, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *ranges::move_result<iterator_t<Rng>, FwdIter2>* otherwise. The *move* algorithm returns the pair of the input iterator *last* and the output iterator to the element in the destination range, one past the last element moved.

hpx/parallel/container_algorithms/nth_element.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename RandomIt, typename Sent, typename Pred, typename Proj>
```

```
RandomIt nth_element (RandomIt first, RandomIt nth, Sent last, Pred &&pred, Proj &&proj)
```

nth_element is a partial sorting algorithm that rearranges elements in [first, last) such that the element pointed at by nth is changed to whatever element would occur in that position if [first, last) were sorted and all of the elements before this new nth element are less than or equal to the elements after the new nth element.

The comparison operations in the parallel *nth_element* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear in std::distance(first, last) on average. O(N) applications of the predicate, and O(N log N) swaps, where N = last - first.

Template Parameters

- RandomIt: The type of the source begin, nth, and end iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- Sent: The type of the source sentinel (deduced). This sentinel type must be a sentinel for RandomIt.
- Pred: Comparison function object which returns true if the first argument is less than the second.
- Proj: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- nth: Refers to the iterator defining the sort partition point
- last: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- pred: Specifies the comparison function object which returns true if the first argument is less than (i.e. is ordered before) the second. The signature of this comparison function should be equivalent to:

```
bool cmp (const Type1 &a, const Type2 &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type must be such that an object of type *randomIt* can be dereferenced and then implicitly converted to Type. This defaults to std::less<>.

- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked. This defaults to *projection_identity*.

Return The *nth_element* algorithm returns returns *RandomIt*. The *nth_element* algorithm returns an iterator equal to last.

```
template<typename ExPolicy, typename RandomIt, typename Sent, typename Pred, typename Proj>
```

```
parallel::util::detail::algorithm_result_t<ExPolicy, RandomIt> nth_element (ExPolicy &&policy,
```

```
RandomIt first,
```

```
RandomIt nth, Sent
```

```
last, Pred &&pred,
```

```
Proj &&proj)
```

nth_element is a partial sorting algorithm that rearranges elements in [first, last) such that the element pointed at by nth is changed to whatever element would occur in that position if [first, last) were sorted and all of the elements before this new nth element are less than or equal to the elements after the new nth element.

The comparison operations in the parallel *nth_element* invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in std::distance(first, last) on average. O(N) applications of the predicate, and O(N log N) swaps, where N = last - first.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **RandomIt**: The type of the source begin, nth, and end iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for RandomIt.
- **Pred**: Comparison function object which returns true if the first argument is less than the second.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **nth**: Refers to the iterator defining the sort partition point
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **pred**: Specifies the comparison function object which returns true if the first argument is less than (i.e. is ordered before) the second. The signature of this comparison function should be equivalent to:

```
bool cmp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type must be such that an object of type *randomIt* can be dereferenced and then implicitly converted to *Type*. This defaults to `std::less<>`.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked. This defaults to *projection_identity*.

The assignments in the parallel *nth_element* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *partition* algorithm returns a *hpx::future<RandomIt>* if the execution policy is of type *parallel_task_policy* and returns *RandomIt* otherwise. The *nth_element* algorithm returns an iterator equal to *last*.

```
template<typename Rng, typename Pred, typename Proj>
hpx::traits::range_iterator_t<Rng> nth_element (Rng &&rng, hpx::traits::range_iterator_t<Rng>
                                                 nth, Pred &&pred, Proj &&proj)
```

nth_element is a partial sorting algorithm that rearranges elements in [first, last) such that the element pointed at by *nth* is changed to whatever element would occur in that position if [first, last) were sorted and all of the elements before this new *nth* element are less than or equal to the elements after the new *nth* element.

The comparison operations in the parallel *nth_element* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear in std::distance(first, last) on average. O(N) applications of the predicate, and O(N log N) swaps, where N = last - first.

Template Parameters

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- **Pred**: Comparison function object which returns true if the first argument is less than the second.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **nth**: Refers to the iterator defining the sort partition point
- **pred**: Specifies the comparison function object which returns true if the first argument is less than (i.e. is ordered before) the second. The signature of this comparison function should be equivalent to:

```
bool cmp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type must be such that an object of type `randomIt` can be dereferenced and then implicitly converted to `Type`. This defaults to `std::less<>`.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked. This defaults to `projection_identity`.

Return The `nth_element` algorithm returns returns `hpx::traits::range_iterator_t<Rng>`. The `nth_element` algorithm returns an iterator equal to last.

```
template<typename ExPolicy, typename Rng, typename Pred, typename Proj>
parallel::util::detail::algorithm_result_t<ExPolicy, hpx::traits::range_iterator_t<Rng>> nth_element(ExPolicy
&&pol-
icy,
Rng
&&rng,
hpx::traits::range_
nth,
Pred
&&pred,
Proj
&&proj)
```

`nth_element` is a partial sorting algorithm that rearranges elements in [first, last) such that the element pointed at by `nth` is changed to whatever element would occur in that position if [first, last) were sorted and all of the elements before this new `nth` element are less than or equal to the elements after the new `nth` element.

The comparison operations in the parallel `nth_element` invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Linear in `std::distance(first, last)` on average. $O(N)$ applications of the predicate, and $O(N \log N)$ swaps, where $N = \text{last} - \text{first}$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- **Pred**: Comparison function object which returns true if the first argument is less than the second.

- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`
- Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **nth**: Refers to the iterator defining the sort partition point
- **pred**: Specifies the comparison function object which returns true if the first argument is less than (i.e. is ordered before) the second. The signature of this comparison function should be equivalent to:

```
bool cmp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type must be such that an object of type `randomIt` can be dereferenced and then implicitly converted to `Type`. This defaults to `std::less<>`.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked. This defaults to `projection_identity`.

The assignments in the parallel `nth_element` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `partition` algorithm returns a `hpx::future<hpx::traits::range_iterator_t<Rng>>` if the execution policy is of type `parallel_task_policy` and returns `hpx::traits::range_iterator_t<Rng>` otherwise. The `nth_element` algorithm returns an iterator equal to last.

hpx/parallel/container_algorithms/partial_sort.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename RandomIt, typename Sent, typename Comp, typename Proj>
RandomIt partial_sort(RandomIt first, Sent last, Comp &&comp = Comp(), Proj &&proj = Proj())
```

Places the first middle - first elements from the range [first, last) as sorted with respect to comp into the range [first, middle). The rest of the elements in the range [middle, last) are placed in an unspecified order.

The assignments in the parallel `partial_sort` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Approximately $(last - first) * \log(middle - first)$ comparisons.

Template Parameters

- **RandomIt**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `RandomIt`.
- **Comp**: The type of the function/function object to use (deduced). `Comp` defaults to `detail::less`.

- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `middle`: Refers to the middle of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator. `Comp` defaults to `detail::less`.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

Return The `partial_sort` algorithm returns `RandomIt`. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```
template<typename ExPolicy, typename RandomIt, typename Sent, typename Comp, typename Proj>
parallel::util::detail::algorithm_result_t<ExPolicy, RandomIt> partial_sort(ExPolicy &&policy,
                                                               RandomIt first,
                                                               RandomIt middle,
                                                               Sent last,
                                                               Comp &&comp =
                                                               = Comp(), Proj
                                                               &&proj = Proj())
```

Places the first `middle - first` elements from the range `[first, last)` as sorted with respect to `comp` into the range `[first, middle)`. The rest of the elements in the range `[middle, last)` are placed in an unspecified order.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Approximately $(last - first) * \log(middle - first)$ comparisons.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `RandomIt`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `RandomIt`.
- `Comp`: The type of the function/function object to use (deduced). `Comp` defaults to `detail::less`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `middle`: Refers to the middle of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator. `Comp` defaults to `detail::less`.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *partial_sort* algorithm returns a *hpx::future<RandomIt>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *RandomIt* otherwise. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```
template<typename Rng, typename Comp, typename Proj>hpx::traits::range_iterator<Rng>
```

Places the first middle - first elements from the range [first, last) as sorted with respect to comp into the range [first, middle). The rest of the elements in the range [middle, last) are placed in an unspecified order.

The assignments in the parallel *partial_sort* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Approximately (last - first) * log(middle - first) comparisons.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Comp*: The type of the function/function object to use (deduced). *Comp* defaults to *detail::less*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *middle*: Refers to the middle of the sequence of elements the algorithm will be applied to.
- *comp*: *comp* is a callable object. The return value of the INVOKE operation applied to an object of type *Comp*, when contextually converted to *bool*, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator. *Comp* defaults to *detail::less*.
- *proj*: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.

Return The *partial_sort* algorithm returns *typename hpx::traits::range_iterator<Rng>::type*. It returns *last*.

```
template<typename ExPolicy, typename Rng, typename Comp, typename Proj>util::detail::partial_sort_fn<ExPolicy, Rng, Comp, Proj>
```

Sorts the elements in the range [first, last) in ascending order. The relative order of equal elements is preserved. The function uses the given comparison function object *comp* (defaults to using operator<()).

A sequence is sorted with respect to a comparator *comp* and a projection *proj* if for every iterator *i* pointing to the sequence and every non-negative integer *n* such that *i + n* is a valid iterator pointing to an element of the sequence, and *(INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false)*.

Note Complexity: $O(N \log(N))$, where $N = \text{std::distance(first, last)}$ comparisons.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Comp**: The type of the function/function object to use (deduced). Comp defaults to `detail::less`;
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the middle of the sequence of elements the algorithm will be applied to.
- **comp**: comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator. Comp defaults to `detail::less`.
- **proj**: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `partial_sort` algorithm returns a `hpx::future<typename hpx::traits::range_iterator<Rng>::type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `typename hpx::traits::range_iterator<Rng>::type` otherwise. It returns *last*.

hpx/parallel/container_algorithms/partial_sort_copy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename InIter, typename Sent1, typename RandIter, typename Sent2, typename Comp, typename Proj>
partial_sort_copy_result<InIter, RandIter> partial_sort_copy(InIter first, Sent1 last, RandIter d_first, Sent2 d_last,
    Comp &&comp = Comp(),
    Proj1 &&proj1 = Proj1(),
    Proj2 &&proj2 = Proj2())
```

Sorts some of the elements in the range [first, last) in ascending order, storing the result in the range [d_first, d_last). At most d_last - d_first of the elements are placed sorted to the range [d_first, d_first + n) where n is the number of elements to sort (n = min(last - first, d_last - d_first)).

The assignments in the parallel `partial_sort_copy` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(N \log(\min(D, N)))$, where $N = \text{std}::distance(first, last)$ and $D = \text{std}::distance(d_first, d_last)$ comparisons.

Template Parameters

- **InIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- **Sent1**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter`.

- **RandIter**: The type of the destination iterators used(deduced) This iterator type must meet the requirements of an random iterator.
- **Sent2**: The type of the destination sentinel (deduced).This sentinel type must be a sentinel for **RandIter**.
- **Comp**: The type of the function/function object to use (deduced). **Comp** defaults to `detail::less`.
- **Proj1**: The type of an optional projection function for the input range. This defaults to `util::projection_identity`.
- **Proj1**: The type of an optional projection function for the output range. This defaults to `util::projection_identity`.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the sentinel value denoting the end of the sequence of elements the algorithm will be applied to.
- **d_first**: Refers to the beginning of the destination range.
- **d_last**: Refers to the sentinel denoting the end of the destination range.
- **comp**: **comp** is a callable object. The return value of the INVOKE operation applied to an object of type **Comp**, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that **comp** will not apply any non-constant function through the dereferenced iterator. This defaults to `detail::less`.
- **proj1**: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate **comp** is invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation after the actual predicate **comp** is invoked.

Return The `partial_sort_copy` algorithm returns a returns `partial_sort_copy_result<InIter, RandIter>`. The algorithm returns `{last, result_first + N}`.

```
template<typename ExPolicy, typename FwdIter, typename Sent1, typename RandIter, typename Sent2, typename parallel::util::detail::algorithm_result_t<ExPolicy>, partial_sort_copy_result<FwdIter, RandIter>> partial_sort_copy
```

Sorts some of the elements in the range `[first, last)` in ascending order, storing the result in the range `[d_first, d_last)`. At most $d_{last} - d_{first}$ of the elements are placed sorted to the range `[d_first, d_first + N)`.

+ n) where n is the number of elements to sort (n = min(last - first, d_last - d_first)).

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: O(Nlog(min(D,N))), where N = std::distance(first, last) and D = std::distance(d_first, d_last) comparisons.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent1**: The type of the source sentinel (deduced).This sentinel type must be a sentinel for FwdIter.
- **RandIter**: The type of the destination iterators used(deduced) This iterator type must meet the requirements of an random iterator.
- **Sent2**: The type of the destination sentinel (deduced).This sentinel type must be a sentinel for RandIter.
- **Comp**: The type of the function/function object to use (deduced). Comp defaults to detail::less.
- **Proj1**: The type of an optional projection function for the input range. This defaults to *util::projection_identity*.
- **Proj1**: The type of an optional projection function for the output range. This defaults to *util::projection_identity*.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the sentinel value denoting the end of the sequence of elements the algorithm will be applied to.
- **d_first**: Refers to the beginning of the destination range.
- **d_last**: Refers to the sentinel denoting the end of the destination range.
- **comp**: comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator. This defaults to detail::less.
- **proj1**: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation after the actual predicate *comp* is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *partial_sort_copy* algorithm returns a *hpx::future<partial_sort_copy_result<FwdIter, RandIter>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *partial_sort_copy_result<FwdIter, RandIter>* otherwise. The algorithm returns {last, result_first + N}.

template<typename **ExPolicy**, typename **FwdIter**, typename **Sent1**, typename **RandIter**, typename **Sent2**, typename

`partial_sort_copy_result<hpx::traits::range_iterator_t<Rng1>, hpx::traits::range_iterator_t<Rng2>> partial_sort_copy`

Sorts some of the elements in the range [first, last) in ascending order, storing the result in the range [d_first, d_last). At most d_last - d_first of the elements are placed sorted to the range [d_first, d_first + n) where n is the number of elements to sort ($n = \min(\text{last} - \text{first}, \text{d_last} - \text{d_first})$).

The assignments in the parallel *partial_sort_copy* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(N \log(\min(D, N)))$, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$ and $D = \text{std}::\text{distance}(\text{d_first}, \text{d_last})$ comparisons.

Template Parameters

- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a input iterator.
- `Rng2`: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of a random iterator.
- `Comp`: The type of the function/function object to use (deduced). `Comp` defaults to `detail::less`.
- `Proj1`: The type of an optional projection function for the input range. This defaults to `util::projection_identity`.
- `Proj2`: The type of an optional projection function for the output range. This defaults to `util::projection_identity`.

Parameters

- `rng1`: Refers to the source range.
- `rng2`: Refers to the destination range.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator. This defaults to `detail::less`.
- `proj1`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation after the actual predicate `comp` is invoked.

Return The *partial_sort_copy* algorithm returns `partial_sort_copy_result<range_iterator_t<Rng1>, range_iterator_t<Rng2>>`. The algorithm returns $\{\text{last}, \text{result_first} + N\}$.

template<typename **ExPolicy**, typename **FwdIter**, typename **Sent1**, typename **RandIter**, typename **Sent2**, typename

`parallel::util::detail::algorithm_result_t<ExPolicy>, partial_sort_copy_result<hpx::traits::range_iterator_t<Rng1>, hpx::traits`

Sorts some of the elements in the range [first, last) in ascending order, storing the result in the range [d_first, d_last). At most d_last - d_first of the elements are placed sorted to the range [d_first, d_first + n) where n is the number of elements to sort (n = min(last - first, d_last - d_first)).

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(N \log(\min(D, N)))$, where $N = \text{std}:\text{distance}(\text{first}, \text{last})$ and $D = \text{std}:\text{distance}(\text{d_first}, \text{d_last})$ comparisons.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- `Rng2`: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of a random iterator.
- `Comp`: The type of the function/function object to use (deduced). `Comp` defaults to `detail::less`.
- `Proj1`: The type of an optional projection function for the input range. This defaults to `util::projection_identity`.
- `Proj2`: The type of an optional projection function for the output range. This defaults to `util::projection_identity`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the source range.
- `rng2`: Refers to the destination range.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator. This defaults to `detail::less`.
- `proj1`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation after the actual predicate `comp` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object

of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *partial_sort_copy* algorithm returns a *hpx::future<partial_sort_copy_result<range_iterator_t<Rng1>, range_iterator_t<Rng2>>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *partial_sort_copy_result<range_iterator_t<Rng1>, range_iterator_t<Rng2>>* otherwise. The algorithm returns {last, result_first + N}.

hpx/parallel/container_algorithms/partition.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename FwdIter, typename Sent, typename Pred, typename Proj>
subrange_t<FwdIter> partition (ExPolicy &&policy, Sent first, Sent last, Pred &&pred, Proj
&&proj)
```

Reorders the elements in the range [first, last) in such a way that all elements for which the predicate *pred* returns true precede the elements for which the predicate *pred* returns false. Relative order of the elements is not preserved.

The assignments in the parallel *partition* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: At most $2 * (\text{last} - \text{first})$ swaps. Exactly $\text{last} - \text{first}$ applications of the predicate and projection.

Template Parameters

- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *partition* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *InIter* can be dereferenced and then implicitly converted to *Type*.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *partition* algorithm returns `subrange_t<FwdIter>`. The *partition* algorithm returns a subrange starting with an iterator to the first element of the second group and finishing with an iterator equal to `last`.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Pred, typename Proj>
util::detail::algorithm_result<ExPolicy, subrange_t<FwdIter>>::type partition(ExPolicy
    &&policy,
    FwdIter first,
    Sent last, Pred
    &&pred, Proj
    &&proj)
```

Reorders the elements in the range `[first, last)` in such a way that all elements for which the predicate `pred` returns true precede the elements for which the predicate `pred` returns false. Relative order of the elements is not preserved.

The assignments in the parallel *partition* algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: At most $2 * (\text{last} - \text{first})$ swaps. Exactly $\text{last} - \text{first}$ applications of the predicate and projection.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- `Pred`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *partition* requires `Pred` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `pred`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `InIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *partition* algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *partition* algorithm returns a `hpx::future<subrange_t<FwdIter>>` if the execution policy is of type `parallel_task_policy` and returns `subrange_t<FwdIter>` otherwise. The *partition* algorithm returns a subrange starting with an iterator to the first element of the second group and finishing with an iterator equal to last.

```
template<typename Rng, typename Pred, typename Proj>
subrange_t<hpx::traits::range_iterator_t<Rng>> partition(Rng &&rng, Pred &&pred, Proj &&proj)
```

Reorders the elements in the range *rng* in such a way that all elements for which the predicate *pred* returns true precede the elements for which the predicate *pred* returns false. Relative order of the elements is not preserved.

The assignments in the parallel *partition* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs at most $2 * N$ swaps, exactly N applications of the predicate and projection, where $N = \text{std}::\text{distance}(\text{begin}(\text{rng}), \text{end}(\text{rng}))$.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *partition* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by the range *rng*. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *partition* algorithm returns `subrange_t<hpx::traits::range_iterator_t<Rng>>` The *partition* algorithm returns a subrange starting with an iterator to the first element of the second group and finishing with an iterator equal to last.

```
template<typename ExPolicy, typename Rng, typename Pred, typename Proj>
util::detail::algorithm_result<ExPolicy, subrange_t<hpx::traits::range_iterator_t<Rng>>>::type partition(ExPolicy &&policy, Rng &&rng, Pred &&pred, Proj &&proj)
```

Reorders the elements in the range *rng* in such a way that all elements for which the predicate *pred* returns true precede the elements for which the predicate *pred* returns false. Relative order of the elements is not preserved.

The assignments in the parallel *partition* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs at most $2 * N$ swaps, exactly N applications of the predicate and projection, where $N = \text{std}::\text{distance}(\text{begin}(\text{rng}), \text{end}(\text{rng}))$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *partition* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by the range *rng*. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *partition* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *partition* algorithm returns a *hpx::future<subrange_t<hpx::traits::range_iterator_t<Rng>>>* if the execution policy is of type *parallel_task_policy* and returns *subrange_t<hpx::traits::range_iterator_t<Rng>>* The *partition* algorithm returns a subrange starting with an iterator to the first element of the second group and finishing with an iterator equal to last.

```
template<typename BidirIter, typename Sent, typename F, typename Proj>
subrange_t<BidirIter> stable_partition(BidirIter first, Sent last, F &&f, Proj &&proj)
```

Permutes the elements in the range [first, last) such that there exists an iterator i such that for every iterator j in the range [first, i) *INVOKE(f, INVOKE (proj, *j)) != false*, and for every iterator k in the range [i, last), *INVOKE(f, INVOKE (proj, *k)) == false*

The invocations of *f* in the parallel *stable_partition* algorithm invoked without an execution policy object executes in sequential order in the calling thread.

Note Complexity: At most $(\text{last} - \text{first}) * \log(\text{last} - \text{first})$ swaps, but only linear number of swaps if there is enough extra memory Exactly *last - first* applications of the predicate and projection.

Template Parameters

- *BidirIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *BidirIter*.

- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *f*: Unary predicate which returns true if the element should be ordered before other elements. Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool fun (const Type &a);
```

The signature does not need to have const&. The type *Type* must be such that an object of type *BidirIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *f* is invoked.

Return The *stable_partition* algorithm returns an iterator *i* such that for every iterator *j* in the range [first, *i*), *f(*j)* != false *(INVOKE(f, INVOKE(proj, *j))* != false, and for every iterator *k* in the range [*i*, last), *f(*k)* == false *(INVOKE(f, INVOKE(proj, *k))* == false. The relative order of the elements in both groups is preserved.

```
template<typename ExPolicy, typename BidirIter, typename Sent, typename F, typename Proj>
util::detail::algorithm_result<ExPolicy, subrange_t<BidirIter>>::type stable_partition (ExPolicy
    &&pol-
    icy,
    BidirIter
    first,
    Sent
    last,
    F
    &&f,
    Proj
    &&proj)
```

Permutes the elements in the range [first, last) such that there exists an iterator *i* such that for every iterator *j* in the range [first, *i*) *(INVOKE(f, INVOKE(proj, *j))* != false, and for every iterator *k* in the range [*i*, last), *(INVOKE(f, INVOKE(proj, *k))* == false

The invocations of *f* in the parallel *stable_partition* algorithm invoked with an execution policy object of type *sequenced_policy* executes in sequential order in the calling thread.

Note Complexity: At most (*last - first*) * $\log(\text{last} - \text{first})$ swaps, but only linear number of swaps if there is enough extra memory Exactly *last - first* applications of the predicate and projection.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of *f*.
- *BidirIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *BidirIter*.
- *F*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires *F* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `f`: Unary predicate which returns true if the element should be ordered before other elements. Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool fun (const Type &a);
```

The signature does not need to have `const&`. The type `Type` must be such that an object of type `BidirIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `f` is invoked.

The invocations of `f` in the parallel `stable_partition` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `stable_partition` algorithm returns an iterator `i` such that for every iterator `j` in the range `[first, i)`, `f(*j) != false` `INVOKE(f, INVOKE(proj, *j)) != false`, and for every iterator `k` in the range `[i, last)`, `f(*k) == false` `INVOKE(f, INVOKE(proj, *k)) == false`. The relative order of the elements in both groups is preserved. If the execution policy is of type `parallel_task_policy` the algorithm returns a future`<>` referring to this iterator.

```
template<typename Rng, typename F, typename Proj>
subrange_t<hpx::traits::range_iterator_t<Rng>> stable_partition (Rng &&rng, F &&f, Proj &&proj)
```

Permutes the elements in the range `[first, last)` such that there exists an iterator `i` such that for every iterator `j` in the range `[first, i)` `INVOKE(f, INVOKE(proj, *j)) != false`, and for every iterator `k` in the range `[i, last)`, `INVOKE(f, INVOKE(proj, *k)) == false`

The invocations of `f` in the parallel `stable_partition` algorithm invoked without an execution policy object executes in sequential order in the calling thread.

Note Complexity: At most $(last - first) * \log(last - first)$ swaps, but only linear number of swaps if there is enough extra memory Exactly `last - first` applications of the predicate and projection.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an bidirectional iterator
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `transform` requires `F` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `f`: Unary predicate which returns true if the element should be ordered before other elements. Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool fun (const Type &a);
```

The signature does not need to have `const&`. The type `Type` must be such that an object of type `BidirIter` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `f` is invoked.

Return The *stable_partition* algorithm returns an iterator *i* such that for every iterator *j* in the range [first, *i*), *f*(**j*) != false *(INVOKE(f, INVOKE(proj, *j)) != false*, and for every iterator *k* in the range [*i*, last), *f*(**k*) == false *(INVOKE(f, INVOKE(proj, *k)) == false*. The relative order of the elements in both groups is preserved.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj>
util::detail::algorithm_result<ExPolicy, subrange_t<hpx::traits::range_iterator_t<Rng>>>::type stable_partition(ExPolicy &exPolicy, Rng &rng, F &f, Proj &proj)
```

Permutes the elements in the range [first, last) such that there exists an iterator *i* such that for every iterator *j* in the range [first, *i*] *(INVOKE(f, INVOKE(proj, *j)) != false*, and for every iterator *k* in the range [*i*, last), *(INVOKE(f, INVOKE(proj, *k)) == false*

The invocations of *f* in the parallel *stable_partition* algorithm invoked with an execution policy object of type *sequenced_policy* executes in sequential order in the calling thread.

Note Complexity: At most (last - first) * log(last - first) swaps, but only linear number of swaps if there is enough extra memory. Exactly *last - first* applications of the predicate and projection.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of *f*.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an bidirectional iterator
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **f**: Unary predicate which returns true if the element should be ordered before other elements. Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```
bool fun(const Type &a);
```

The signature does not need to have *const&*. The type *Type* must be such that an object of type *BidirIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *f* is invoked.

The invocations of *f* in the parallel *stable_partition* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *stable_partition* algorithm returns an iterator *i* such that for every iterator *j* in the range [first, *i*), *f*(**j*) != false *(INVOKE(f, INVOKE(proj, *j)) != false*, and for every iterator *k* in the range [*i*, last), *f*(**k*) == false *(INVOKE(f, INVOKE(proj, *k)) == false*. The relative order of the

elements in both groups is preserved. If the execution policy is of type *parallel_task_policy* the algorithm returns a future<> referring to this iterator.

```
template<typename FwdIter1, typename Sent, typename FwdIter2, typename FwdIter3, typename Pred, typename Proj  

hpX::util::in_out_out<FwdIter1, FwdIter2, FwdIter3> partition_copy (FwdIter1 first, Sent last,  

FwdIter2 dest_true,  

FwdIter3 dest_false,  

Pred &&pred, Proj  

&&proj)
```

Copies the elements in the range, defined by [first, last), to two different ranges depending on the value returned by the predicate *pred*. The elements, that satisfy the predicate *pred* are copied to the range beginning at *dest_true*. The rest of the elements are copied to the range beginning at *dest_false*. The order of the elements is preserved.

The assignments in the parallel `partition_copy` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs not more than $last - first$ assignments, exactly $last - first$ applications of the predicate f .

Template Parameters

- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
 - `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
 - `FwdIter2`: The type of the iterator representing the destination range for the elements that satisfy the predicate `pred` (deduced). This iterator type must meet the requirements of an forward iterator.
 - `FwdIter3`: The type of the iterator representing the destination range for the elements that don't satisfy the predicate `pred` (deduced). This iterator type must meet the requirements of an forward iterator.
 - `Pred`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `partition_copy` requires `Pred` to meet the requirements of `CopyConstructible`.
 - `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
 - **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
 - **dest_true**: Refers to the beginning of the destination range for the elements that satisfy the predicate *pred*
 - **dest_false**: Refers to the beginning of the destination range for the elements that don't satisfy the predicate *pred*.
 - **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIterI` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *partition_copy* algorithm returns a *partition_copy_result*<*FwdIter*, *OutIter2*, *OutIter3*>.

The *partition_copy* algorithm returns the tuple of the source iterator *last*, the destination iterator to the end of the *dest_true* range, and the destination iterator to the end of the *dest_false* range.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename FwdIter3, ty  
util::detail::algorithm_result<ExPolicy, partition_copy_result<FwdIter, OutIter2, OutIter3>>::type partition_copy (Ex
```

Copies the elements in the range, defined by [first, last), to two different ranges depending on the value returned by the predicate *pred*. The elements, that satisfy the predicate *pred* are copied to the range beginning at *dest_true*. The rest of the elements are copied to the range beginning at *dest_false*. The order of the elements is preserved.

The assignments in the parallel *partition_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first* applications of the predicate *f*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *FwdIter2*: The type of the iterator representing the destination range for the elements that satisfy the predicate *pred* (deduced). This iterator type must meet the requirements of an forward iterator.
- *FwdIter3*: The type of the iterator representing the destination range for the elements that don't satisfy the predicate *pred* (deduced). This iterator type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *partition_copy* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest_true*: Refers to the beginning of the destination range for the elements that satisfy the predicate *pred*

- `dest_false`: Refers to the beginning of the destination range for the elements that don't satisfy the predicate `pred`.
- `pred`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` invoked.

The assignments in the parallel `partition_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `partition_copy` algorithm returns a `hpx::future<partition_copy_result<FwdIter, OutIter2, OutIter3>>` if the execution policy is of type `parallel_task_policy` and returns `partition_copy_result<FwdIter, OutIter2, OutIter3>` otherwise. The `partition_copy` algorithm returns the tuple of the source iterator `last`, the destination iterator to the end of the `dest_true` range, and the destination iterator to the end of the `dest_false` range.

```
template<typename Rng, typename FwdIter2, typename FwdIter3, typename Pred, typename Proj>
friend partition_copy_result<hpx::traits::range_iterator_t<Rng>, FwdIter2, FwdIter3>::type partition_copy(Rng
&&r
FwdI
dest_
FwdI
dest_
Pred
&&p
Proj
&&p
```

Copies the elements in the range `rng`, to two different ranges depending on the value returned by the predicate `pred`. The elements, that satisfy the predicate `pred` are copied to the range beginning at `dest_true`. The rest of the elements are copied to the range beginning at `dest_false`. The order of the elements is preserved.

The assignments in the parallel `partition_copy` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs not more than `N` assignments, exactly `N` applications of the predicate `pred`, where `N = std::distance(begin(rng), end(rng))`.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the iterator representing the destination range for the elements that satisfy the predicate `pred` (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter3`: The type of the iterator representing the destination range for the elements that don't satisfy the predicate `pred` (deduced). This iterator type must meet the requirements of an forward iterator.
- `Pred`: The type of the function/function object to use (deduced). Unlike its sequential form,

the parallel overload of *partition_copy* requires *Pred* to meet the requirements of *CopyConstructible*.

- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest_true*: Refers to the beginning of the destination range for the elements that satisfy the predicate *pred*
- *dest_false*: Refers to the beginning of the destination range for the elements that don't satisfy the predicate *pred*.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *partition_copy* algorithm returns a *partition_copy_result*<*hpx::traits::range_iterator_t<Rng>, FwdIter2, FwdIter3>>*. The *partition_copy* algorithm returns the tuple of the source iterator *last*, the destination iterator to the end of the *dest_true* range, and the destination iterator to the end of the *dest_false* range.

```
template<typename ExPolicy, typename Rng, typename FwdIter2, typename FwdIter3, typename Pred, typename ...>
friend parallel::util::detail::algorithm_result<ExPolicy, partition_copy_result<hpx::traits::range_iterator_t<Rng>, FwdIter2, FwdIter3>> partition_copy(
```

Copies the elements in the range *rng*, to two different ranges depending on the value returned by the predicate *pred*. The elements, that satisfy the predicate *pred* are copied to the range beginning at *dest_true*. The rest of the elements are copied to the range beginning at *dest_false*. The order of the elements is preserved.

The assignments in the parallel *partition_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than N assignments, exactly N applications of the predicate *pred*, where N = std::distance(begin(rng), end(rng)).

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the iterator representing the destination range for the elements that satisfy the predicate `pred` (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter3`: The type of the iterator representing the destination range for the elements that don't satisfy the predicate `pred` (deduced). This iterator type must meet the requirements of an forward iterator.
- `Pred`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `partition_copy` requires `Pred` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest_true`: Refers to the beginning of the destination range for the elements that satisfy the predicate `pred`
- `dest_false`: Refers to the beginning of the destination range for the elements that don't satisfy the predicate `pred`.
- `pred`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel `partition_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `partition_copy` algorithm returns a `hpx::future<partition_copy_result<hpx::traits::range_iterator_t<Rng>, FwdIter2, FwdIter3>>` if the execution policy is of type `parallel_task_policy` and returns `partition_copy_result<hpx::traits::range_iterator_t<Rng>, FwdIter2, FwdIter3>` otherwise. The `partition_copy` algorithm returns the tuple of the source iterator `last`, the destination iterator to the end of the `dest_true` range, and the destination iterator to the end of the `dest_false` range.

hpx/parallel/container_algorithms/reduce.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename T, typename F>
util::detail::algorithm_result<ExPolicy, T>::type reduce (ExPolicy &&policy, FwdIter first, Sent
last, T init, F &&f)
>Returns GENERALIZED_SUM(f, init, *first, ..., *(first + (last - first) - 1)).
```

The reduce operations in the parallel *reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *f*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel used (deduced). This iterator type must meet the requirements of an forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *copy_if* requires *F* to meet the requirements of *CopyConstructible*.
- **T**: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have **const&**. The types *Type1 Ret* must be such that an object of type *FwdIterB* can be dereferenced and then implicitly converted to any of those types.

- **init**: The initial value for the generalized sum.

The reduce operations in the parallel *copy_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *reduce* and *accumulate* is that the behavior of *reduce* may be non-deterministic for non-associative or non-commutative binary predicate.

Return The *reduce* algorithm returns a *hpx::future<T>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *T* otherwise. The *reduce* algorithm returns the result of the generalized sum over the elements given by the input range [first, last].

Note GENERALIZED_SUM(*op*, *a*₁, ..., *a*_N) is defined as follows:

- *a*₁ when *N* is 1
- *op*(GENERALIZED_SUM(*op*, *b*₁, ..., *b*_K), GENERALIZED_SUM(*op*, *b*_M, ..., *b*_N)), where:
 - *b*₁, ..., *b*_N may be any permutation of *a*₁, ..., *a*_N and
 - $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename T>
```

```
util::detail::algorithm_result<ExPolicy, T>::type reduce (ExPolicy &&policy, FwdIter first, Sent last, T init)
>Returns GENERALIZED_SUM(+, init, *first, ..., *(first + (last - first) - 1)).
```

The reduce operations in the parallel *reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: O(*last - first*) applications of the operator+().

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel used (deduced). This iterator type must meet the requirements of an forward iterator.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *init*: The initial value for the generalized sum.

The reduce operations in the parallel *copy_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *reduce* and *accumulate* is that the behavior of *reduce* may be non-deterministic for non-associative or non-commutative binary predicate.

Return The *reduce* algorithm returns a *hpx::future<T>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *T* otherwise. The *reduce* algorithm returns the result of the generalized sum (applying operator+()) over the elements given by the input range [first, last).

Note GENERALIZED_SUM(+, a1, ..., aN) is defined as follows:

- a1 when N is 1
- op(GENERALIZED_SUM(+, b1, ..., bK), GENERALIZED_SUM(+, bM, ..., bN)), where:
 - b1, ..., bN may be any permutation of a1, ..., aN and
 - 1 < K+1 = M <= N.

```
template<typename ExPolicy, typename FwdIter, typename Sent>
util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<FwdIter>::value_type>::type reduce (ExPolicy &&policy,
&&policy,
FwdIter first,
Sent last)
```

Returns GENERALIZED_SUM(+, T(), *first, ..., *(first + (last - first) - 1)).

The reduce operations in the parallel *reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: O(*last - first*) applications of the operator+().

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel used (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

The reduce operations in the parallel *copy_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *reduce* and *accumulate* is that the behavior of *reduce* may be non-deterministic for non-associative or non-commutative binary predicate.

Return The *reduce* algorithm returns a *hpx::future<T>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *T* otherwise (where *T* is the value_type of *FwdIterB*). The *reduce* algorithm returns the result of the generalized sum (applying operator+()) over the elements given by the input range [first, last).

Note The type of the initial value (and the result type) *T* is determined from the value_type of the used *FwdIterB*.

Note GENERALIZED_SUM(+, a1, ..., aN) is defined as follows:

- a1 when N is 1
- op(GENERALIZED_SUM(+, b1, ..., bK), GENERALIZED_SUM(+, bM, ..., bN)), where:
 - b1, ..., bN may be any permutation of a1, ..., aN and
 - 1 < K+1 = M <= N.

```
template<typename ExPolicy, typename Rng, typename T, typename FExPolicy, T>::type reduce (ExPolicy &&policy, Rng &&rng, T init,  

F &&f)  
Returns GENERALIZED_SUM(f, init, *first, ..., *(first + (last - first) - 1)).
```

The reduce operations in the parallel *reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: O(*last - first*) applications of the predicate *f*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *copy_if* requires *F* to meet the requirements of *CopyConstructible*.
- **T**: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`. The types `Type1 Ret` must be such that an object of type `FwdIterB` can be dereferenced and then implicitly converted to any of those types.

- `init`: The initial value for the generalized sum.

The reduce operations in the parallel `copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `reduce` and `accumulate` is that the behavior of `reduce` may be non-deterministic for non-associative or non-commutative binary predicate.

Return The `reduce` algorithm returns a `hpx::future<T>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `T` otherwise. The `reduce` algorithm returns the result of the generalized sum over the elements given by the input range `[first, last)`.

Note `GENERALIZED_SUM(op, a1, ..., aN)` is defined as follows:

- $a1$ when N is 1
- $op(GENERALIZED_SUM(op, b1, \dots, bK), GENERALIZED_SUM(op, bM, \dots, bN))$, where:
 - $b1, \dots, bN$ may be any permutation of $a1, \dots, aN$ and
 - $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng, typename T>
util::detail::algorithm_result<ExPolicy, T>::type reduce (ExPolicy &&policy, Rng &&rng, T init)
    Returns GENERALIZED_SUM(+, init, *first, ..., *(first + (last - first) - 1)).
```

The reduce operations in the parallel `reduce` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the operator`+`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `init`: The initial value for the generalized sum.

The reduce operations in the parallel `copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `reduce` and `accumulate` is that the behavior of `reduce` may be non-deterministic for non-associative or non-commutative binary predicate.

Return The `reduce` algorithm returns a `hpx::future<T>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `T` otherwise. The `reduce` algorithm returns the result of the generalized sum (applying operator`+`()) over the elements given by the input range `[first, last)`.

Note `GENERALIZED_SUM(+, a1, ..., aN)` is defined as follows:

- $a1$ when N is 1

- $\text{op}(\text{GENERALIZED_SUM}(+, b_1, \dots, b_K), \text{GENERALIZED_SUM}(+, b_M, \dots, b_N))$, where:
 - b_1, \dots, b_N may be any permutation of a_1, \dots, a_N and
 - $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng>
util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<typename hpx::traits::range_traits<Rng>::iterator>
```

Returns $\text{GENERALIZED_SUM}(+, T(), *first, \dots, *(first + (last - first) - 1))$.

The reduce operations in the parallel *reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the operator $+()$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.

The reduce operations in the parallel *copy_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *reduce* and *accumulate* is that the behavior of *reduce* may be non-deterministic for non-associative or non-commutative binary predicate.

Return The *reduce* algorithm returns a *hpx::future<T>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *T* otherwise (where *T* is the *value_type* of *FwdIterB*). The *reduce* algorithm returns the result of the generalized sum (applying operator $+()$) over the elements given by the input range [first, last].

Note The type of the initial value (and the result type) *T* is determined from the *value_type* of the used *FwdIterB*.

Note $\text{GENERALIZED_SUM}(+, a_1, \dots, a_N)$ is defined as follows:

- a_1 when N is 1
- $\text{op}(\text{GENERALIZED_SUM}(+, b_1, \dots, b_K), \text{GENERALIZED_SUM}(+, b_M, \dots, b_N))$, where:
 - b_1, \dots, b_N may be any permutation of a_1, \dots, a_N and
 - $1 < K+1 = M \leq N$.

hpx/parallel/algorithms/reduce_by_key.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

Functions

```
template<typename ExPolicy, typename RanIter, typename RanIter2, typename FwdIter1, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, util::in_out_result<FwdIter1, FwdIter2>>::type reduce_by_key(ExPolicy
    &&policy,
    Ran-
    Iter
    key_first,
    Ran-
    Iter
    key_last,
    Ran-
    Iter2
    val-
    ues_first,
    FwdIter1
    keys_output,
    FwdIter2
    val-
    ues_output,
    Com-
    pare
    &&comp
    =
    Com-
    pare(),
    Func
    &&func
    =
    Func())
```

Reduce by Key performs an inclusive scan reduction operation on elements supplied in key/value pairs. The algorithm produces a single output value for each set of equal consecutive keys in [key_first, key_last). the value being the GENERALIZED_NONCOMMUTATIVE_SUM(op, init, *first, ..., *(first + (i - result))). for the run of consecutive matching keys. The number of keys supplied must match the number of values.

comp has to induce a strict weak ordering on the values.

Note Complexity: $O(last - first)$ applications of the predicate *op*.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **RanIter:** The type of the key iterators used (deduced). This iterator type must meet the requirements of a random access iterator.

- `RanIter2`: The type of the value iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- `FwdIter1`: The type of the iterator representing the destination key range (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the iterator representing the destination value range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Compare`: The type of the optional function/function object to use to compare keys (deduced). Assumed to be `std::equal_to` otherwise.
- `Func`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `copy_if` requires `F` to meet the requirements of `CopyConstructible`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `key_first`: Refers to the beginning of the sequence of key elements the algorithm will be applied to.
- `key_last`: Refers to the end of the sequence of key elements the algorithm will be applied to.
- `values_first`: Refers to the beginning of the sequence of value elements the algorithm will be applied to.
- `keys_output`: Refers to the start output location for the keys produced by the algorithm.
- `values_output`: Refers to the start output location for the values produced by the algorithm.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.
- `func`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`. The types `Type1` `Ret` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to any of those types.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `reduce_by_key` algorithm returns a `hpx::future<pair<Iter1,Iter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `pair<Iter1,Iter2>` otherwise.

hpx/parallel/container_algorithms/remove.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter, typename Sent, typename T, typename Proj = util::projection_identity>
subrange_t<FwdIter, Sent> remove(FwdIter first, Sent last, T const &value, Proj &&proj =
    Proj())
```

Removes all elements that are equal to *value* from the range [first, last) and returns a subrange [ret, last), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first* applications of the operator`==()` and the projection *proj*.

Template Parameters

- *FwdIter*: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *T*: The type of the value to remove (deduced). This value type must meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *value*: Specifies the value of elements to remove.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *remove* algorithm returns a *subrange_t<FwdIter, Sent>*. The *remove* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange of the values all in valid but unspecified state.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename T, typename Proj = util::projection_id
```

```
parallel::util::detail::algorithm_result<ExPolicy, subrange_t<FwdIter, Sent>>::type remove (ExPolicy
    &&policy,
    FwdIter
    first,
    Sent
    last,
    T
    const
    &value,
    Proj
    &&proj
    =
    Proj())

```

Removes all elements that are equal to *value* from the range [first, last) and returns a subrange [ret, last), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first* applications of the operator==() and the projection *proj*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *T*: The type of the value to remove (deduced). This value type must meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *value*: Specifies the value of elements to remove.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *remove* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *remove* algorithm returns a *hpx::future<subrange_t<FwdIter, Sent>>*. The *remove* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange of the values all in valid but unspecified state.

```
template<typename Rng, typename T, typename Proj = util::projection_identity>
subrange_t<typename hpx::traits::range_iterator<Rng>::type> remove (Rng &&rng, T const
    &value, Proj &&proj =
    Proj())

```

Removes all elements that are equal to *value* from the range *rng* and returns a subrange [ret, *util::end(rng)*), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs not more than `util::end(rng)`

- `util::begin(rng)` assignments, exactly `util::end(rng) - util::begin(rng)` applications of the operator`==()` and the projection `proj`.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `T`: The type of the value to remove (deduced). This value type must meet the requirements of *CopyConstructible*.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `value`: Specifies the value of elements to remove.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *remove* algorithm returns a `subrange_t<typename hpx::traits::range_iterator<Rng>::type>`. The *remove* algorithm returns an object `{ret, last}`, where `ret` is a past-the-end iterator for a new subrange of the values all in valid but unspecified state.

```
template<typename ExPolicy, typename Rng, typename T, typename Proj = util::projection_identity>
parallel::util::detail::algorithm_result<ExPolicy, subrange_t<typename hpx::traits::range_iterator<Rng>::type>>::type re
```

Removes all elements that are equal to `value` from the range `rng` and and returns a subrange `[ret, util::end(rng)]`, where `ret` is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than `util::end(rng)`

- `util::begin(rng)` assignments, exactly `util::end(rng) - util::begin(rng)` applications of the operator`==()` and the projection `proj`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `T`: The type of the value to remove (deduced). This value type must meet the requirements of *CopyConstructible*.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **value**: Specifies the value of elements to remove.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *remove* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *remove* algorithm returns a *hpx::future<subrange_t<typename hpx::traits::range_iterator<Rng>::type>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *remove* algorithm returns the iterator to the new end of the range.

```
template<typename FwdIter, typename Sent, typename Pred, typename Proj = hpx::parallel::util::projection_identity<typename hpx::traits::range_iterator<FwdIter>::type>>
subrange_t<FwdIter, Sent> remove_if(FwdIter first, Sent sent, Pred &&pred, Proj &&proj = Proj())
```

Removes all elements for which predicate *pred* returns true from the range [first, last) and returns a subrange [ret, last), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove_if* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first* applications of the predicate *pred* and the projection *proj*.

Template Parameters

- **FwdIter**: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *remove_if* requires *Pred* to meet the requirements of *CopyConstructible..*
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *remove_if* algorithm returns a *subrange_t<FwdIter, Sent>*. The *remove_if* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange of the values all in valid but unspecified state.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Pred, typename Proj = hpx::parallel::util::projection_identity<typename hpx::traits::range_iterator<FwdIter>::type>>
subrange_t<FwdIter, Sent> remove_if(FwdIter first, Sent sent, Pred &&pred, Proj &&proj = Proj(), ExPolicy ex_policy)
```

```
parallel::util::detail::algorithm_result<ExPolicy, subrange_t<FwdIter, Sent>>::type remove_if (ExPolicy
    &&policy,
    FwdIter
    first,
    Sent
    sent,
    Pred
    &&pred,
    Proj
    &&proj
    =
    Proj())
```

Removes all elements for which predicate *pred* returns true from the range [first, last) and returns a subrange [ret, last), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first* applications of the predicate *pred* and the projection *proj*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *remove_if* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *remove_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *remove_if* algorithm returns a *hpx::future<subrange_t<FwdIter, Sent>>*. The *remove_if* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new

subrange of the values all in valid but unspecified state.

```
template<typename Rng, typename T, typename Proj = util::projection_identity>
subrange_t<typename hpx::traits::range_iterator<Rng>::type> remove_if(Rng &&rng, Pred
&&pred, Proj
&&proj = Proj())
```

Removes all elements that are equal to *value* from the range *rng* and and returns a subrange [ret, util::end(rng)), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove_if* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs not more than *util::end(rng)*

- *util::begin(rng)* assignments, exactly *util::end(rng) - util::begin(rng)* applications of the operator==() and the projection *proj*.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *remove_if* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *remove_if* algorithm returns a *subrange_t<typename hpx::traits::range_iterator<Rng>::type>*. The *remove_if* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange of the values all in valid but unspecified state.

```
template<typename ExPolicy, typename Rng, typename Pred, typename Proj = util::projection_identity>
parallel::util::detail::algorithm_result<ExPolicy, subrange_t<typename hpx::traits::range_iterator<Rng>::type>>::type re
```

Removes all elements that are equal to *value* from the range *rng* and and returns a subrange [ret, util::end(rng)), where ret is a past-the-end iterator for the new end of the range.

The assignments in the parallel *remove_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than *util::end(rng)*

- *util::begin(rng)* assignments, exactly *util::end(rng) - util::begin(rng)* applications of the operator==() and the projection *proj*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *remove_if* requires *Pred* to meet the requirements of *CopyConstructible*.
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *remove_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *remove_if* algorithm returns a *hpx::future<subrange_t< typename hpx::traits::range_iterator<Rng>::type>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *remove_if* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange of the values all in valid but unspecified state.

hpx/parallel/container_algorithms/remove_copy.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/parallel/container_algorithms/replace.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename Iter, typename Sent, typename T1, typename T2, typename Proj = hpx::parallel::util::projection_
Iter replace (Iter first, Sent sent, T1 const &old_value, T2 const &new_value, Proj &&proj =
Proj())
```

Replaces all elements satisfying specific criteria with *new_value* in the range [first, last).

Effects: Substitutes elements referred by the iterator it in the range [first, last) with *new_value*, when the following corresponding conditions hold: $\text{INVOKED}(\text{proj}, *i) == \text{old_value}$

Note Complexity: Performs exactly *last - first* assignments.

The assignments in the parallel *replace* algorithm execute in sequential order in the calling thread.

Template Parameters

- **Iter**: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *Iter*.
- **T1**: The type of the old value to replace (deduced).
- **T2**: The type of the new values to replace (deduced).
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **old_value**: Refers to the old value of the elements to replace.
- **new_value**: Refers to the new value to use as the replacement.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *replace* algorithm returns an *Iter*.

```
template<typename Rng, typename T1, typename T2, typename Proj = hpx::parallel::util::projection_identity>
hpx::traits::range_iterator<Rng>::type replace (Rng &&rng, T1 const &old_value, T2 const
&new_value, Proj &&proj = Proj())
```

Replaces all elements satisfying specific criteria with *new_value* in the range *rng*.

Effects: Substitutes elements referred by the iterator it in the range *rng* with *new_value*, when the following corresponding conditions hold: $\text{INVOKED}(\text{proj}, *i) == \text{old_value}$

Note Complexity: Performs exactly *util::end(rng) - util::begin(rng)* assignments.

The assignments in the parallel *replace* algorithm execute in sequential order in the calling thread.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- `T1`: The type of the old value to replace (deduced).
- `T2`: The type of the new values to replace (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `old_value`: Refers to the old value of the elements to replace.
- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *replace* algorithm returns an `hpx::traits::range_iterator<Rng>::type`.

```
template<typename ExPolicy, typename Iter, typename Sent, typename T1, typename T2, typename Proj = hpx::parallel::util::detail::algorithm_result<ExPolicy, Iter>::type> replace(ExPolicy &&policy, Iter first, Sent sent, T1 const &old_value, T2 const &new_value, Proj &&proj = Proj())
```

Replaces all elements satisfying specific criteria with `new_value` in the range [first, last).

Effects: Substitutes elements referred by the iterator it in the range [first, last) with `new_value`, when the following corresponding conditions hold: `INVOKE(proj, *i) == old_value`

Note Complexity: Performs exactly `last - first` assignments.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Iter`: The type of the source iterator used (deduced). The iterator type must meet the requirements of a forward iterator.
- `Sent`: The type of the end iterators used (deduced). This sentinel type must be a sentinel for `Iter`.
- `T1`: The type of the old value to replace (deduced).
- `T2`: The type of the new values to replace (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `old_value`: Refers to the old value of the elements to replace.
- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace* algorithm returns a `hpx::future<Iter>` if the execution policy is of type `se-`

quenced_task_policy or *parallel_task_policy* and returns *Iter* otherwise.

```
template<typename ExPolicy, typename Rng, typename T1, typename T2, typename Proj = hpx::parallel::util::projection
parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type replace (ExPolicy
&& icy,
Rng
&& T1
const &ole
T2
const &ne
Proj
&& =
Proj)
```

Replaces all elements satisfying specific criteria with *new_value* in the range *rng*.

Effects: Substitutes elements referred by the iterator *it* in the range *rng* with *new_value*, when the following corresponding conditions hold: *INVOKE(proj, *i) == old_value*

Note Complexity: Performs exactly *util::end(rng) - util::begin(rng)* assignments.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **T1**: The type of the old value to replace (deduced).
- **T2**: The type of the new values to replace (deduced).
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **old_value**: Refers to the old value of the elements to replace.
- **new_value**: Refers to the new value to use as the replacement.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* is invoked. The assignments in the parallel *replace* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Return The *replace* algorithm returns an *hpx::future<hpx::traits::range_iterator<Rng>::type>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *hpx::traits::range_iterator<Rng>::type* otherwise.

```
template<typename Iter, typename Sent, typename Pred, typename T, typename Proj = hpx::parallel::util::projection
Iter replace_if (Iter first, Sent sent, Pred &&pred, T const &new_value, Proj &&proj =
Proj())
```

Replaces all elements satisfying specific criteria (for which predicate *f* returns true) with *new_value* in the range [first, sent).

Effects: Substitutes elements referred by the iterator it in the range [first, sent) with new_value, when the following corresponding conditions hold: $\text{(INVOKE}(f, \text{(INVOKE}(\text{proj}, *it)) \neq \text{false})$

Note Complexity: Performs exactly *sent - first* applications of the predicate.

The assignments in the parallel *replace_if* algorithm execute in sequential order in the calling thread.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Iter*: The type of the source iterator used (deduced). The iterator type must meet the requirements of a forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *Iter*.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*. (deduced).
- *T*: The type of the new values to replace (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns *true* for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *Iter* can be dereferenced and then implicitly converted to *Type*.

- *new_value*: Refers to the new value to use as the replacement.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *replace_if* algorithm returns an *Iter* It returns *last*.

```
template<typename Rng, typename Pred, typename T, typename Proj = hpx::parallel::util::projection_identity>
hpx::traits::range_iterator<Rng>::type replace_if(Rng &&rng, Pred &&pred, T const &new_value, Proj &&proj = Proj())
```

Replaces all elements satisfying specific criteria (for which predicate *pred* returns true) with *new_value* in the range *rng*.

Effects: Substitutes elements referred by the iterator it in the range *rng* with *new_value*, when the following corresponding conditions hold: $\text{(INVOKE}(f, \text{(INVOKE}(\text{proj}, *it)) \neq \text{false})$

Note Complexity: Performs exactly *util::end(rng) - util::begin(rng)* applications of the predicate.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Template Parameters

- **Rng:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **Pred:** The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *Pred* to meet the requirements of *CopyConstructible*. (deduced).
- **T:** The type of the new values to replace (deduced).
- **Proj:** The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **rng:** Refers to the sequence of elements the algorithm will be applied to.
- **pred:** Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by *rng*. This is an unary predicate which returns *true* for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- **new_value:** Refers to the new value to use as the replacement.
- **proj:** Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_if* algorithm returns an *hpx::traits::range_iterator<Rng>::type* It returns *last*.

```
template<typename ExPolicy, typename Iter, typename Sent, typename Pred, typename T
```

Replaces all elements satisfying specific criteria (for which predicate *pred* returns *true*) with *new_value* in the range *rng*.

Effects: Substitutes elements referred by the iterator *it* in the range *rng* with *new_value*, when the following corresponding conditions hold: *(INVOKE(f, INVOKE(proj, *it)) != false)*

Note Complexity: Performs exactly *util::end(rng) - util::begin(rng)* applications of the predicate.

The assignments in the parallel *replace_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter:** The type of the source iterator used (deduced). The iterator type must meet the requirements of a forward iterator.
- **Sent:** The type of the end iterators used (deduced). This sentinel type must be a sentinel for *Iter*.
- **Pred:** The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *Pred* to meet the requirements of *CopyConstructible*. (deduced).
- **T:** The type of the new values to replace (deduced).
- **Proj:** The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is an unary predicate which returns *true* for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *new_value*: Refers to the new value to use as the replacement.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *replace_if* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_if* algorithm returns a *hpx::future<Iter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy*. It returns *last*.

```
template<typename ExPolicy, typename Rng, typename Pred, typename T, typename Proj = hpx::parallel::util::projected<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type replace_if(
```

Replaces all elements satisfying specific criteria (for which predicate *pred* returns *true*) with *new_value* in the range *rng*.

Effects: Substitutes elements referred by the iterator *it* in the range *rng* with *new_value*, when the following corresponding conditions hold: *(INVOKE(f, INVOKE(proj, *it)) != false)*

Note Complexity: Performs exactly *util::end(rng) - util::begin(rng)* applications of the predicate.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form,

the parallel overload of *equal* requires *F* to meet the requirements of *CopyConstructible*. (deduced).

- *T*: The type of the new values to replace (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by *rng*. This is an unary predicate which returns *true* for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *new_value*: Refers to the new value to use as the replacement.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *replace* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_if* algorithm returns a *hpx::future<typename hpx::traits::range_iterator<Rng>::type>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy*. It returns *last*.

```
template<typename InIter, typename Sent, typename OutIter, typename T1, typename T2, typename Proj = hpx::traits::range_iterator<Rng>::type> replace_copy_result<InIter, OutIter> replace_copy(InIter first, Sent sent, OutIter dest,
T1 const &old_value, T2 const &new_value, Proj &&proj = Proj())
```

Copies all elements from the range [first, sent) to another range beginning at *dest* replacing all elements satisfying a specific criteria with *new_value*.

Effects: Assigns to every iterator *it* in the range [result, result + (sent - first)) either *new_value* or *(first + (it - result)) depending on whether the following corresponding condition holds: *(INVOKE(proj, *(first + (i - result))) == old_value)*

The assignments in the parallel *replace_copy* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly *sent - first* applications of the predicate.

Template Parameters

- *Iter*: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *Iter*.
- *OutIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *T1*: The type of the old value to replace (deduced).
- *T2*: The type of the new values to replace (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *sent*: Refers to the end of the sequence of elements the algorithm will be applied to.

- `dest`: Refers to the beginning of the destination range.
- `old_value`: Refers to the old value of the elements to replace.
- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The `replace_copy` algorithm returns an `in_out_result<InIter, OutIter>`. The `copy` algorithm returns the pair of the input iterator `last` and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename Rng, typename OutIter, typename T1, typename T2, typename Proj = hpx::parallel::util::projection>
replace_copy_result<typename hpx::traits::range_iterator<Rng>::type, OutIter> replace_copy(Rng &&rng,
OutIter dest, T1 const &old_value, T2 const &new_value, Proj &&proj = Proj())
```

Copies the all elements from the range `rgb` to another range beginning at `dest` replacing all elements satisfying a specific criteria with `new_value`.

Effects: Assigns to every iterator `it` in the range `[result, result + (util::end(rng) - util::begin(rng))]` either `new_value` or `*(first + (it - result))` depending on whether the following corresponding condition holds: `(INVOKE(proj, *(first + (i - result))) == old_value)`

The assignments in the parallel `replace_copy` algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly `util::end(rng) - util::begin(rng)` applications of the predicate.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `T1`: The type of the old value to replace (deduced).
- `T2`: The type of the new values to replace (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `old_value`: Refers to the old value of the elements to replace.
- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The `replace_copy` algorithm returns an `in_out_result<typename hpx::traits::range_iterator<Rng>::type, OutIter>`. The `copy` algorithm returns the pair of the input iterator `last` and the output

iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename T1, typename 'Proj'>
parallel::util::detail::algorithm_result<ExPolicy, replace_copy_result<FwdIter1, FwdIter2>>::type replace_copy(ExPolicy pol, first, sent, FwdIter1 dest, T1 old, T2 new, Proj proj) = {
    Proj proj{proj};
    return parallel::util::detail::algorithm_result<ExPolicy, replace_copy_result<FwdIter1, FwdIter2>>{pol, first, sent, dest, old, new, proj};
}
```

Copies the all elements from the range [*first*, *sent*] to another range beginning at *dest* replacing all elements satisfying a specific criteria with *new_value*.

Effects: Assigns to every iterator *it* in the range [*result*, *result* + (*sent* - *first*)) either *new_value* or *(*first* + (*i* - *result*)) depending on whether the following corresponding condition holds: `INVOKE(proj, *(first + (i - result))) == old_value`

The assignments in the parallel *replace_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *sent* - *first* applications of the predicate.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterator used (deduced). The iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *Iter*.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **T1**: The type of the old value to replace (deduced).
- **T2**: The type of the new values to replace (deduced).
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **sent**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **old_value**: Refers to the old value of the elements to replace.
- **new_value**: Refers to the new value to use as the replacement.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *replace_copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *replace_copy* algorithm returns a *hpx::future<in_out_result<FwdIter1, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *in_out_result<FwdIter1, FwdIter2>* otherwise. The *copy* algorithm returns the pair of the forward iterator *last* and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng, typename FwdIter, typename T1, typename T2, typename Proj = hpx::parallel::util::detail::algorithm_result<ExPolicy, replace_copy_result<typename hpx::traits::range_iterator<Rng>::type, FwdIter>>
```

Copies the all elements from the range *rbg* to another range beginning at *dest* replacing all elements satisfying a specific criteria with *new_value*.

Effects: Assigns to every iterator *it* in the range *[result, result + (util::end(rng) - util::begin(rng))]* either *new_value* or **(first + (it - result))* depending on whether the following corresponding condition holds: *(INVOKE(proj, *(first + (i - result))) == old_value)*

The assignments in the parallel *replace_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *util::end(rng) - util::begin(rng)* applications of the predicate.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *FwdIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *T1*: The type of the old value to replace (deduced).
- *T2*: The type of the new values to replace (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.

- `old_value`: Refers to the old value of the elements to replace.
- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel `replace_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `replace_copy` algorithm returns a `hpx::future<in_out_result< typename hpx::traits::range_iterator<Rng>::type, FwdIter>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `in_out_result< typename hpx::traits::range_iterator<Rng>::type, FwdIter>>`. The `copy` algorithm returns the pair of the input iterator `last` and the forward iterator to the element in the destination range, one past the last element copied.

```
template<typename InIter, typename Sent, typename OutIter, typename Pred, typename T, typename Proj = hpx::traits::projection_identity>
replace_copy_if<InIter, OutIter> replace_copy_if(InIter first, Sent sent, OutIter dest, Pred &&pred, T const &new_value, Proj &&proj = Proj())
```

Copies the all elements from the range [first, sent) to another range beginning at `dest` replacing all elements satisfying a specific criteria with `new_value`.

Effects: Assigns to every iterator it in the range [result, result + (sent - first)) either `new_value` or `*(first + (it - result))` depending on whether the following corresponding condition holds: `INVOKE(f, INVOKE(proj, *(first + (i - result)))) != false`

The assignments in the parallel `replace_copy_if` algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly `sent - first` applications of the predicate.

Template Parameters

- `InIter`: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.
- `Sent`: The type of the end iterators used (deduced). This sentinel type must be a sentinel for `InIter`.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `Pred`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `equal` requires `Pred` to meet the requirements of `CopyConstructible`. (deduced).
- `T`: The type of the new values to replace (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `sent`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `pred`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns `true` for the elements which need to replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced

and then implicitly converted to *Type*.

- *new_value*: Refers to the new value to use as the replacement.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *replace_copy_if* algorithm returns a *in_out_result*<*InIter*, *OutIter*>. The *replace_copy_if* algorithm returns the input iterator *last* and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename Rng, typename OutIter, typename Pred, typename T, typename Proj = hpx::parallel::util::project<range<Rng>, Pred>> replace_copy_if(Rng &&rng, OutIter dest, Pred &&pred, T const &new_value, Proj &&proj = Proj())
```

Copies the all elements from the range *rng* to another range beginning at *dest* replacing all elements satisfying a specific criteria with *new_value*.

Effects: Assigns to every iterator *it* in the range [*result*, *result* + (*util::end(rng)* - *util::begin(rng)*)) either *new_value* or *(*first* + (*i* - *result*)) depending on whether the following corresponding condition holds: *INVOKE(f, INVOKE(proj, *(*first* + (*i* - *result*)))) != false*

The assignments in the parallel *replace_copy_if* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly *util::end(rng)* - *util::begin(rng)* applications of the predicate.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *OutIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *Pred* to meet the requirements of *CopyConstructible*. (deduced).
- *T*: The type of the new values to replace (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [*first*, *last*). This is an unary predicate which returns *true* for the elements which need to replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects

passed to it. The type *Type* must be such that an object of type *FwdIter* can be dereferenced and then implicitly converted to *Type*.

- *new_value*: Refers to the new value to use as the replacement.
- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *replace_copy_if* algorithm returns an *in_out_result<typename hpx::traits::range_iterator<Rng>::type, OutIter>*. The *replace_copy_if* algorithm returns the input iterator *last* and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename Pred, typename parallel::util::detail::algorithm_result<ExPolicy, replace_copy_if_result<FwdIter1, FwdIter2>>::type replace_copy_if(
```

Copies the all elements from the range [first, sent) to another range beginning at *dest* replacing all elements satisfying a specific criteria with *new_value*.

Effects: Assigns to every iterator *it* in the range [result, result + (sent - first)) either *new_value* or *(first + (it - result)) depending on whether the following corresponding condition holds: *(INVOKE(f, INVOKE(proj, *(first + (i - result)))) != false*

The assignments in the parallel *replace_copy_if* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *sent - first* applications of the predicate.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterator used (deduced). The iterator type must meet the requirements of a forward iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *InIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *equal* requires *Pred* to meet the requirements of *CopyConstructible*. (deduced).
- *T*: The type of the new values to replace (deduced).

- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`
- Parameters**

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `sent`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `pred`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns `true` for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` invoked.

The assignments in the parallel `replace_copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `replace_copy_if` algorithm returns an `hpx::future<FwdIter1, FwdIter2>`. The `replace_copy_if` algorithm returns the input iterator `last` and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng, typename FwdIter, typename Pred, typename T, typename Proj = hpx::parallel::util::detail::algorithm_result<ExPolicy>, replace_copy_if_result<typename hpx::traits::range_iterator<Rng>::type>
```

Copies the all elements from the range `rng` to another range beginning at `dest` replacing all elements satisfying a specific criteria with `new_value`.

Effects: Assigns to every iterator `it` in the range `[result, result + (util::end(rng) - util::begin(rng))]` either `new_value` or `*(first + (it - result))` depending on whether the following corresponding condition holds: `INVOKE(f, INVOKE(proj, *(first + (i - result)))) != false`

The assignments in the parallel `replace_copy_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly `util::end(rng) - util::begin(rng)` applications of the predicate.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `Pred`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `equal` requires `Pred` to meet the requirements of `CopyConstructible`. (deduced).
- `T`: The type of the new values to replace (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `pred`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns `true` for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `is` invoked.

The assignments in the parallel `replace_copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `replace_copy_if` algorithm returns an `hpx::future<in_out_result<typename hpx::traits::range_iterator<Rng>::type, OutIter>>`. The `replace_copy_if` algorithm returns the input iterator `last` and the output iterator to the element in the destination range, one past the last element copied.

hpx/parallel/container_algorithms/reverse.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename Iter, typename Sent>
Iter reverse (Iter first, Sent last)
```

Reverses the order of the elements in the range [first, last). Behaves as if applying std::iter_swap to every pair of iterators first+i, (last-i) - 1 for each non-negative i < (last-first)/2.

The assignments in the parallel *reverse* algorithm execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- *Iter*: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *Iter*.

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to the end of the sequence of elements the algorithm will be applied to.

Return The *reverse* algorithm returns a *Iter*. It returns *last*.

```
template<typename Rng>
hpx::traits::range_iterator<Rng>::type reverse (Rng &&rng)
```

Uses *rng* as the source range, as if using *util::begin(rng)* as *first* and *ranges::end(rng)* as *last*. Reverses the order of the elements in the range [first, last). Behaves as if applying std::iter_swap to every pair of iterators first+i, (last-i) - 1 for each non-negative i < (last-first)/2.

The assignments in the parallel *reverse* algorithm execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a bidirectional iterator.

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.

Return The *reverse* algorithm returns a *hpx::traits::range_iterator<Rng>::type*. It returns *last*.

```
template<typename ExPolicy, typename Iter, typename Sent>
parallel::util::detail::algorithm_result<ExPolicy, Iter>::type reverse (ExPolicy &&policy, Iter
first, Sent last)
```

Reverses the order of the elements in the range [first, last). Behaves as if applying std::iter_swap to every pair of iterators first+i, (last-i) - 1 for each non-negative i < (last-first)/2.

The assignments in the parallel *reverse* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Iter*: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.

- **Sent:** The type of the end iterators used (deduced). This sentinel type must be a sentinel for **Iter**.

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last:** Refers to the end of the sequence of elements the algorithm will be applied to.

The assignments in the parallel *reverse* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *reverse* algorithm returns a *hpx::future<Iter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *Iter* otherwise. It returns *last*.

```
template<typename ExPolicy, typename Rng>
parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type reverse (ExPolicy && icy, Rng &&)
```

Uses *rng* as the source range, as if using *util::begin(rng)* as *first* and *ranges::end(rng)* as *last*. Reverses the order of the elements in the range [first, last). Behaves as if applying *std::iter_swap* to every pair of iterators *first+i*, *(last-i) - 1* for each non-negative *i* < (*last-first*) / 2.

The assignments in the parallel *reverse* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a bidirectional iterator.

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **rng:** Refers to the sequence of elements the algorithm will be applied to.

The assignments in the parallel *reverse* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *reverse* algorithm returns a *hpx::future<typename hpx::traits::range_iterator<Rng>::type>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *hpx::future< typename hpx::traits::range_iterator<Rng>::type>* otherwise. It returns *last*.

```
template<typename Iter, typename Sent, typename OutIter>
reverse_copy_result<Iter, OutIter> reverse_copy (Iter first, Sent last, OutIter result)
```

Copies the elements from the range [first, last) to another range beginning at *result* in such a way that the elements in the new range are in reverse order. Behaves as if by executing the assignment **(result + (last - first) - 1 - i) = *(first + i)* once for each non-negative *i* < (*last - first*). If the source and destination ranges (that is, [first, last) and [result, *result+(last-first)*) respectively) overlap, the behavior is undefined.

The assignments in the parallel *reverse_copy* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **Iter**: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for **Iter**.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **result**: Refers to the begin of the destination range.

Return The *reverse_copy* algorithm returns a *reverse_copy_result<Iter, OutIter>*. The *reverse_copy* algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename Rng, typename OutIter>
ranges::reverse_copy_result<typename hpx::traits::range_iterator<Rng>::type, OutIter> reverse_copy(Rng
&&rng,
Out-
Iter
re-
sult)
```

Uses *rng* as the source range, as if using *util::begin(rng)* as *first* and *ranges::end(rng)* as *last*. Copies the elements from the range [*first*, *last*] to another range beginning at *result* in such a way that the elements in the new range are in reverse order. Behaves as if by executing the assignment $*(\text{result} + (\text{last} - \text{first}) - 1 - i) = *(first + i)$ once for each non-negative $i < (\text{last} - \text{first})$. If the source and destination ranges (that is, [*first*, *last*) and [*result*, *result*+(*last*-*first*)) overlap, the behavior is undefined.

The assignments in the parallel *reverse_copy* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a bidirectional iterator.
- **OutputIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **result**: Refers to the begin of the destination range.

Return The *reverse_copy* algorithm returns a *ranges::reverse_copy_result< typename hpx::traits::range_iterator<Rng>::type, OutIter>>::type*. The *reverse_copy* algorithm returns an object equal to {*last*, *result* + *N*} where *N* = *last* - *first*

```
template<typename ExPolicy, typename Iter, typename Sent, typename OutIter>
```

```
parallel::util::detail::algorithm_result<ExPolicy, reverse_copy_result<Iter, OutIter>>::type reverse_copy(ExPolicy
&&policy,
Iter,
first,
Sent,
last,
OutIter
re-
sult)
```

Copies the elements from the range [first, last) to another range beginning at result in such a way that the elements in the new range are in reverse order. Behaves as if by executing the assignment *(result + (last - first) - 1 - i) = *(first + i) once for each non-negative i < (last - first). If the source and destination ranges (that is, [first, last) and [result, result+(last-first)) respectively) overlap, the behavior is undefined.

The assignments in the parallel *reverse_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Iter**: The type of the source iterator used (deduced). The iterator type must meet the requirements of an input iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for Iter.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **result**: Refers to the begin of the destination range.

The assignments in the parallel *reverse_copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *reverse_copy* algorithm returns a *hpx::future<reverse_copy_result<Iter, OutIter>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *reverse_copy_result<Iter, OutIter>* otherwise. The *reverse_copy* algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng, typename OutIter>
```

`util::detail::algorithm_result<ExPolicy, ranges::reverse_copy_result<typename hpx::traits::range_iterator<Rng>::type, Ov`

Uses `rng` as the source range, as if using `util::begin(rng)` as `first` and `ranges::end(rng)` as `last`. Copies the elements from the range `[first, last)` to another range beginning at `result` in such a way that the elements in the new range are in reverse order. Behaves as if by executing the assignment $*(\text{result} + (\text{last} - \text{first}) - 1 - i) = *(first + i)$ once for each non-negative $i < (\text{last} - \text{first})$. If the source and destination ranges (that is, `[first, last)` and `[result, result+(last-first))` respectively) overlap, the behavior is undefined.

The assignments in the parallel `reverse_copy` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly `last - first` assignments.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a bidirectional iterator.
- `OutputIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `result`: Refers to the begin of the destination range.

The assignments in the parallel `reverse_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `reverse_copy` algorithm returns a `hpx::future<ranges::reverse_copy_result< typename hpx::traits::range_iterator<Rng>::type, OutIter>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::reverse_copy_result< typename hpx::traits::range_iterator<Rng>::type, OutIter>` otherwise. The `reverse_copy` algorithm returns an object equal to `{last, result + N}` where $N = \text{last} - \text{first}$

hpx/parallel/container_algorithms/rotate.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

```
template<typename FwdIter, typename Sent>
subrange_t<FwdIter, Sent> rotate (FwdIter first, FwdIter middle, Sent last)
    Performs a left rotation on a range of elements. Specifically, rotate swaps the elements in the range [first, last) in such a way that the element middle becomes the first element of the new range and middle - 1 becomes the last element.
```

The assignments in the parallel *rotate* algorithm execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for **FwdIter**.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

Note The type of dereferenced *FwdIter* must meet the requirements of *MoveAssignable* and *MoveConstructible*.

Return The *rotate* algorithm returns a *subrange_t<FwdIter, Sent>*. The *rotate* algorithm returns the iterator equal to pair(first + (last - middle), last).

```
template<typename ExPolicy, typename FwdIter, typename Sent>
parallel::util::detail::algorithm_result<ExPolicy, subrange_t<FwdIter, Sent>>::type rotate (ExPolicy
    &&policy,
    FwdIter
    first,
    FwdIter
    mid-
    dle,
    Sent
    last)
```

Performs a left rotation on a range of elements. Specifically, *rotate* swaps the elements in the range [first, last) in such a way that the element middle becomes the first element of the new range and middle - 1 becomes the last element.

The assignments in the parallel *rotate* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for **FwdIter**.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

The assignments in the parallel *rotate* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced *FwdIter* must meet the requirements of *MoveAssignable* and *MoveConstructible*.

Return The *rotate* algorithm returns a *hpx::future<subrange_t<FwdIter, Sent>>* if the execution policy is of type *parallel_task_policy* and returns a *subrange_t<FwdIter, Sent>* otherwise. The *rotate* algorithm returns the iterator equal to pair(*first + (last - middle)*, *last*).

```
template<typename Rng>
subrange_t<hpx::traits::range_iterator_t<Rng>, hpx::traits::range_iterator_t<Rng>> rotate (Rng
&&rng,
hpx::traits::range_iterator_t<
mid-
dle>)
```

Uses *rng* as the source range, as if using *util::begin(rng)* as *first* and *ranges::end(rng)* as *last*. Performs a left rotation on a range of elements. Specifically, *rotate* swaps the elements in the range [*first*, *last*] in such a way that the element *middle* becomes the first element of the new range and *middle - 1* becomes the last element.

The assignments in the parallel *rotate* algorithm execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.

Parameters

- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.

Note The type of dereferenced *FwdIter* must meet the requirements of *MoveAssignable* and *MoveConstructible*.

Return The *rotate* algorithm returns a *subrange_t<hpx::traits::range_iterator_t<Rng>, hpx::traits::range_iterator_t<Rng>>*. The *rotate* algorithm returns the iterator equal to pair(*first + (last - middle)*, *last*).

```
template<typename ExPolicy, typename Rng>
util::detail::algorithm_result<ExPolicy, subrange_t<hpx::traits::range_iterator_t<Rng>, hpx::traits::range_iterator_t<Rng>>>
```

Uses *rng* as the source range, as if using *util::begin(rng)* as *first* and *ranges::end(rng)* as *last*. Performs a left rotation on a range of elements. Specifically, *rotate* swaps the elements in the range [*first*, *last*] in such a way that the element *middle* becomes the first element of the new range and *middle - 1* becomes the last element.

The assignments in the parallel *rotate* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.

Parameters

- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **policy**: The execution policy to use for the scheduling of the iterations.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.

The assignments in the parallel *rotate* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced *FwdIter* must meet the requirements of *MoveAssignable* and *MoveConstructible*.

Return The *rotate* algorithm returns a *hpx::future* <*subrange_t*<*hpx::traits::range_iterator_t*<*Rng*>, *hpx::traits::range_iterator_t*<*Rng*>>> if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *subrange_t*<*hpx::traits::range_iterator_t*<*Rng*>, *hpx::traits::range_iterator_t*<*Rng*>>. otherwise. The *rotate* algorithm returns the iterator equal to pair(*first* + (*last* - *middle*), *last*).

```
template<typename FwdIter, typename Sent, typename OutIter>
rotate_copy_result<FwdIter, OutIter> rotate_copy (FwdIter first, FwdIter middle, Sent last,
                                                OutIter dest_first)
```

Copies the elements from the range [*first*, *last*), to another range beginning at *dest_first* in such a way, that the element *middle* becomes the first element of the new range and *middle* - 1 becomes the last element.

The assignments in the parallel *rotate_copy* algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last* - *first* assignments.

Template Parameters

- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the end iterators used (deduced). This sentinel type must be a sentinel for *FwdIter*.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

Return The *rotate_copy* algorithm returns a *rotate_copy_result*<*FwdIter*, *OutIter*>. The *rotate_copy* algorithm returns the output iterator to the element past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2>
```

```
parallel::util::detail::algorithm_result<ExPolicy, rotate_copy_result<FwdIter1, FwdIter2>>::type rotate_copy (ExPolicy  
&pol-  
icy,  
FwdIter  
first,  
FwdIter  
mid-  
dle,  
Sent  
last,  
FwdIter  
dest_fir
```

Copies the elements from the range [first, last), to another range beginning at *dest_first* in such a way, that the element *middle* becomes the first element of the new range and *middle* - 1 becomes the last element.

The assignments in the parallel `rotate_copy` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
 - `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
 - `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
 - `Sent`: The type of the end iterators used (deduced). This sentinel type must be a sentinel for `FwdIter`.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
 - **middle**: Refers to the element that should appear at the beginning of the rotated range.
 - **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

The assignments in the parallel `rotate_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `rotate_copy` algorithm returns a returns `hpx::future<rotate_copy_result<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `rotate_copy_result<FwdIter1, FwdIter2>` otherwise. The `rotate_copy` algorithm returns the output iterator to the element past the last element copied.

Uses *rng* as the source range, as if using `util::begin(rng)` as *first* and `ranges::end(rng)` as *last*. Copies the elements from the range [*first*, *last*), to another range beginning at *dest_first* in such a way, that the element *middle* becomes the first element of the new range and *middle* - 1 becomes the last element.

The assignments in the parallel `rotate_copy` algorithm execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters

- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.
- **dest_first**: Refers to the begin of the destination range.

Return The *rotate* algorithm returns a *rotate_copy_result*<*hpx::traits::range_iterator_t<Rng>*, *OutIter*>. The *rotate_copy* algorithm returns the output iterator to the element past the last element copied.

```
template<typename ExPolicy, typename Rng, typename OutIter>
parallel::util::detail::algorithm_result<ExPolicy, rotate_copy_result<hpx::traits::range_iterator_t<Rng>, OutIter>>::type rc
```

Uses *rng* as the source range, as if using *util::begin(rng)* as *first* and *ranges::end(rng)* as *last*. Copies the elements from the range [first, last), to another range beginning at *dest_first* in such a way, that the element *new_first* becomes the first element of the new range and *new_first* - 1 becomes the last element.

The assignments in the parallel *rotate_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.
- **dest_first**: Refers to the begin of the destination range.

The assignments in the parallel *rotate_copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *rotate_copy* algorithm returns a *hpx::future<rotate_copy_result<*

`hpx::traits::range_iterator_t<Rng>, OutIter>>` if the execution policy is of type `parallel_task_policy` and returns `rotate_copy_result< hpx::traits::range_iterator_t<Rng>, OutIter>` otherwise. The `rotate_copy` algorithm returns the output iterator to the element past the last element copied.

`hpx/parallel/container_algorithms/search.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter, typename Sent, typename FwdIter2, typename Sent2, typename Pred = hpx::ranges::FwdIter search(FwdIter first, Sent last, FwdIter2 s_first, Sent2 s_last, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Searches the range [first, last) for any elements in the range [s_first, s_last). Uses a provided predicate to compare elements.

The comparison operations in the parallel `search` algorithm execute in sequential order in the calling thread.

Note Complexity: at most (S^*N) comparisons where $S = \text{distance}(s_{\text{first}}, s_{\text{last}})$ and $N = \text{distance}(\text{first}, \text{last})$.

Template Parameters

- `FwdIter`: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel used for the first range (deduced). This iterator type must meet the requirements of an sentinel.
- `FwdIter2`: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent2`: The type of the source sentinel used for the second range (deduced). This iterator type must meet the requirements of an sentinel.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `adjacent_find` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- `Proj1`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter`.
- `Proj2`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter2`.

Parameters

- `first`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `s_first`: Refers to the beginning of the sequence of elements the algorithm will be searching for.
- `s_last`: Refers to the end of the sequence of elements of the algorithm will be searching for.
- `op`: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `FwdIter1` as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `FwdIter2` as a projection operation before the actual predicate *is* invoked.

Return The `search` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `search` algorithm returns an iterator to the beginning of the first subsequence $[s_first, s_last)$ in range $[first, last)$. If the length of the subsequence $[s_first, s_last)$ is greater than the length of the range $[first, last)$, `last` is returned. Additionally if the size of the subsequence is empty `first` is returned. If no subsequence is found, `last` is returned.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename FwdIter2, typename Sent2, typename util::detail::algorithm_result<ExPolicy, FwdIter>::type search (ExPolicy &&policy, FwdIter first,
                                                               Sent last, FwdIter2 s_first, Sent2
                                                               s_last, Pred &&op = Pred(),
                                                               Proj1 &&proj1 = Proj1(), Proj2
                                                               &&proj2 = Proj2())
```

Searches the range $[first, last)$ for any elements in the range $[s_first, s_last)$. Uses a provided predicate to compare elements.

The comparison operations in the parallel `search` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: at most (S^*N) comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{distance}(first, last)$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel used for the first range (deduced). This iterator type must meet the requirements of an sentinel.
- `FwdIter2`: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent2`: The type of the source sentinel used for the second range (deduced). This iterator type must meet the requirements of an sentinel.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `adjacent_find` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- `Proj1`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter`.
- `Proj2`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter2`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.

- `first`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `s_first`: Refers to the beginning of the sequence of elements the algorithm will be searching for.
- `s_last`: Refers to the end of the sequence of elements of the algorithm will be searching for.
- `op`: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `FwdIter1` as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `FwdIter2` as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel `search` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `search` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `search` algorithm returns an iterator to the beginning of the first subsequence $[s_first, s_last)$ in range $[first, last)$. If the length of the subsequence $[s_first, s_last)$ is greater than the length of the range $[first, last)$, `last` is returned. Additionally if the size of the subsequence is empty `first` is returned. If no subsequence is found, `last` is returned.

```
template<typename Rng1, typename Rng2, typename Pred = hpx::ranges::equal_to, typename Proj1 = hpx::parallel::util::traits::range_iterator<Rng1>::type> search(Rng1 &&rng1, Rng2 &&rng2, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Searches the range $[first, last)$ for any elements in the range $[s_first, s_last)$. Uses a provided predicate to compare elements.

The comparison operations in the parallel `search` algorithm execute in sequential order in the calling thread.

Note Complexity: at most (S^*N) comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{distance}(first, last)$.

Template Parameters

- `Rng1`: The type of the examine range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Rng2`: The type of the search range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `adjacent_find` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- `Proj1`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng1`.

- `Proj2`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng2`.

Parameters

- `rng1`: Refers to the sequence of elements the algorithm will be examining.
- `rng2`: Refers to the sequence of elements the algorithm will be searching for.
- `op`: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of `rng1` as a projection operation before the actual predicate *is* invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of `rng2` as a projection operation before the actual predicate *is* invoked.

Return The `search` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `search` algorithm returns an iterator to the beginning of the first subsequence $[s_first, s_last]$ in range $[first, last]$. If the length of the subsequence $[s_first, s_last]$ is greater than the length of the range $[first, last]$, `last` is returned. Additionally if the size of the subsequence is empty `first` is returned. If no subsequence is found, `last` is returned.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = hpx::ranges::equal_to, typename Pr hpx::parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng1>::type>::type search(
```

Searches the range $[first, last]$ for any elements in the range $[s_first, s_last]$. Uses a provided predicate to compare elements.

The comparison operations in the parallel `search` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: at most $(S*N)$ comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{distance}(first, last)$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in

which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- *Rng1*: The type of the examine range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the search range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *adjacent_find* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj1*: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of *Rng1*.
- *Proj2*: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of *Rng2*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the sequence of elements the algorithm will be examining.
- *rng2*: Refers to the sequence of elements the algorithm will be searching for.
- *op*: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and *FwdIter2* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of *rng1* as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of *rng2* as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *search* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *search* algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type *task_execution_policy* and returns *FwdIter* otherwise. The *search* algorithm returns an iterator to the beginning of the first subsequence $[s_first, s_last)$ in range $[first, last)$. If the length of the subsequence $[s_first, s_last)$ is greater than the length of the range $[first, last)$, *last* is returned. Additionally if the size of the subsequence is empty *first* is returned. If no subsequence is found, *last* is returned.

```
template<typename FwdIter, typename FwdIter2, typename Sent2, typename Pred = hpx::ranges::equal_to, typename FwdIter search_n(ExPolicy &&policy, FwdIter first, std::size_t count, FwdIter2 s_first, Sent s_last, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Searches the range $[first, last)$ for any elements in the range $[s_first, s_last)$. Uses a provided predicate to compare elements.

The comparison operations in the parallel *search_n* algorithm execute in sequential order in the calling thread.

Note Complexity: at most $(S * N)$ comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{count}$.

Template Parameters

- *FwdIter*: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.

- *FwdIter2*: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent2*: The type of the source sentinel used for the second range (deduced). This iterator type must meet the requirements of an sentinel.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *adjacent_find* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`

Parameters

- *first*: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- *count*: Refers to the range of elements of the first range the algorithm will be applied to.
- *s_first*: Refers to the beginning of the sequence of elements the algorithm will be searching for.
- *s_last*: Refers to the end of the sequence of elements of the algorithm will be searching for.
- *op*: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and *FwdIter2* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced *FwdIter1* as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced *FwdIter2* as a projection operation before the actual predicate *is* invoked.

Return The *search_n* algorithm returns *FwdIter*. The *search_n* algorithm returns an iterator to the beginning of the last subsequence $[s_first, s_last]$ in range $[first, first+count]$. If the length of the subsequence $[s_first, s_last]$ is greater than the length of the range $[first, first+count]$, *first* is returned. Additionally if the size of the subsequence is empty or no subsequence is found, *first* is also returned.

```
template<typename ExPolicy, typename FwdIter, typename FwdIter2, typename Sent2, typename Pred = hpx::rangeutil::detail::algorithm_result<ExPolicy, FwdIter>::type search_n (ExPolicy &&policy, FwdIter first, std::size_t count, FwdIter2 s_first, Sent2 s_last, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Searches the range $[first, last]$ for any elements in the range $[s_first, s_last]$. Uses a provided predicate to compare elements.

The comparison operations in the parallel *search_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: at most $(S*N)$ comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{count}$.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.

- *FwdIter2*: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent2*: The type of the source sentinel used for the second range (deduced). This iterator type must meet the requirements of an sentinel.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *adjacent_find* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- *count*: Refers to the range of elements of the first range the algorithm will be applied to.
- *s_first*: Refers to the beginning of the sequence of elements the algorithm will be searching for.
- *s_last*: Refers to the end of the sequence of elements of the algorithm will be searching for.
- *op*: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and *FwdIter2* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced *FwdIter1* as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced *FwdIter2* as a projection operation before the actual predicate *is* invoked.

The comparison operations in the parallel *search_n* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *search_n* algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type *task_execution_policy* and returns *FwdIter* otherwise. The *search_n* algorithm returns an iterator to the beginning of the last subsequence $[s_first, s_last)$ in range $[first, first+count)$. If the length of the subsequence $[s_first, s_last)$ is greater than the length of the range $[first, first+count)$, *first* is returned. Additionally if the size of the subsequence is empty or no subsequence is found, *first* is also returned.

```
template<typename Rng1, typename Rng2, typename Pred = hpx::ranges::equal_to, typename Proj1 = hpx::parallel::traits::range_iterator<Rng1>::type>
search_n(Rng1 &&rng1, std::size_t count, Rng2 &&rng2, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Searches the range $[first, last)$ for any elements in the range $[s_first, s_last)$. Uses a provided predicate to compare elements.

The comparison operations in the parallel *search* algorithm execute in sequential order in the calling thread.

Note Complexity: at most $(S*N)$ comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{distance}(first, last)$.

Template Parameters

- *Rng1*: The type of the examine range used (deduced). The iterators extracted from this range

type must meet the requirements of an input iterator.

- *Rng2*: The type of the search range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *adjacent_find* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj1*: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of *Rng1*.
- *Proj2*: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of *Rng2*.

Parameters

- *rng1*: Refers to the sequence of elements the algorithm will be examining.
- *count*: The number of elements to apply the algorithm on.
- *rng2*: Refers to the sequence of elements the algorithm will be searching for.
- *op*: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter1* and *FwdIter2* can be dereferenced and then implicitly converted to *Type1* and *Type2* respectively

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of *rng1* as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of *rng2* as a projection operation before the actual predicate *is* invoked.

Return The *search* algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns *FwdIter* otherwise. The *search* algorithm returns an iterator to the beginning of the first subsequence $[s_first, s_last)$ in range $[first, last]$. If the length of the subsequence $[s_first, s_last)$ is greater than the length of the range $[first, last)$, *last* is returned. Additionally if the size of the subsequence is empty *first* is returned. If no subsequence is found, *last* is returned.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = hpx::ranges::equal_to, typename Pr
```

`hpx::parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng1>::type>::type search_`

Searches the range [first, last) for any elements in the range [s_first, s_last). Uses a provided predicate to compare elements.

The comparison operations in the parallel *search* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: at most (S^*N) comparisons where $S = \text{distance}(s_first, s_last)$ and $N = \text{distance}(first, last)$.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng1`: The type of the examine range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Rng2`: The type of the search range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *adjacent_find* requires `Pred` to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- `Proj1`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng1`.
- `Proj2`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng2`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the sequence of elements the algorithm will be examining.
- `count`: The number of elements to apply the algorithm on.
- `rng2`: Refers to the sequence of elements the algorithm will be searching for.
- `op`: Refers to the binary predicate which returns true if the elements should be treated as equal. the signature of the function should be equivalent to

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of `rng1` as a projection operation before the actual predicate `is` invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of `rng2` as a projection operation before the actual predicate `is` invoked.

The comparison operations in the parallel `search` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `search` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `search` algorithm returns an iterator to the beginning of the first subsequence $[s_first, s_last)$ in range $[first, last)$. If the length of the subsequence $[s_first, s_last)$ is greater than the length of the range $[first, last)$, `last` is returned. Additionally if the size of the subsequence is empty `first` is returned. If no subsequence is found, `last` is returned.

hpx/parallel/container_algorithms/set_difference.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename It
```

```
util::detail::algorithm_result<ExPolicy, ranges::set_difference_result<Iter1, Iter3>>::type set_difference (ExPolicy  
  &&policy,  
  Iter1  
  first1,  
  Sent1  
  last1,  
  Iter2  
  first2,  
  Sent2  
  last2,  
  Iter3  
  dest,  
  Pred  
  &&op  
  =  
  Pred(),  
  Proj1  
  &&proj1  
  =  
  Proj1(),  
  Proj2  
  &&proj2  
  =  
  Proj2())
```

Constructs a sorted range beginning at *dest* consisting of all elements present in the range [*first1*, *last1*] and not present in the range [*first2*, *last2*). This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

Equivalent elements are treated individually, that is, if some element is found *m* times in [*first1*, *last1*] and *n* times in [*first2*, *last2*), it will be copied to *dest* exactly `std::max(m-n, 0)` times. The resulting range cannot overlap with either of the input ranges.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where *N1* is the length of the first sequence and *N2* is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Iter1**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent1**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for *Iter1*.
- **Iter2**: The type of the source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent2**: The type of the end source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an sentinel for *Iter2*.
- **Iter3**: The type of the iterator representing the destination range (deduced). This iterator

type must meet the requirements of an output iterator.

- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *set_difference* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first1*: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- *last1*: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- *first2*: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- *last2*: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *op*: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *InIter* can be dereferenced and then implicitly converted to *Type1*

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate *op* is invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate *op* is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *set_difference* algorithm returns a `hpx::future<ranges::set_difference_result<Iter1, Iter3>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `ranges::set_difference_result<Iter1, Iter3>` otherwise. The *set_difference* algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Iter3, typename Pred = detail::less, typename
```

```
util::detail::algorithm_result<ExPolicy, ranges::set_difference_result<typename traits::range_iterator<Rng1>::type, Iter3>
```

Constructs a sorted range beginning at *dest* consisting of all elements present in the range [first1, last1) and not present in the range [first2, last2). This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

Equivalent elements are treated individually, that is, if some element is found *m* times in [first1, last1) and *n* times in [first2, last2), it will be copied to *dest* exactly std::max(m-n, 0) times. The resulting range cannot overlap with either of the input ranges.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where *N1* is the length of the first sequence and *N2* is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Iter3*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *set_difference* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to std::less<>
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to *util::projection_identity*
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to *util::projection_identity*

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the first sequence of elements the algorithm will be applied to.
- `rng2`: Refers to the second sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `op`: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `set_difference` algorithm returns a `hpx::future<ranges::set_difference_result<Iter1, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::set_difference_result<Iter1, Iter3>` otherwise. where `Iter1` is `range_iterator_t<Rng1>` and `Iter2` is `range_iterator_t<Rng2>`. The `set_difference` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

[hpx/parallel/container_algorithms/set_intersection.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3>
util::detail::algorithm_result<ExPolicy, ranges::set_intersection_result<Iter1, Iter2, Iter3>>::type set_intersection(
```

Constructs a sorted range beginning at dest consisting of all elements present in both sorted ranges [first1, last1) and [first2, last2). This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

If some element is found *m* times in [first1, last1) and *n* times in [first2, last2), the first std::min(m, n) elements will be copied from the first range to the destination range. The order of equivalent elements is preserved. The resulting range cannot overlap with either of the input ranges.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where *N1* is the length of the first sequence and *N2* is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Iter1:** The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent1:** The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Iter2:** The type of the source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an forward iterator.

- `Sent2`: The type of the end source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an sentinel for `Iter2`.
- `Iter3`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `set_intersection` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj1`: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `first2`: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- `last2`: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `op`: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `set_intersection` algorithm returns a `hpx::future<ranges::set_intersection_result<Iter1, Iter2, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::set_intersection_result<Iter1, Iter2, Iter3>` otherwise. The `set_intersection` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Iter3, typename Pred = detail::less, typename Op = detail::equal>
```

```
util::detail::algorithm_result<ExPolicy, ranges::set_intersection_result<typename traits::range_iterator<Rng1>::type, typename traits::range_iterator<Rng2>::type>
```

Constructs a sorted range beginning at dest consisting of all elements present in both sorted ranges [first1, last1) and [first2, last2). This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

If some element is found *m* times in [first1, last1) and *n* times in [first2, last2), the first std::min(m, n) elements will be copied from the first range to the destination range. The order of equivalent elements is preserved. The resulting range cannot overlap with either of the input ranges.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where *N1* is the length of the first sequence and *N2* is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Iter3*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *set_intersection* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the first sequence of elements the algorithm will be applied to.
- `rng2`: Refers to the second sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `op`: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `set_intersection` algorithm returns a `hpx::future<ranges::set_intersection_result<Iter1, Iter2, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::set_intersection_result<Iter1, Iter2, Iter3>` otherwise. where `Iter1` is `range_iterator_t<Rng1>` and `Iter2` is `range_iterator_t<Rng2>`. The `set_intersection` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

hpx/parallel/container_algorithms/set_symmetric_difference.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3>
util::detail::algorithm_result<ExPolicy, ranges::set_symmetric_difference_result<Iter1, Iter2, Iter3>>::type set_symmetric_difference(
```

Constructs a sorted range beginning at *dest* consisting of all elements present in either of the sorted ranges [*first1*, *last1*) and [*first2*, *last2*), but not in both of them are copied to the range beginning at *dest*. The resulting range is also sorted. This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

If some element is found *m* times in [*first1*, *last1*) and *n* times in [*first2*, *last2*), it will be copied to *dest* exactly $\text{std::abs}(m-n)$ times. If *m*>*n*, then the last *m-n* of those elements are copied from [*first1*,*last1*), otherwise the last *n-m* elements are copied from [*first2*,*last2*). The resulting range cannot overlap with either of the input ranges.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where *N1* is the length of the first sequence and *N2* is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Iter1:** The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent1:** The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.

- `Iter2`: The type of the source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an forward iterator.
- `Sent2`: The type of the end source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an sentinel for `Iter2`.
- `Iter3`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `set_symmetric_difference` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj1`: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `first2`: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- `last2`: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `op`: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `set_symmetric_difference` algorithm returns a `hpx::future<ranges::set_symmetric_difference_result<Iter1, Iter2, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::set_symmetric_difference_result<Iter1, Iter2, Iter3>` otherwise. The `set_symmetric_difference` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Iter3, typename Pred = detail::less, typename
```

```
util::detail::algorithm_result<ExPolicy, ranges::set_symmetric_difference_result<typename traits::range_iterator<Rng1>::type, typename traits::range_iterator<Rng2>::type> set_symmetric_difference(
```

Constructs a sorted range beginning at *dest* consisting of all elements present in either of the sorted ranges [first1, last1) and [first2, last2), but not in both of them are copied to the range beginning at *dest*. The resulting range is also sorted. This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

If some element is found *m* times in [first1, last1) and *n* times in [first2, last2), it will be copied to *dest* exactly std::abs(*m*-*n*) times. If *m*>*n*, then the last *m*-*n* of those elements are copied from [first1, last1), otherwise the last *n*-*m* elements are copied from [first2, last2). The resulting range cannot overlap with either of the input ranges.

Note Complexity: At most $2*(N_1 + N_2 - 1)$ comparisons, where N_1 is the length of the first sequence and N_2 is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Iter3*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *set_symmetric_difference* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to std::less<>
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to *util::projection_identity*

- `Proj2`: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the first sequence of elements the algorithm will be applied to.
- `rng2`: Refers to the second sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `op`: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `Inter` can be dereferenced and then implicitly converted to `Type1`

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `set_symmetric_difference` algorithm returns a `hpx::future<ranges::set_symmetric_difference_result<Iter1, Iter2, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::set_symmetric_difference_result<Iter1, Iter2, Iter3>` otherwise. where `Iter1` is `range_iterator_t<Rng1>` and `Iter2` is `range_iterator_t<Rng2>` The `set_symmetric_difference` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

hpx/parallel/container_algorithms/set_union.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename ExPolicy, typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3, typename Sent3hpx::parallel::container_algorithms::set_union(ExPolicy policy, Iter1 first1, Sent1 last1, Iter2 first2, Sent2 last2, Iter3 dest, Op op, Proj1 proj1 = util::projection_identity, Proj2 proj2 = util::projection_identity)
```

```
util::detail::algorithm_result<ExPolicy, ranges::set_union_result<Iter1, Iter2, Iter3>>::type set_union(ExPolicy  
    &&pol-  
    icy,  
    Iter1  
    first1,  
    Sent1  
    last1,  
    Iter2  
    first2,  
    Sent2  
    last2,  
    Iter3  
    dest,  
    Pred  
    &&op  
    =  
    Pred(),  
    Proj1  
    &&proj1  
    =  
    Proj1(),  
    Proj2  
    &&proj2  
    =  
    Proj2())
```

Constructs a sorted range beginning at dest consisting of all elements present in one or both sorted ranges [first1, last1) and [first2, last2). This algorithm expects both input ranges to be sorted with the given binary predicate f.

If some element is found m times in [first1, last1) and n times in [first2, last2), then all m elements will be copied from [first1, last1) to dest, preserving order, and then exactly std::max(n-m, 0) elements will be copied from [first2, last2) to dest, also preserving order.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where $N1$ is the length of the first sequence and $N2$ is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **Iter1:** The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent1:** The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Iter2:** The type of the source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an forward iterator.
- **Sent2:** The type of the end source iterators used (deduced) representing the second sequence. This iterator type must meet the requirements of an sentinel for Iter2.
- **Iter3:** The type of the iterator representing the destination range (deduced). This iterator

type must meet the requirements of an output iterator.

- *Pred*: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *set_union* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::less<>`
- *Proj1*: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- *Proj2*: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first1*: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- *last1*: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- *first2*: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- *last2*: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *op*: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type *Type1* must be such that objects of type *InIter* can be dereferenced and then implicitly converted to *Type1*

- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate *op* is invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate *op* is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *set_union* algorithm returns a `hpx::future<ranges::set_union_result<Iter1, Iter2, Iter3>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `ranges::set_union_result<Iter1, Iter2, Iter3>` otherwise. The *set_union* algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Iter3, typename Pred = detail::less, typename
```

`util::detail::algorithm_result<ExPolicy, ranges::set_union_result<typename traits::range_iterator<Rng1>::type, typename traits::range_iterator<Rng2>::type>`

Constructs a sorted range beginning at dest consisting of all elements present in one or both sorted ranges [first1, last1) and [first2, last2). This algorithm expects both input ranges to be sorted with the given binary predicate *f*.

If some element is found *m* times in [first1, last1) and *n* times in [first2, last2), then all *m* elements will be copied from [first1, last1) to dest, preserving order, and then exactly std::max(n-m, 0) elements will be copied from [first2, last2) to dest, also preserving order.

Note Complexity: At most $2*(N1 + N2 - 1)$ comparisons, where *N1* is the length of the first sequence and *N2* is the length of the second sequence.

The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (*sequenced_policy*) or in a single new thread spawned from the current thread (for *sequenced_task_policy*).

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Rng2`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Iter3`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `set_union` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- `Proj1`: The type of an optional projection function applied to the first sequence. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function applied to the second sequence. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the first sequence of elements the algorithm will be applied to.
- `rng2`: Refers to the second sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `op`: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `op` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `op` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `set_union` algorithm returns a `hpx::future<ranges::set_union_result<Iter1, Iter2, Iter3>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `ranges::set_union_result<Iter1, Iter2, Iter3>` otherwise. where `Iter1` is `range_iterator_t<Rng1>` and `Iter2` is `range_iterator_t<Rng2>` The `set_union` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

hpx/parallel/container_algorithms/shift_left.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter, typename Sent, typename Size>
FwdIter shift_left(FwdIter first, Sent last, Size n)
```

Shifts the elements in the range $[first, last)$ by n positions towards the beginning of the range. For every integer i in $[0, last - first]$

- n), moves the element originally at position $first + n + i$ to position $first + i$.

The assignment operations in the parallel `shift_left` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: At most $(last - first) - n$ assignments.

Template Parameters

- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- **Size**: The type of the argument specifying the number of positions to shift by.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **n**: Refers to the number of positions to shift.

Note The type of dereferenced `FwdIter` must meet the requirements of `MoveAssignable`.

Return The `shift_left` algorithm returns `FwdIter`. The `shift_left` algorithm returns an iterator to the end of the resulting range.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Size>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter> shift_left (ExPolicy      &&policy,
                                                               FwdIter first, Sent last,
                                                               Size n)
```

Shifts the elements in the range [first, last) by n positions towards the beginning of the range. For every integer i in [0, last - first]

- n), moves the element originally at position first + n + i to position first + i.

The assignment operations in the parallel `shift_left` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- **Size**: The type of the argument specifying the number of positions to shift by.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **n**: Refers to the number of positions to shift.

The assignment operations in the parallel `shift_left` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced `FwdIter` must meet the requirements of `MoveAssignable`.

Return The `shift_left` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `shift_left` algorithm returns an iterator to the end of the resulting range.

```
template<typename Rng, typename Size>
hpx::traits::range_iterator_t<Rng> shift_left (Rng &&rng, Size n)
```

Shifts the elements in the range [first, last) by n positions towards the beginning of the range. For every integer i in [0, last - first]

- n), moves the element originally at position first + n + i to position first + i.

The assignment operations in the parallel *shift_left* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- **Rng**: The type of the range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Size**: The type of the argument specifying the number of positions to shift by.

Parameters

- **rng**: Refers to the range in which the elements will be shifted.
- **n**: Refers to the number of positions to shift.

Note The type of dereferenced *hpx::traits::range_iterator_t<Rng>* must meet the requirements of *MoveAssignable*.

Return The *shift_left* algorithm returns *hpx::traits::range_iterator_t<Rng>*. The *shift_left* algorithm returns an iterator to the end of the resulting range.

```
template<typename ExPolicy, typename Rng, typename Size>
parallel::util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator_t<Rng>>::type shift_left(ExPolicy
&&policy,
Rng
&&rng,
Size
n)
```

Shifts the elements in the range [first, last) by n positions towards the beginning of the range. For every integer i in [0, last - first]

- n), moves the element originally at position first + n + i to position first + i.

The assignment operations in the parallel *shift_left* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Size**: The type of the argument specifying the number of positions to shift by.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the range in which the elements will be shifted.
- **n**: Refers to the number of positions to shift.

The assignment operations in the parallel *shift_left* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced *hpx::traits::range_iterator_t<Rng>* must meet the requirements of *MoveAssignable*.

Return The *shift_left* algorithm returns a *hpx::future<hpx::traits::range_iterator_t<Rng>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *hpx::traits::range_iterator_t<Rng>* otherwise. The *shift_left* algorithm returns an iterator to the end of the resulting range.

hpx/parallel/container_algorithms/shift_right.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename FwdIter, typename Sent, typename Size>
FwdIter shift_right (FwdIter first, Sent last, Size n)
```

Shifts the elements in the range [first, last) by n positions towards the end of the range. For every integer i in [0, last - first - n), moves the element originally at position first + i to position first

- n + i.

The assignment operations in the parallel *shift_right* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **FwdIter**.
- **Size**: The type of the argument specifying the number of positions to shift by.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **n**: Refers to the number of positions to shift.

Note The type of dereferenced *FwdIter* must meet the requirements of *MoveAssignable*.

Return The *shift_right* algorithm returns *FwdIter*. The *shift_right* algorithm returns an iterator to the end of the resulting range.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Size>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter> shift_right (ExPolicy &&policy,
                                                               FwdIter first, Sent last,
                                                               Size n)
```

Shifts the elements in the range [first, last) by n positions towards the end of the range. For every integer i in [0, last - first - n), moves the element originally at position first + i to position first

- n + i.

The assignment operations in the parallel *shift_right* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

- **Sent:** The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- **Size:** The type of the argument specifying the number of positions to shift by.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `n`: Refers to the number of positions to shift.

The assignment operations in the parallel `shift_right` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced `FwdIter` must meet the requirements of `MoveAssignable`.

Return The `shift_right` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `shift_right` algorithm returns an iterator to the end of the resulting range.

```
template<typename Rng, typename Size>
hpx::traits::range_iterator_t<Rng> shift_right (Rng &&rng, Size n)
```

Shifts the elements in the range [first, last) by n positions towards the end of the range. For every integer i in [0, last - first - n), moves the element originally at position first + i to position first

- `n + i`.

The assignment operations in the parallel `shift_right` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- `Rng`: The type of the range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `Size`: The type of the argument specifying the number of positions to shift by.

Parameters

- `rng`: Refers to the range in which the elements will be shifted.
- `n`: Refers to the number of positions to shift.

Note The type of dereferenced `hpx::traits::range_iterator_t<Rng>` must meet the requirements of `MoveAssignable`.

Return The `shift_right` algorithm returns `hpx::traits::range_iterator_t<Rng>`. The `shift_right` algorithm returns an iterator to the end of the resulting range.

```
template<typename ExPolicy, typename Rng, typename Size>
parallel::util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator_t<Rng>>::type shift_right (ExPolicy &&policy,
&&pol-
icy,
Rng &&rng,
Size n)
```

Shifts the elements in the range [first, last) by n positions towards the end of the range. For every integer i in [0, last - first - n), moves the element originally at position first + i to position first

- `n + i`.

The assignment operations in the parallel `shift_right` algorithm invoked with an execution policy ob-

ject of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: At most (last - first) - n assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Size*: The type of the argument specifying the number of positions to shift by.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the range in which the elements will be shifted.
- *n*: Refers to the number of positions to shift.

The assignment operations in the parallel *shift_right* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced *hpx::traits::range_iterator_t<Rng>* must meet the requirements of *MoveAssignable*.

Return The *shift_right* algorithm returns a *hpx::future<hpx::traits::range_iterator_t<Rng>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *hpx::traits::range_iterator_t<Rng>* otherwise. The *shift_right* algorithm returns an iterator to the end of the resulting range.

hpx/parallel/container_algorithms/sort.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename RandomIt, typename Sent, typename Comp, typename Proj>
RandomIt sort (RandomIt first, Sent last, Comp &&comp, Proj &&proj)
```

Sorts the elements in the range [first, last) in ascending order. The order of equal elements is not guaranteed to be preserved. The function uses the given comparison function object *comp* (defaults to using operator<()).

A sequence is sorted with respect to a comparator *comp* and a projection *proj* if for every iterator *i* pointing to the sequence and every non-negative integer *n* such that *i + n* is a valid iterator pointing to an element of the sequence, and *INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false*.

Note Complexity: O(Nlog(N)), where N = *detail::distance(first, last)* comparisons.
comp has to induce a strict weak ordering on the values.

Template Parameters

- *RandomIt*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random iterator.

- **Sent:** The type of the source sentinel (deduced). This sentinel type must be a sentinel for RandomIt.
- **Comp:** The type of the function/function object to use (deduced).
- **Proj:** The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last:** Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **comp:** comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator.
- **proj:** Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.

The assignments in the parallel *sort* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Return The *sort* algorithm returns *RandomIt*. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```
template<typename ExPolicy, typename RandomIt, typename Sent, typename Comp, typename Proj>
parallel::util::detail::algorithm_result<ExPolicy, RandomIt>::type sort (ExPolicy &&policy, RandomIt first, Sent last, Comp &&comp, Proj &&proj)
```

Sorts the elements in the range [first, last) in ascending order. The order of equal elements is not guaranteed to be preserved. The function uses the given comparison function object comp (defaults to using operator<()).

A sequence is sorted with respect to a comparator *comp* and a projection *proj* if for every iterator *i* pointing to the sequence and every non-negative integer *n* such that *i* + *n* is a valid iterator pointing to an element of the sequence, and $\text{(INVOKE}(\text{comp}, \text{(INVOKE}(\text{proj}, *(\text{i} + \text{n})), \text{(INVOKE}(\text{proj}, *\text{i}))) == \text{false})$.

Note Complexity: $O(N\log(N))$, where $N = \text{detail}::\text{distance}(\text{first}, \text{last})$ comparisons.

comp has to induce a strict weak ordering on the values.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **RandomIt:** The type of the source iterators used (deduced). This iterator type must meet the requirements of an random iterator.
- **Sent:** The type of the source sentinel (deduced). This sentinel type must be a sentinel for RandomIt.
- **Comp:** The type of the function/function object to use (deduced).
- **Proj:** The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last:** Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **comp:** comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator.

- *proj*: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *sort* algorithm returns a *hpx::future<RandomIt>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *RandomIt* otherwise. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```
template<typename Rng, typename Comp, typename Proj>
hpx::traits::range_iterator<Rng>::type sort (Rng &&rng, Compare &&comp, Proj &&proj)
Sorts the elements in the range rng in ascending order. The order of equal elements is not guaranteed to be preserved. The function uses the given comparison function object comp (defaults to using operator<()).
```

A sequence is sorted with respect to a comparator *comp* and a projection *proj* if for every iterator *i* pointing to the sequence and every non-negative integer *n* such that *i* + *n* is a valid iterator pointing to an element of the sequence, and *INVOKE*(*comp*, *INVOKE*(*proj*, *(*i* + *n*)), *INVOKE*(*proj*, **i*)) == false.

Note Complexity: O(Nlog(N)), where N = std::distance(begin(*rng*), end(*rng*)) comparisons. *comp* has to induce a strict weak ordering on the values.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Comp*: The type of the function/function object to use (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *comp*: *comp* is a callable object. The return value of the *INVOKE* operation applied to an object of type *Comp*, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator.
- *proj*: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.

The assignments in the parallel *sort* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Return The *sort* algorithm returns *typename hpx::traits::range_iterator<Rng>::type*. It returns *last*.

```
template<typename ExPolicy, typename Rng, typename Pred, typename Proj>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type sort (ExPolicy
&&policy,
Rng
&&rng,
Comp
&&comp,
Proj &&)
```

Sorts the elements in the range *rng* in ascending order. The order of equal elements is not guaranteed

to be preserved. The function uses the given comparison function object `comp` (defaults to using `operator<()`).

A sequence is sorted with respect to a comparator `comp` and a projection `proj` if for every iterator `i` pointing to the sequence and every non-negative integer `n` such that `i + n` is a valid iterator pointing to an element of the sequence, and `INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false.`

Note Complexity: $O(N \log(N))$, where $N = \text{std}::\text{distance}(\text{begin}(\text{rng}), \text{end}(\text{rng}))$ comparisons. `comp` has to induce a strict weak ordering on the values.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Comp`: The type of the function/function object to use (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `comp`: `comp` is a callable object. The return value of the `INVOKE` operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `sort` algorithm returns a `hpx::future<typename hpx::traits::range_iterator<Rng> ::type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `typename hpx::traits::range_iterator<Rng> ::type` otherwise. It returns `last`.

hpx/parallel/algorithms/sort_by_key.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

Typedefs

```
template<typename KeyIter, typename ValueIter>
using sort_by_key_result = std::pair<KeyIter, ValueIter>
```

Functions

```
template<typename ExPolicy, typename KeyIter, typename ValueIter, typename Compare = detail::less>
util::detail::algorithm_result_t<ExPolicy, sort_by_key_result<KeyIter, ValueIter>> sort_by_key(ExPolicy
    &&policy,
    KeyIter
    key_first,
    KeyIter
    key_last,
    ValueIter
    value_first,
    Compare
    &&comp
    =
    Compare()
)
```

Sorts one range of data using keys supplied in another range. The key elements in the range [key_first, key_last) are sorted in ascending order with the corresponding elements in the value range moved to follow the sorted order. The algorithm is not stable, the order of equal elements is not guaranteed to be preserved. The function uses the given comparison function object comp (defaults to using operator<()).

A sequence is sorted with respect to a comparator *comp* and a projection *proj* if for every iterator *i* pointing to the sequence and every non-negative integer *n* such that *i + n* is a valid iterator pointing to an element of the sequence, and INVOKE(*comp*, INVOKE(*proj*, *(*i + n*)), INVOKE(*proj*, **i*)) == false.

Note Complexity: O(Nlog(N)), where N = std::distance(first, last) comparisons.

comp has to induce a strict weak ordering on the values.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **KeyIter:** The type of the key iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- **ValueIter:** The type of the value iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- **Comp:** The type of the function/function object to use (deduced).

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **key_first:** Refers to the beginning of the sequence of key elements the algorithm will be applied to.
- **key_last:** Refers to the end of the sequence of key elements the algorithm will be applied to.

- `value_first`: Refers to the beginning of the sequence of value elements the algorithm will be applied to, the range of elements must match `[key_first, key_last)`
- `comp`: `comp` is a callable object. The return value of the `INVOKE` operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `sort_by-key` algorithm returns a `hpx::future<sort_by_key_result<KeyIter, ValueIter>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns *otherwise*. The algorithm returns a pair holding an iterator pointing to the first element after the last element in the input key sequence and an iterator pointing to the first element after the last element in the input value sequence.

hpx/parallel/container_algorithms/stable_sort.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename RandomIt, typename Sent, typename Comp, typename Proj>
RandomIt stable_sort(RandomIt first, Sent last, Comp &&comp, Proj &&proj)
```

Sorts the elements in the range `[first, last)` in ascending order. The relative order of equal elements is preserved. The function uses the given comparison function object `comp` (defaults to using operator`<()`).

A sequence is sorted with respect to a comparator `comp` and a projection `proj` if for every iterator `i` pointing to the sequence and every non-negative integer `n` such that `i + n` is a valid iterator pointing to an element of the sequence, and `INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false`.

Note Complexity: $O(N \log(N))$, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$ comparisons.

`comp` has to induce a strict weak ordering on the values.

Template Parameters

- `RandomIt`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `RandomIt`.
- `Comp`: The type of the function/function object to use (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.

- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The assignments in the parallel `stable_sort` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Return The `stable_sort` algorithm returns `RandomIt`. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```
template<typename ExPolicy, typename RandomIt, typename Sent, typename Comp, typename Proj>
parallel::util::detail::algorithm_result<ExPolicy, RandomIt>::type stable_sort(ExPolicy
    &&policy,
    RandomIt first,
    Sent last, Comp
    &&comp, Proj
    &&proj)
```

Sorts the elements in the range `[first, last)` in ascending order. The relative order of equal elements is preserved. The function uses the given comparison function object `comp` (defaults to using operator`<()`).

A sequence is sorted with respect to a comparator `comp` and a projection `proj` if for every iterator `i` pointing to the sequence and every non-negative integer `n` such that `i + n` is a valid iterator pointing to an element of the sequence, and `INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false`.

Note Complexity: $O(N \log(N))$, where $N = \text{std}::\text{distance}(first, last)$ comparisons.
`comp` has to induce a strict weak ordering on the values.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `RandomIt`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `RandomIt`.
- `Comp`: The type of the function/function object to use (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `comp`: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of

type *sequenced_policy* execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *stable_sort* algorithm returns a *hpx::future<RandomIt>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *RandomIt* otherwise. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```
template<typename Rng, typename Comp, typename Proj>
hpx::traits::range_iterator<Rng>::type stable_sort (Rng &&rng, Compare &&comp, Proj &&proj)
```

Sorts the elements in the range [first, last) in ascending order. The relative order of equal elements is preserved. The function uses the given comparison function object *comp* (defaults to using operator<()).

A sequence is sorted with respect to a comparator *comp* and a projection *proj* if for every iterator *i* pointing to the sequence and every non-negative integer *n* such that *i + n* is a valid iterator pointing to an element of the sequence, and *INVOKED(comp, INVOKED(proj, *(i + n)), INVOKED(proj, *i)) == false*.

Note Complexity: $O(N \log(N))$, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$ comparisons. *comp* has to induce a strict weak ordering on the values.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Comp*: The type of the function/function object to use (deduced).
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *comp*: *comp* is a callable object. The return value of the *INVOKED* operation applied to an object of type *Comp*, when contextually converted to *bool*, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that *comp* will not apply any non-constant function through the dereferenced iterator.
- *proj*: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate *comp* is invoked.

The assignments in the parallel *stable_sort* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Return The *stable_sort* algorithm returns *typename hpx::traits::range_iterator<Rng>::type*. It returns *last*.

```
template<typename ExPolicy, typename Rng, typename Pred, typename Proj>
```

```
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type stable_sort (ExPolicy &&policy, Rng &&rng, Comp &&comp, Proj &&proj)
```

Sorts the elements in the range [first, last) in ascending order. The relative order of equal elements

is preserved. The function uses the given comparison function object `comp` (defaults to using operator`<()`).

A sequence is sorted with respect to a comparator `comp` and a projection `proj` if for every iterator `i` pointing to the sequence and every non-negative integer `n` such that `i + n` is a valid iterator pointing to an element of the sequence, and `INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false.`

Note Complexity: $O(N \log(N))$, where $N = \text{std}::\text{distance}(\text{first}, \text{last})$ comparisons. `comp` has to induce a strict weak ordering on the values.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Comp`: The type of the function/function object to use (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `comp`: `comp` is a callable object. The return value of the `INVOKE` operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `stable_sort` algorithm returns a `hpx::future<typename hpx::traits::range_iterator<Rng>::type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `typename hpx::traits::range_iterator<Rng>::type` otherwise. It returns `last`.

hpx/parallel/container_algorithms/stable_sort.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2, typename Pred, typename Proj1, typename Proj2>>
```

`bool starts_with(FwdIter1 first1, Sent1 last1, FwdIter2 first2, Sent2 last2, Pred &&pred, Proj1
 &&proj1, Proj2 &&proj2)`

Checks whether the second range defined by [first1, last1) matches the prefix of the first range defined by [first2, last2)

The assignments in the parallel *starts_with* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear: at most min(N1, N2) applications of the predicate and both projections.

Template Parameters

- **Iter1**: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- **Sent1**: The type of the end source iterators used(deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- **Iter2**: The type of the begin destination iterators used deduced). This iterator type must meet the requirements of a input iterator.
- **Sent2**: The type of the end destination iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter2.
- **Pred**: The binary predicate that compares the projected elements.
- **Proj1**: The type of an optional projection function for the source range. This defaults to *util::projection_identity*
- **Proj1**: The type of an optional projection function for the destination range. This defaults to *util::projection_identity*

Parameters

- **first1**: Refers to the beginning of the source range.
- **last1**: Sentinel value referring to the end of the source range.
- **first2**: Refers to the beginning of the destination range.
- **last2**: Sentinel value referring to the end of the destination range.
- **pred**: Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the in two ranges projected by proj1 and proj2 respectively.
- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

Return The *starts_with* algorithm returns *bool*. The *starts_with* algorithm returns a boolean with the value true if the second range matches the prefix of the first range, false otherwise.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2, typename hp::parallel::util::detail::algorithm_result<ExPolicy, bool>::type starts_with(ExPolicy  
    &&policy,  
    FwdIter1 first1,  
    Sent1 last1,  
    FwdIter2 first2,  
    Sent2 last2,  
    Pred &&pred,  
    Proj1 &&proj1,  
    Proj2 &&proj2)
```

Checks whether the second range defined by [first1, last1) matches the prefix of the first range defined by [first2, last2)

The assignments in the parallel *starts_with* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear: at most min(N1, N2) applications of the predicate and both projections.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the begin source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent1*: The type of the end source iterators used(deduced). This iterator type must meet the requirements of an sentinel for Iter1.
- *FwdIter2*: The type of the begin destination iterators used deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent2*: The type of the end destination iterators used (deduced). This iterator type must meet the requirements of an sentinel for Iter2.
- *Pred*: The binary predicate that compares the projected elements.
- *Proj1*: The type of an optional projection function for the source range. This defaults to *util::projection_identity*
- *Proj1*: The type of an optional projection function for the destination range. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first1*: Refers to the beginning of the source range.
- *last1*: Sentinel value referring to the end of the source range.
- *first2*: Refers to the beginning of the destination range.
- *last2*: Sentinel value referring to the end of the destination range.
- *pred*: Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the in two ranges projected by proj1 and proj2 respectively.
- *proj1*: Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- *proj2*: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *starts_with* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *starts_with* algorithm returns a *hpx::future<bool>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *bool* otherwise. The *starts_with* algorithm returns a boolean with the value true if the second range matches the prefix of the first range, false otherwise.

```
template<typename Rng1, typename Rng2, typename Pred, typename Proj1, typename Proj2>
bool starts_with(Rng1 &&rng1, Rng2 &&rng2, Pred &&pred, Proj1 &&proj1, Proj2 &&proj2)
```

Checks whether the second range *rng2* matches the prefix of the first range *rng1*.

The assignments in the parallel *starts_with* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear: at most min(N1, N2) applications of the predicate and both projections.

Template Parameters

- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Rng2*: The type of the destination range used (deduced). The iterators extracted from this

range type must meet the requirements of an forward iterator.

- **Pred:** The binary predicate that compares the projected elements.
- **Proj1:** The type of an optional projection function for the source range. This defaults to `util::projection_identity`
- **Proj1:** The type of an optional projection function for the destination range. This defaults to `util::projection_identity`

Parameters

- **rng1:** Refers to the source range.
- **rng2:** Refers to the destination range.
- **pred:** Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the in two ranges projected by proj1 and proj2 respectively.
- **proj1:** Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.
- **proj2:** Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.

Return The `starts_with` algorithm returns `bool`. The `starts_with` algorithm returns a boolean with the value true if the second range matches the prefix of the first range, false otherwise.

```
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred, typename Proj1, typename Proj2,
         hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type starts_with(ExPolicy
                                         &&policy,
                                         Rng1 &&rng1,
                                         Rng2 &&rng2,
                                         Pred &&pred,
                                         Proj1 &&proj1,
                                         Proj2 &&proj2)
```

Checks whether the second range `rng2` matches the prefix of the first range `rng1`.

The assignments in the parallel `starts_with` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Linear: at most $\min(N_1, N_2)$ applications of the predicate and both projections.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng1:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Rng2:** The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Pred:** The binary predicate that compares the projected elements.
- **Proj1:** The type of an optional projection function for the source range. This defaults to `util::projection_identity`
- **Proj1:** The type of an optional projection function for the destination range. This defaults to `util::projection_identity`

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **rng1:** Refers to the source range.
- **rng2:** Refers to the destination range.
- **pred:** Specifies the binary predicate function (or function object) which will be invoked for comparison of the elements in the in two ranges projected by proj1 and proj2 respectively.
- **proj1:** Specifies the function (or function object) which will be invoked for each of the elements in the source range as a projection operation before the actual predicate *is* invoked.

- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements in the destination range as a projection operation before the actual predicate *is* invoked.
- The assignments in the parallel *starts_with* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *starts_with* algorithm returns a `hpx::future<bool>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `bool` otherwise. The *starts_with* algorithm returns a boolean with the value true if the second range matches the prefix of the first range, false otherwise.

hpx/parallel/container_algorithms/swap_ranges.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

```
template<typename InIter1, typename Sent1, typename InIter2, typename Sent2>
swap_ranges_result<InIter1, InIter2> swap_ranges(InIter1 first1, Sent1 last1, InIter2 first2,
                                                Sent2 last2)
```

Exchanges elements between range [first1, last1) and another range starting at first2.

The swap operations in the parallel *swap_ranges* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first1* and *last1*

Template Parameters

- `InIter1`: The type of the first range of iterators to swap (deduced).
- `Sent1`: The type of the first sentinel (deduced). This sentinel type must be a sentinel for `InIter1`.
- `InIter2`: The type of the second range of iterators to swap (deduced).
- `Sent2`: The type of the second sentinel (deduced). This sentinel type must be a sentinel for `InIter2`.

Parameters

- `first1`: Refers to the beginning of the sequence of elements for the first range.
- `last1`: Refers to sentinel value denoting the end of the sequence of elements for the first range.
- `first2`: Refers to the beginning of the sequence of elements for the second range.
- `last2`: Refers to sentinel value denoting the end of the sequence of elements for the second range.

Return The *swap_ranges* algorithm returns `swap_ranges_result<InIter1, InIter2>`. The *swap_ranges* algorithm returns `in_in_result` with the first element as the iterator to the element past the last element exchanged in range beginning with *first1* and the second element as the iterator to the element past the last element exchanged in the range beginning with *first2*.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2>
```

```
parallel::util::detail::algorithm_result<ExPolicy, swap_ranges_result<FwdIter1, FwdIter2>>::type swap_ranges (ExPolicy
&&policy,
FwdIter1,
first1,
Sent1,
last1,
FwdIter2,
first2,
Sent2,
last2)
```

Exchanges elements between range [first1, last1) and another range starting at first2.

The swap operations in the parallel *swap_ranges* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first1* and *last1*

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the first range of iterators to swap (deduced).
- *Sent1*: The type of the first sentinel (deduced). This sentinel type must be a sentinel for *FwdIter1*.
- *FwdIter2*: The type of the second range of iterators to swap (deduced).
- *Sent2*: The type of the second sentinel (deduced). This sentinel type must be a sentinel for *FwdIter2*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first1*: Refers to the beginning of the sequence of elements for the first range.
- *last1*: Refers to sentinel value denoting the end of the sequence of elements for the first range.
- *first2*: Refers to the beginning of the sequence of elements for the second range.
- *last2*: Refers to sentinel value denoting the end of the sequence of elements for the second range.

The swap operations in the parallel *swap_ranges* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *swap_ranges* algorithm returns a *hpx::future<swap_ranges_result<FwdIter1, FwdIter2>>* if the execution policy is of type *parallel_task_policy* and returns *FwdIter2* otherwise. The *swap_ranges* algorithm returns *in_in_result* with the first element as the iterator to the element past the last element exchanged in range beginning with *first1* and the second element as the iterator to the element past the last element exchanged in the range beginning with *first2*.

```
template<typename Rng1, typename Rng2>
swap_ranges_result<hpx::traits::range_iterator_t<Rng1>, hpx::traits::range_iterator_t<Rng2>> swap_ranges (Rng1
&&rng1,
Rng2
&&rng2)
```

Exchanges elements between range [first1, last1) and another range starting at first2.

The swap operations in the parallel *swap_ranges* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first1* and *last1*

Template Parameters

- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- *rng1*: Refers to the sequence of elements of the first range.
- *rng2*: Refers to the sequence of elements of the second range.

Return The *swap_ranges* algorithm returns *swap_ranges_result*<
hpx::traits::range_iterator_t<*Rng1*>, hpx::traits::range_iterator_t<*Rng1*>>. The *swap_ranges* algorithm returns *in_in_result* with the first element as the iterator to the element past the last element exchanged in range beginning with *first1* and the second element as the iterator to the element past the last element exchanged in the range beginning with *first2*.

```
template<typename ExPolicy, typename Rng1, typename Rng2>
```

```
parallel::util::detail::algorithm_result<ExPolicy, swap_ranges_result<hpx::traits::range_iterator_t<Rng1>, hpx::traits::range_
```

Exchanges elements between range [*first1*, *last1*] and another range starting at *first2*.

The swap operations in the parallel *swap_ranges* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first1* and *last1*

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the sequence of elements of the first range.
- *rng2*: Refers to the sequence of elements of the second range.

The swap operations in the parallel *swap_ranges* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *swap_ranges* algorithm returns a *hpx::future<swap_ranges_result<*

`hpx::traits::range_iterator_t<Rng1>, hpx::traits::range_iterator_t<Rng1>>>` if the execution policy is of type `parallel_task_policy` and returns `swap_ranges_result<hpx::traits::range_iterator_t<Rng1>, hpx::traits::range_iterator_t<Rng1>>`. otherwise. The `swap_ranges` algorithm returns `in_in_result` with the first element as the iterator to the element past the last element exchanged in range beginning with `first1` and the second element as the iterator to the element past the last element exchanged in the range beginning with `first2`.

hpx/parallel/container_algorithms/transform.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

`template<typename ExPolicy, typename Rng, typename OutIter, typename F, typename Proj = util::projection_identity<util::detail::algorithm_result<ExPolicy, ranges::unary_transform_result<typename hpx::traits::range_iterator<Rng>>>>`

Applies the given function *f* to the given range *rng* and stores the result in another range, beginning at *dest*.

The invocations of *f* in the parallel *transform* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly `size(rng)` applications of *f*

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of *f*.
- **Rng:** The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **OutIter:** The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **F:** The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj:** The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.

- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type &a);
```

The signature does not need to have `const&`. The type `Type` must be such that an object of type `range_iterator<Rng>::type` can be dereferenced and then implicitly converted to `Type`. The type `Ret` must be such that an object of type `OutIter` can be dereferenced and assigned a value of type `Ret`.

- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate `f` is invoked.

The invocations of `f` in the parallel `transform` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `transform` algorithm returns a `hpx::future<ranges::unary_transform_result<range_iterator<Rng>::type, OutIter> >` if the execution policy is of type `parallel_task_policy` and returns `ranges::unary_transform_result<range_iterator<Rng>::type, OutIter>` otherwise. The `transform` algorithm returns a tuple holding an iterator referring to the first element after the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename F, typename OutIter> OutIter > if the execution policy is of type parallel_task_policy and returns ranges::unary_transform_result<range_iterator<Rng>::type, OutIter> otherwise. The transform algorithm returns a tuple holding an iterator referring to the first element after the input sequence and the output iterator to the element in the destination range, one past the last element copied.
```

Applies the given function `f` to the given range `rng` and stores the result in another range, beginning at `dest`.

The invocations of `f` in the parallel `transform` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Exactly `size(rng)` applications of `f`

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of `f`.
- `FwdIter1`: The type of the source iterators for the first range used (deduced). This iterator type must meet the requirements of an forward iterator.

- **Sent1**: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for *FwdIter1*.
- **FwdIter2**: The type of the source iterators for the first range used (deduced). This iterator type must meet the requirements of an forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires *F* to meet the requirements of *CopyConstructible*.
- **Proj**: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the first sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is an unary predicate. The signature of this predicate should be equivalent to:

```
Ret fun (const Type &a);
```

The signature does not need to have `const&`. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *Ret* must be such that an object of type *FwdIter2* can be dereferenced and assigned a value of type *Ret*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *f* is invoked.

The invocations of *f* in the parallel *transform* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *transform* algorithm returns a *hpx::future<ranges::unary_transform_result<FwdIter1, FwdIter2>* if the execution policy is of type *parallel_task_policy* and returns *ranges::unary_transform_result<FwdIter1, FwdIter2>* otherwise. The *transform* algorithm returns a tuple holding an iterator referring to the first element after the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2, typename Proj = util::projection_identity> Ret hpx::parallel::transform(ExPolicy&& policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, FwdIter2 sent1, FwdIter2 sent2, Proj proj = Proj{})
```

`parallel::util::detail::algorithm_result<ExPolicy, ranges::binary_transform_result<FwdIter1, FwdIter2, FwdIter3>>::type t`

Applies the given function f to pairs of elements from two ranges: one defined by rng and the other beginning at $first2$, and stores the result in another range, beginning at $dest$.

The invocations of f in the parallel *transform* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Exactly $\text{size}(rng)$ applications of f

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of f .
- `FwdIter1`: The type of the source iterators for the first range used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent1`: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for `FwdIter1`.
- `FwdIter2`: The type of the source iterators for the first range used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent2`: The type of the end source iterators used (deduced). This iterator type must meet the requirements of an sentinel for `FwdIter2`.
- `FwdIter3`: The type of the source iterators for the first range used (deduced). This iterator type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires F to meet the requirements of *CopyConstructible*.
- `Proj1`: The type of an optional projection function to be used for elements of the first sequence. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function to be used for elements of the second sequence. This defaults to `util::projection_identity`

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the first sequence of elements the algorithm will be applied to.

- `last1`: Refers to the end of the first sequence of elements the algorithm will be applied to.
- `first2`: Refers to the beginning of the second sequence of elements the algorithm will be applied to.
- `last2`: Refers to the end of the second sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively. The type `Ret` must be such that an object of type `FwdIter3` can be dereferenced and assigned a value of type `Ret`.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `f` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `f` is invoked.

The invocations of `f` in the parallel `transform` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `transform` algorithm returns a `hpx::future<ranges::binary_transform_result<FwdIter1, FwdIter2, FwdIter3>>` if the execution policy is of type `parallel_task_policy` and returns `ranges::binary_transform_result<FwdIter1, FwdIter2, FwdIter3>` otherwise. The `transform` algorithm returns a tuple holding an iterator referring to the first element after the first input sequence, an iterator referring to the first element after the second input sequence, and the output iterator referring to the element in the destination range, one past the last element copied.

hpx/parallel/container_algorithms/transform_exclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename InIter, typename Sent, typename OutIter, typename T, typename BinOp, typename UnOp>
transform_exclusive_scan_result<InIter, OutIter> transform_exclusive_scan(InIter first,
                                                               Sent last,
                                                               OutIter dest, T
                                                               init, BinOp
                                                               &&binary_op,
                                                               UnOp
                                                               &&unary_op)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), ..., conv(* $(first + (i - result) - 1)$)).

The reduce operations in the parallel *transform_exclusive_scan* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(\text{last} - \text{first})$ applications of the predicates *op* and *conv*.

Template Parameters

- *InIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *OutIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Conv*: The type of the unary function object used for the conversion operation.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).
- *Op*: The type of the binary function object used for the reduction operation.

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

- *init*: The initial value for the generalized sum.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

Neither *conv* nor *op* shall invalidate iterators or subranges, or modify elements in the ranges [first, last) or [result, result + (last - first)).

Return The *transform_exclusive_scan* algorithm returns *transform_exclusive_scan_result<InIter, OutIter>*. The *transform_exclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note *GENERALIZED_NONCOMMUTATIVE_SUM*(*op*, *a*₁, ..., *a*_N) is defined as:

- *a*₁ when *N* is 1
- *op*(*GENERALIZED_NONCOMMUTATIVE_SUM*(*op*, *a*₁, ..., *a*_K), *GENERALIZED_NONCOMMUTATIVE_SUM*(*op*, *a*_M, ..., *a*_N)) where $1 < K+1 = M \leq N$.

The behavior of *transform_exclusive_scan* may be non-deterministic for a non-associative predicate.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename T, typename B
```

`parallel::util::detail::algorithm_result<ExPolicy, transform_exclusive_result<FwdIter1, FwdIter2>>::type transform_ex`

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), \dots , conv(* $(first + (i - result) - 1)$)).

The reduce operations in the parallel *transform_exclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates *op* and *conv*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Conv*: The type of the unary function object used for the conversion operation.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).
- *Op*: The type of the binary function object used for the reduction operation.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun (const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

- *init*: The initial value for the generalized sum.
- *op*: Specifies the function (or function object) which will be invoked for each of the values

of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel `transform_exclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges [first, last) or [result, result + (last - first)).

Return The `transform_exclusive_scan` algorithm returns a `hpx::future<transform_exclusive_result<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `transform_exclusive_result<FwdIter1, FwdIter2>` otherwise. The `transform_exclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(`op`, `a1, ..., aN`) is defined as:

- `a1` when `N` is 1
- `op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN))` where $1 < K+1 = M \leq N$.

The behavior of `transform_exclusive_scan` may be non-deterministic for a non-associative predicate.

```
template<typename Rng, typename O, typename T, typename BinOp, typename UnOp>
transform_exclusive_scan_result<traits::range_iterator_t<Rng>, O> transform_exclusive_scan(Rng
    &&rng,
    O
    dest,
    T
    init,
    BinOp
    &&bi-
    nary_op,
    UnOp
    &&unary_op)
```

Assigns through each iterator i in [result, result + (last - first)) the value of `GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), ..., conv(*first + (i - result) - 1))`.

The reduce operations in the parallel `transform_exclusive_scan` algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates `op` and `conv`.

Template Parameters

- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `O`: The type of the iterator representing the destination range (deduced).
- `Conv`: The type of the unary function object used for the conversion operation.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `rng`: Refers to the sequence of elements the algorithm will be applied to.

- `conv`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

- `init`: The initial value for the generalized sum.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges [first, last) or [result, result + (last - first)).

Return The `transform_exclusive_scan` algorithm returns a returns `transform_exclusive_scan_result<traits::range_iterator_t<Rng>, O>`. The `transform_exclusive_scan` algorithm returns an input iterator to one past the end of the range and an output iterator to the element in the destination range, one past the last element copied.

Note `GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN)` is defined as:

- a_1 when N is 1
- $op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_K), GENERALIZED_NONCOMMUTATIVE_SUM(op, a_{M+1}, \dots, a_N))$ where $1 < K+1 = M \leq N$.

The behavior of `transform_exclusive_scan` may be non-deterministic for a non-associative predicate.

```
template<typename ExPolicy, typename Rng, typename O, typename T, typename BinOp, typename UnOpExPolicy, transform_exclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type
```

Assigns through each iterator i in [result, result + (last - first)) the value of `GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), \dots, conv(*first + (i - result) - 1))`.

The reduce operations in the parallel `transform_exclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates `op` and `conv`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `O`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator. This iterator type must meet the requirements of an forward iterator.
- `Conv`: The type of the unary function object used for the conversion operation.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).
- `Op`: The type of the binary function object used for the reduction operation.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `conv`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

- `init`: The initial value for the generalized sum.
- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel `transform_exclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The `transform_exclusive_scan` algorithm returns a `hpx::future<transform_exclusive_scan_result<traits::range_iterator_t<Rng>, O>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `transform_exclusive_scan_result<traits::range_iterator_t<Rng>, O>` otherwise. The `transform_exclusive_scan` algorithm returns an input iterator to one past the end of the range and an output iterator to the element in the destination range, one past the last element copied.

Note `GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN)` is defined as:

- a_1 when N is 1
- $op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a_1, \dots, a_K), GENERALIZED_NONCOMMUTATIVE_SUM(op, a_M, \dots, a_N))$ where $1 < K+1 = M \leq N$.

The behavior of `transform_exclusive_scan` may be non-deterministic for a non-associative predicate.

hpx/parallel/container_algorithms/transform_inclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename InIter, typename Sent, typename OutIter, typename BinOp, typename UnOpInIter, OutIter> transform_inclusive_scan(InIter first,
                                                               Sent last,
                                                               OutIter dest,
                                                               BinOp &&bi-
                                                               nary_op,
                                                               UnOp &&unary_op)
```

Assigns through each iterator *i* in [*result*, *result* + (*last* - *first*)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(*op*, conv(**first*), ..., conv(*(*first* + (*i* - *result*)))).

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates *op* and *conv*.

Template Parameters

- **InIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **InIter**.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **Op**: The type of the binary function object used for the reduction operation.
- **Conv**: The type of the unary function object used for the conversion operation.

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **dest**: Refers to the beginning of the destination range.
- **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have **const&**, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- **conv**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [*first*, *last*). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun (const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges `[first,last)` or `[result,result + (last - first))`.

Return The `transform_inclusive_scan` algorithm returns `transform_inclusive_scan_result<InIter, OutIter>`. The `transform_inclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note `GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN)` is defined as:

- $a1$ when N is 1
- $op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, \dots, aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, \dots, aN))$ where $1 < K+1 = M \leq N$.

The behavior of `transform_inclusive_scan` may be non-deterministic for a non-associative predicate.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename BinOp, typename parallel::util::detail::algorithm_result<ExPolicy, transform_inclusive_result<FwdIter1, FwdIter2>>::type transform_inclusive_scan(FwdIter1 first, FwdIter1 result, FwdIter1 last, ExPolicy exPolicy, Sent sent, FwdIter2 dest, BinOp op, UnaryFunction conv)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of `GENERALIZED_NONCOMMUTATIVE_SUM(op, conv(*first), \dots, conv(*((first + (i - result))))`.

The reduce operations in the parallel `transform_inclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates `op` and `conv`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Op`: The type of the binary function object used for the reduction operation.
- `Conv`: The type of the unary function object used for the conversion operation.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **dest**: Refers to the beginning of the destination range.
- **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- **conv**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

The reduce operations in the parallel `transform_inclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges `[first, last)` or `[result, result + (last - first))`.

Return The `transform_inclusive_scan` algorithm returns a `hpx::future<transform_inclusive_result<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `transform_inclusive_result<FwdIter1, FwdIter2>` otherwise. The `transform_inclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(`op, a1, ..., aN`) is defined as:

- `a1` when `N` is 1
- `op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN))` where $1 < K+1 = M \leq N$.

The behavior of `transform_inclusive_scan` may be non-deterministic for a non-associative predicate.

```
template<typename Rng, typename O, typename BinOp, typename UnOp>
transform_inclusive_result<traits::range_iterator_t<Rng>, O> transform_inclusive_scan(Rng
&&rng,
O
dest,
BinOp
&&bi-
nary_op,
UnOp
&&unary_op)
```

Assigns through each iterator i in `[result, result + (last - first))` the value of `GENERALIZED_NONCOMMUTATIVE_SUM(op, conv(*first), ..., conv(*first + (i - result))))`.

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates *op* and *conv*.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *O*: The type of the iterator representing the destination range (deduced).
- *Op*: The type of the binary function object used for the reduction operation.
- *Conv*: The type of the unary function object used for the conversion operation.

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

Neither *conv* nor *op* shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The *transform_inclusive_scan* algorithm returns a returns *transform_inclusive_scan_result*<traits::range_iterator_t<*Rng*>, *O*>. The *transform_inclusive_scan* algorithm returns an input iterator to one past the end of the range and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(*op*, *a*₁, ..., *a*_N) is defined as:

- *a*₁ when *N* is 1
- *op*(GENERALIZED_NONCOMMUTATIVE_SUM(*op*, *a*₁, ..., *a*_K), GENERALIZED_NONCOMMUTATIVE_SUM(*op*, *a*_M, ..., *a*_N) where $1 < K+1 = M \leq N$.

The behavior of *transform_inclusive_scan* may be non-deterministic for a non-associative predicate.

```
template<typename ExPolicy, typename Rng, typename O, typename BinOp, typename UnOp
```

`parallel::util::detail::algorithm_result<ExPolicy, transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type`

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM($op, conv(*first), \dots, conv(*(first + (i - result)))$)).

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates *op* and *conv*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *O*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator. This iterator type must meet the requirements of an forward iterator.
- *Conv*: The type of the unary function object used for the conversion operation.
- *Op*: The type of the binary function object used for the reduction operation.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by $[first, last)$. This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered

fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither *conv* nor *op* shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The *transform_inclusive_scan* algorithm returns a *hpx::future<transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O>* otherwise. The *transform_inclusive_scan* algorithm returns an input iterator to one past the end of the range and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(*op*, *a*₁, ..., *a*_N) is defined as:

- *a*₁ when N is 1
- *op*(GENERALIZED_NONCOMMUTATIVE_SUM(*op*, *a*₁, ..., *a*_K), GENERALIZED_NONCOMMUTATIVE_SUM(*op*, *a*_M, ..., *a*_N) where 1 < K+1 = M <= N).

The behavior of *transform_inclusive_scan* may be non-deterministic for a non-associative predicate.

```
template<typename InIter, typename Sent, typename OutIter, typename BinOp, typename UnOp, typename T>
transform_inclusive_scan_result<InIter, OutIter> transform_inclusive_scan(InIter first,
                                                               Sent last,
                                                               OutIter dest, BinOp
                                                               &&binary_op,
                                                               UnOp
                                                               &&unary_op,
                                                               T init)
```

Assigns through each iterator *i* in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), ..., conv(*(first + (i - result))))).

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: O(*last* - *first*) applications of the predicates *op* and *conv*.

Template Parameters

- *InIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *OutIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- *Conv*: The type of the unary function object used for the conversion operation.
- *Op*: The type of the binary function object used for the reduction operation.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun (const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

- `op`: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- `init`: The initial value for the generalized sum.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The `transform_inclusive_scan` algorithm returns `transform_inclusive_scan_result<InIter, OutIter>`. The `transform_inclusive_scan` algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note `GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN)` is defined as:

- `a1` when `N` is 1
- `op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN))` where $1 < K+1 = M \leq N$.

The behavior of `transform_inclusive_scan` may be non-deterministic for a non-associative predicate.

```
template<typename ExPolicy, typename FwdIter1, typename Sent, typename FwdIter2, typename BinOp, typename parallel::util::detail::algorithm_result<ExPolicy, transform_inclusive_result<FwdIter1, FwdIter2>>::type transform_inclusive_scan
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of `GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), ..., conv(*first + (i - result))))`.

The reduce operations in the parallel `transform_inclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates `op` and `conv`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes

the assignments.

- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- *Conv*: The type of the unary function object used for the conversion operation.
- *Op*: The type of the binary function object used for the reduction operation.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *dest*: Refers to the beginning of the destination range.
- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

- *op*: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types *Type1* and *Ret* must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- *init*: The initial value for the generalized sum.

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither *conv* nor *op* shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The *transform_inclusive_scan* algorithm returns a *hpx::future<transform_inclusive_result<FwdIter1, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *transform_inclusive_result<FwdIter1, FwdIter2>* otherwise. The *transform_inclusive_scan* algorithm returns an input iterator to the point denoted by the sentinel and an output iterator to the element in the destination range, one past the last element copied.

Note *GENERALIZED_NONCOMMUTATIVE_SUM*(*op*, *a*₁, ..., *a*_N) is defined as:

- *a*₁ when *N* is 1
- *op*(*GENERALIZED_NONCOMMUTATIVE_SUM*(*op*, *a*₁, ..., *a*_K), *GENERALIZED_NONCOMMUTATIVE_SUM*(*op*, *a*_M, ..., *a*_N)) where $1 < K+1 = M \leq N$.

The behavior of *transform_inclusive_scan* may be non-deterministic for a non-associative predicate.

```
template<typename Rng, typename O, typename BinOp, typename UnOp, typename T>
```

```
transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O> transform_inclusive_scan(Rng
&&rng,
O
dest,
BinOp
&&bi-
nary_op,
UnOp
&&unary_op,
T
init)
```

Assigns through each iterator i in $[result, result + (last - first))$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), \dots , conv(*(first + (i - result)))).

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates *op* and *conv*.

Template Parameters

- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- O: The type of the iterator representing the destination range (deduced).
- Conv: The type of the unary function object used for the conversion operation.
- Op: The type of the binary function object used for the reduction operation.
- T: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- rng: Refers to the sequence of elements the algorithm will be applied to.
- conv: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to Type. The type R must be such that an object of this type can be implicitly converted to T.

- op: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The types Type1 and Ret must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- init: The initial value for the generalized sum.

Neither *conv* nor *op* shall invalidate iterators or subranges, or modify elements in the ranges [first, last) or [result, result + (last - first)).

Return The *transform_inclusive_scan* algorithm returns a returns *transform_inclusive_scan_result*< traits::range_iterator_t<Rng>, O>. The *transform_inclusive_scan* algorithm returns an input iterator to one past the end of the range and an output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN) is defined as:

- a1 when N is 1
- op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN)) where $1 < K+1 = M \leq N$.

The behavior of transform_inclusive_scan may be non-deterministic for a non-associative predicate.

```
template<typename ExPolicy, typename Rng, typename O, typename BinOp, typename UnOp, typename T>
parallel::util::detail::algorithm_result<ExPolicy, transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O>>::type
```

Assigns through each iterator i in $[result, result + (last - first)]$ the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), ..., conv(*(first + (i - result))))).

The reduce operations in the parallel *transform_inclusive_scan* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates *op* and *conv*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *O*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator. This iterator type must meet the requirements of an forward iterator.
- *Conv*: The type of the unary function object used for the conversion operation.
- *Op*: The type of the binary function object used for the reduction operation.
- *T*: The type of the value to be used as initial (and intermediate) values (deduced).

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *dest*: Refers to the beginning of the destination range.
- *conv*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by $[first, last]$. This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun (const Type &a);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

- *op*: Specifies the function (or function object) which will be invoked for each of the values

of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

- `init`: The initial value for the generalized sum.

The reduce operations in the parallel `transform_inclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges [first, last) or [result, result + (last - first)).

Return The `transform_inclusive_scan` algorithm returns a `hpx::future<transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `transform_inclusive_scan_result<traits::range_iterator_t<Rng>, O>` otherwise. The `transform_inclusive_scan` algorithm returns an input iterator to one past the end of the range and an output iterator to the element in the destination range, one past the last element copied.

Note `GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN)` is defined as:

- `a1` when `N` is 1
- `op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN))` where $1 < K+1 = M \leq N$.

The behavior of `transform_inclusive_scan` may be non-deterministic for a non-associative predicate.

hpx/parallel/container_algorithms/transform_reduce.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename ExPolicy, typename Rng, typename T, typename Reduce, typename Convert>
util::detail::algorithm_result<ExPolicy, T>::type transform_reduce(ExPolicy &&policy, Rng
&&rng, T init, Reduce &&red_op, Convert
&&conv_op)
```

Returns `GENERALIZED_SUM(red_op, init, conv_op(*first), ..., conv_op(*(first + (last - first) - 1)))`.

The reduce operations in the parallel `transform_reduce` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicates `red_op` and `conv_op`.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *copy_if* requires F to meet the requirements of *CopyConstructible*.
- T: The type of the value to be used as initial (and intermediate) values (deduced).
- Reduce: The type of the binary function object used for the reduction operation.
- Convert: The type of the unary function object used to transform the elements of the input sequence before invoking the reduce function.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- init: The initial value for the generalized sum.
- red_op: Specifies the function (or function object) which will be invoked for each of the values returned from the invocation of *conv_op*. This is a binary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have **const&**, but the function must not modify the objects passed to it. The types *Type1*, *Type2*, and *Ret* must be such that an object of a type as returned from *conv_op* can be implicitly converted to any of those types.

- conv_op: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is a unary predicate. The signature of this predicate should be equivalent to:

```
R fun(const Type &a);
```

The signature does not need to have **const&**, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *Iter* can be dereferenced and then implicitly converted to *Type*. The type *R* must be such that an object of this type can be implicitly converted to *T*.

The reduce operations in the parallel *transform_reduce* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *transform_reduce* and *accumulate* is that the behavior of *transform_reduce* may be non-deterministic for non-associative or non-commutative binary predicate.

Return The *transform_reduce* algorithm returns a *hpx::future<T>* if the execution policy is of type *parallel_task_policy* and returns *T* otherwise. The *transform_reduce* algorithm returns the result of the generalized sum over the values returned from *conv_op* when applied to the elements given by the input range [first, last].

Note GENERALIZED_SUM(op, a1, ..., aN) is defined as follows:

- a1 when N is 1
- op(GENERALIZED_SUM(op, b1, ..., bK), GENERALIZED_SUM(op, bM, ..., bN)), where:
 - b1, ..., bN may be any permutation of a1, ..., aN and

- $1 < K+1 = M \leq N$.

```
template<typename ExPolicy, typename Rng1, typename FwdIter2, typename T>
util::detail::algorithm_result<ExPolicy, T>::type transform_reduce(ExPolicy &&policy, Rng1
&&rng1, FwdIter2 first2, T
init)
```

Returns the result of accumulating init with the inner products of the pairs formed by the elements of two ranges starting at first1 and first2.

The operations in the parallel *transform_reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op2*.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng1**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **FwdIter2**: The type of the second source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **T**: The type of the value to be used as return) values (deduced).

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng1**: Refers to the sequence of elements the algorithm will be applied to.
- **first2**: Refers to the beginning of the second sequence of elements the result will be calculated with.
- **init**: The initial value for the sum.

The operations in the parallel *transform_reduce* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *transform_reduce* algorithm returns a *hpx::future<T>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *T* otherwise.

```
template<typename ExPolicy, typename Rng1, typename FwdIter2, typename T, typename Reduce, typename Convert>
util::detail::algorithm_result<ExPolicy, T>::type transform_reduce(ExPolicy &&policy, Rng1
&&rng1, FwdIter2 first2,
T init, Reduce &&red_op,
Convert &&conv_op)
```

Returns the result of accumulating init with the inner products of the pairs formed by the elements of two ranges starting at first1 and first2.

The operations in the parallel *transform_reduce* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: $O(last - first)$ applications of the predicate *op2*.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `FwdIter2`: The type of the second source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to be used as return) values (deduced).
- `Reduce`: The type of the binary function object used for the multiplication operation.
- `Convert`: The type of the unary function object used to transform the elements of the input sequence before invoking the reduce function.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng1`: Refers to the sequence of elements the algorithm will be applied to.
- `first2`: Refers to the beginning of the second sequence of elements the result will be calculated with.
- `init`: The initial value for the sum.
- `red_op`: Specifies the function (or function object) which will be invoked for the initial value and each of the return values of `op2`. This is a binary predicate. The signature of this predicate should be equivalent to should be equivalent to:

```
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Ret` must be such that it can be implicitly converted to a type of `T`.

- `conv_op`: Specifies the function (or function object) which will be invoked for each of the input values of the sequence. This is a binary predicate. The signature of this predicate should be equivalent to

```
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Ret` must be such that it can be implicitly converted to an object for the second argument type of `op1`.

The operations in the parallel `transform_reduce` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `transform_reduce` algorithm returns a `hpx::future<T>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `T` otherwise.

hpx/parallel/algorithms/transform_reduce_binary.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parallel/container_algorithms/uninitialized_copy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace ranges

Functions

```
template<typename InIter, typename Sent1, typename FwdIter, typename Sent2>
hpx::parallel::util::in_out_result<InIter, FwdIter> uninitialized_copy(InIter first1, Sent1
last1, FwdIter first2,
Sent2 last2)
```

Copies the elements in the range, defined by [first, last), to an uninitialized memory area beginning at *dest*. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel *uninitialized_copy* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *InIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent1*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *FwdIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent2*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter2*.

Parameters

- *first1*: Refers to the beginning of the sequence of elements that will be copied from
- *last1*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied
- *first2*: Refers to the beginning of the destination range.
- *last2*: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

Return The *uninitialized_copy* algorithm returns an *in_out_result<InIter, FwdIter>*. The *uninitialized_copy* algorithm returns an input iterator to one past the last element copied from and the output iterator to the element in the destination range, one past the last element copied.

template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2>

`parallel::util::detail::algorithm_result<ExPolicy, parallel::util::in_out_result<FwdIter1, FwdIter2>>::type uninitialized_copy`

Copies the elements in the range, defined by [first, last), to an uninitialized memory area beginning at *dest*. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel *uninitialized_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent1*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent2*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter2*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first1*: Refers to the beginning of the sequence of elements that will be copied from
- *last1*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *first2*: Refers to the beginning of the destination range.
- *last2*: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

The assignments in the parallel *uninitialized_copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_copy* algorithm returns a `hpx::future<in_out_result<InIter, FwdIter>>`, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `in_out_result<InIter, FwdIter>` otherwise. The *uninitialized_copy* algorithm returns an input iterator to one past the last element copied from and the output iterator to the element in the destination range, one past the last element copied.

`template<typename Rng1, typename Rng2>
hpx::parallel::util::in_out_result<typename hpx::traits::range_traits<Rng1>::iterator_type, typename hpx::traits::range_`

Copies the elements in the range, defined by [first, last), to an uninitialized memory area beginning at

dest. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel *uninitialized_copy* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- Rng1: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- Rng2: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

Parameters

- rng1: Refers to the range from which the elements will be copied from
- rng2: Refers to the range to which the elements will be copied to

Return The *uninitialized_copy* algorithm returns an *in_out_result<typename hpx::traits::range_traits<Rng1> ::iterator_type, typename hpx::traits::range_traits<Rng2> ::iterator_type>*. The *uninitialized_copy* algorithm returns an input iterator to one past the last element copied from and the output iterator to the element in the destination range, one past the last element copied.

template<typename **ExPolicy**, typename **Rng1**, typename **Rng2**>

parallel::util::detail::algorithm_result<ExPolicy, hpx::parallel::util::in_out_result<typename hpx::traits::range_traits<Rng1> ::iterator_type, typename hpx::traits::range_traits<Rng2> ::iterator_type>>

Copies the elements in the range, defined by [first, last), to an uninitialized memory area beginning at *dest*. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel *uninitialized_copy* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- Rng1: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- Rng2: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- rng1: Refers to the range from which the elements will be copied from
- rng2: Refers to the range to which the elements will be copied to

The assignments in the parallel *uninitialized_copy* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_copy* algorithm returns a *hpx::future<in_out_result<InIter, FwdIter>>*, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and

returns `in_out_result< typename hpx::traits::range_traits<Rng1>::iterator_type , typename hpx::traits::range_traits<Rng2>::iterator_type>` otherwise. The `uninitialized_copy` algorithm returns the input iterator to one past the last element copied from and the output iterator to the element in the destination range, one past the last element copied.

```
template<typename InIter, typename Size, typename FwdIter, typename Sent2>
hpx::parallel::util::in_out_result<InIter, FwdIter> uninitialized_copy_n(InIter first1, Size
count, FwdIter
first2, Sent2 last2)
```

Copies the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel `uninitialized_copy_n` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly `last - first` assignments.

Template Parameters

- `InIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- `Size`: The type of the argument specifying the number of elements to apply f to.
- `FwdIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- `Sent2`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.

Parameters

- `first1`: Refers to the beginning of the sequence of elements that will be copied from
- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.
- `first2`: Refers to the beginning of the destination range.
- `last2`: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

Return The `uninitialized_copy_n` algorithm returns `in_out_result<InIter, FwdIter>`. The `uninitialized_copy_n` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter1, typename Size, typename FwdIter2, typename Sent2>
parallel::util::detail::algorithm_result<ExPolicy, parallel::util::in_out_result<FwdIter1, FwdIter2>>::type uninitialized
```

Copies the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel `uninitialized_copy_n` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- `Size`: The type of the argument specifying the number of elements to apply *f* to.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- `Sent2`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter2`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements that will be copied from
- `count`: Refers to the number of elements starting at *first* the algorithm will be applied to.
- `first2`: Refers to the beginning of the destination range.
- `last1`: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

The assignments in the parallel *uninitialized_copy_n* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_copy_n* algorithm returns a `hpx::future<in_out_result<FwdIter1, FwdIter2>>` if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `FwdIter2` otherwise. The *uninitialized_copy_n* algorithm returns the output iterator to the element in the destination range, one past the last element copied.

[hpx/parallel/container_algorithms/uninitialized_default_construct.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

```
template<typename FwdIter, typename Sent>
FwdIter uninitialized_default_construct(FwdIter first, Sent last)
```

Constructs objects of type `typename iterator_traits<ForwardIt>::value_type` in the uninitialized storage designated by the range by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_default_construct* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

- **Sent:** The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.

Parameters

- **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last:** Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.

Return The `uninitialized_default_construct` algorithm returns a returns `FwdIter`. The `uninitialized_default_construct` algorithm returns the output iterator to the element in the range, one past the last element constructed.

```
template<typename ExPolicy, typename FwdIter, typename Sent>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_default_construct (ExPolicy
&&policy,
FwdIter,
first,
Sent
last)
```

Constructs objects of type typename iterator_traits<ForwardIt> ::value_type in the uninitialized storage designated by the range by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel `uninitialized_default_construct` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly `last - first` assignments.

Template Parameters

- **ExPolicy:** The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter:** The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent:** The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.

Parameters

- **policy:** The execution policy to use for the scheduling of the iterations.
- **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last:** Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.

The assignments in the parallel `uninitialized_default_construct` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `uninitialized_default_construct` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `uninitialized_default_construct` algorithm returns the iterator to the element in the source range, one past the last element constructed.

```
template<typename Rng>
hpx::traits::range_traits<Rng>::iterator_type uninitialized_default_construct (Rng
&&rng)
```

Constructs objects of type typename iterator_traits<ForwardIt> ::value_type in the uninitialized storage designated by the range by default-initialization. If an exception is thrown during the initialization,

the function has no effects.

The assignments in the parallel *uninitialized_default_construct* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- *rng*: Refers to the range to which will be default constructed.

Return The *uninitialized_default_construct* algorithm returns a *returns* *hpx::traits::range_traits<Rng> ::iterator_type*. The *uninitialized_default_construct* algorithm returns the output iterator to the element in the range, one past the last element constructed.

```
template<typename ExPolicy, typename Rng>
```

```
parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type uninitialized_
```

Constructs objects of type *typename iterator_traits<ForwardIt> ::value_type* in the uninitialized storage designated by the range by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_default_construct* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the range to which the value will be default constructed

The assignments in the parallel *uninitialized_default_construct* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_default_construct* algorithm returns a *hpx::future<typename hpx::traits::range_traits<Rng> ::iterator_type>*, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *typename hpx::traits::range_traits<Rng>::iterator_type* otherwise. The *uninitialized_default_construct* algorithm returns the output iterator to the element in the range, one past the last element constructed.

```
template<typename FwdIter, typename Size>
```

```
FwdIter uninitialized_default_construct_n(FwdIter first, Size count)
```

Constructs objects of type *typename iterator_traits<ForwardIt> ::value_type* in the uninitialized storage designated by the range [first, first + count) by default-initialization. If an exception is thrown

during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_default_construct_n* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* assignments, if *count* > 0, no assignments otherwise.

Template Parameters

- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *count*: Refers to the number of elements starting at *first* the algorithm will be applied to.

Return The *uninitialized_default_construct_n* algorithm returns a returns *FwdIter*. The *uninitialized_default_construct_n* algorithm returns the iterator to the element in the source range, one past the last element constructed.

```
template<typename ExPolicy, typename FwdIter, typename Size>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_default_construct_n(ExPolicy
&&policy,
FwdIter
first,
Size
count)
```

Constructs objects of type typename iterator_traits<ForwardIt> ::value_type in the uninitialized storage designated by the range [first, first + count) by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_default_construct_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* assignments, if *count* > 0, no assignments otherwise.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *count*: Refers to the number of elements starting at *first* the algorithm will be applied to.

The assignments in the parallel *uninitialized_default_construct_n* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an un-ordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_default_construct_n* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *uninitialized_default_construct_n* algorithm returns the iterator to the element in the source range, one past the last element constructed.

`hpx/parallel/container_algorithms/uninitialized_fill.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

`template<typename FwdIter, typename Sent, typename T>`
`FwdIter uninitialized_fill (FwdIter first, Sent last, T const &value)`

Copies the given *value* to an uninitialized memory area, defined by the range [first, last). If an exception is thrown during the initialization, the function has no effects.

The assignments in the ranges *uninitialized_fill* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*

Template Parameters

- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.
- `T`: The type of the value to be assigned (deduced).

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `value`: The value to be assigned.

Return The *uninitialized_fill* algorithm returns a returns `FwdIter`. The *uninitialized_fill* algorithm returns the output iterator to the element in the range, one past the last element copied.

`template<typename ExPolicy, typename FwdIter, typename Sent>`
`parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_fill (ExPolicy`
`&&policy,`
`FwdIter`
`first,`
`Sent`
`last, T`
`const`
`&value)`

Copies the given *value* to an uninitialized memory area, defined by the range [first, last). If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_fill* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **FwdIter**.
- **T**: The type of the value to be assigned (deduced).

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **value**: The value to be assigned.

The assignments in the parallel *uninitialized_fill* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_fill* algorithm returns a returns *FwdIter*. The *uninitialized_fill* algorithm returns the output iterator to the element in the range, one past the last element copied.

```
template<typename Rng, typename T>
hpx::traits::range_traits<Rng>::iterator_type uninitialized_fill(Rng &&rng, T const
&value)
```

Copies the given *value* to an uninitialized memory area, defined by the range [first, last). If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_fill* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*

Template Parameters

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **T**: The type of the value to be assigned (deduced).

Parameters

- **rng**: Refers to the range to which the value will be filled
- **value**: The value to be assigned.

Return The *uninitialized_fill* algorithm returns a returns *hpx::traits::range_traits<Rng>* ::iterator_type. The *uninitialized_fill* algorithm returns the output iterator to the element in the range, one past the last element copied.

```
template<typename ExPolicy, typename Rng, typename T>
parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type uninit
```

Copies the given *value* to an uninitialized memory area, defined by the range [first, last). If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_fill* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between *first* and *last*

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *T*: The type of the value to be assigned (deduced).

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the range to which the value will be filled
- *value*: The value to be assigned.

The assignments in the parallel *uninitialized_fill* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_fill* algorithm returns a *hpx::future<typename hpx::traits::range_traits<Rng>::iterator_type>*, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *typename hpx::traits::range_traits<Rng>::iterator_type* otherwise. The *uninitialized_fill* algorithm returns the iterator to one past the last element filled in the range.

```
template<typename FwdIter, typename Size, typename T>
FwdIter uninitialized_fill_n(FwdIter first, Size count, T const &value)
```

Copies the given *value* value to the first *count* elements in an uninitialized memory area beginning at *first*. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_fill_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* assignments, if *count* > 0, no assignments otherwise.

Template Parameters

- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.
- *T*: The type of the value to be assigned (deduced).

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *count*: Refers to the number of elements starting at *first* the algorithm will be applied to.
- *value*: The value to be assigned.

Return The *uninitialized_fill_n* algorithm returns a returns *FwdIter*. The *uninitialized_fill_n* algorithm returns the output iterator to the element in the range, one past the last element copied.

```
template<typename ExPolicy, typename FwdIter, typename Size, typename T>
```

```
parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_fill_n(ExPolicy
    &&policy,
    FwdIter
    first,
    Size
    count,
    T
    const
    &value)
```

Copies the given *value* value to the first *count* elements in an uninitialized memory area beginning at *first*. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_fill_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* assignments, if *count* > 0, no assignments otherwise.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.
- *T*: The type of the value to be assigned (deduced).

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *count*: Refers to the number of elements starting at *first* the algorithm will be applied to.
- *value*: The value to be assigned.

The assignments in the parallel *uninitialized_fill_n* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_fill_n* algorithm returns a *hpx::future<FwdIter>*, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *uninitialized_fill_n* algorithm returns the output iterator to the element in the range, one past the last element copied.

hpx/parallel/container_algorithms/uninitialized_move.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename InIter, typename Sent1, typename FwdIter, typename Sent2>
hpx::parallel::util::in_out_result<InIter, FwdIter> uninitialized_move (InIter first1, Sent1
last1, FwdIter first2,
Sent2 last2)
```

Moves the elements in the range, defined by [first, last), to an uninitialized memory area beginning at *dest*. If an exception is thrown during the initialization, some objects in [first, last) are left in a valid but unspecified state.

The assignments in the parallel *uninitialized_move* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *InIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Sent1*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter*.
- *FwdIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent2*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter2*.

Parameters

- *first1*: Refers to the beginning of the sequence of elements that will be moved from
- *last1*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied
- *first2*: Refers to the beginning of the destination range.
- *last2*: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

Return The *uninitialized_move* algorithm returns an *in_out_result<InIter, FwdIter>*. The *uninitialized_move* algorithm returns an input iterator to one past the last element moved from and the output iterator to the element in the destination range, one past the last element moved.

```
template<typename ExPolicy, typename FwdIter1, typename Sent1, typename FwdIter2, typename Sent2>
parallel::util::detail::algorithm_result<ExPolicy, parallel::util::in_out_result<FwdIter1, FwdIter2>>::type uninitialized_
```

Moves the elements in the range, defined by [first, last), to an uninitialized memory area beginning at *dest*. If an exception is thrown during the initialization, some objects in [first, last) are left in a valid but unspecified state.

The assignments in the parallel *uninitialized_move* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- `Sent1`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter`.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- `Sent2`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `InIter2`.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements that will be moved from
- `last1`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- `first2`: Refers to the beginning of the destination range.
- `last2`: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

The assignments in the parallel `uninitialized_move` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `uninitialized_move` algorithm returns a `hpx::future<in_out_result<InIter, FwdIter>>`, if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `in_out_result<InIter, FwdIter>` otherwise. The `uninitialized_move` algorithm returns an input iterator to one past the last element moved from and the output iterator to the element in the destination range, one past the last element moved.

```
template<typename Rng1, typename Rng2>
hpx::parallel::util::in_out_result<typename hpx::traits::range_traits<Rng1>::iterator_type, typename hpx::traits::range_
```

Moves the elements in the range, defined by `[first, last)`, to an uninitialized memory area beginning at `dest`. If an exception is thrown during the initialization, some objects in `[first, last)` are left in a valid but unspecified state.

The assignments in the parallel `uninitialized_move` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- `Rng1`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `Rng2`: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

Parameters

- `rng1`: Refers to the range from which the elements will be moved from
- `rng2`: Refers to the range to which the elements will be moved to

Return The `uninitialized_move` algorithm returns an `in_out_result<typename`

`hpx::traits::range_traits<Rng1> ::iterator_type, typename hpx::traits::range_traits<Rng2> ::iterator_type>. The uninitialized_move algorithm returns an input iterator to one past the last element moved from and the output iterator to the element in the destination range, one past the last element moved.`

```
template<typename ExPolicy, typename Rng1, typename Rng2>
parallel::util::detail::algorithm_result<ExPolicy, hpx::parallel::util::in_out_result<typename hpx::traits::range_traits<Rng1> ::iterator_type, typename hpx::traits::range_traits<Rng2> ::iterator_type>> uninitialized_move(ExPolicy&& policy, Rng1 rng1, Rng2 rng2);
```

Moves the elements in the range, defined by [first, last), to an uninitialized memory area beginning at *dest*. If an exception is thrown during the initialization, some objects in [first, last) are left in a valid but unspecified state.

The assignments in the parallel *uninitialized_move* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng1*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- *Rng2*: The type of the destination range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng1*: Refers to the range from which the elements will be moved from
- *rng2*: Refers to the range to which the elements will be moved to

The assignments in the parallel *uninitialized_move* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_move* algorithm returns a `hpx::future<in_out_result<InIter, FwdIter>>`, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns `in_out_result< typename hpx::traits::range_traits<Rng1>::iterator_type , typename hpx::traits::range_traits<Rng2>::iterator_type >` otherwise. The *uninitialized_move* algorithm returns the input iterator to one past the last element moved from and the output iterator to the element in the destination range, one past the last element moved.

```
template<typename InIter, typename Size, typename FwdIter, typename Sent2>
hpx::parallel::util::in_out_result<InIter, FwdIter> uninitialized_move_n(InIter first1, Size count, FwdIter first2, Sent2 last2)
```

Moves the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest. If an exception is thrown during the initialization, some objects in [first, first + count) are left in a valid but unspecified state.

The assignments in the parallel *uninitialized_move_n* algorithm invoked with an execution policy

object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* movements, if *count* > 0, no move operations otherwise.

Template Parameters

- *InIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.
- *FwdIter*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent2*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.

Parameters

- *first1*: Refers to the beginning of the sequence of elements that will be moved from
- *count*: Refers to the number of elements starting at *first* the algorithm will be applied to.
- *first2*: Refers to the beginning of the destination range.
- *last2*: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

Return The *uninitialized_move_n* algorithm returns *in_out_result<InIter, FwdIter>*. The *uninitialized_move_n* algorithm returns the output iterator to the element in the destination range, one past the last element moved.

```
template<typename ExPolicy, typename FwdIter1, typename Size, typename FwdIter2, typename Sent2>
parallel::util::detail::algorithm_result<ExPolicy, parallel::util::in_out_result<FwdIter1, FwdIter2>>::type uninitialized_
```

Moves the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest. If an exception is thrown during the initialization, some objects in [first, first + count) are left in a valid but unspecified state.

The assignments in the parallel *uninitialized_move_n* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* movements, if *count* > 0, no move operations otherwise.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter1*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.
- *FwdIter2*: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.
- *Sent2*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *InIter2*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.

- `first1`: Refers to the beginning of the sequence of elements that will be moved from.
- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.
- `first2`: Refers to the beginning of the destination range.
- `last1`: Refers to sentinel value denoting the end of the second range the algorithm will be applied to.

The assignments in the parallel `uninitialized_move_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `uninitialized_move_n` algorithm returns a `hpx::future<in_out_result<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter2` otherwise. The `uninitialized_move_n` algorithm returns the output iterator to the element in the destination range, one past the last element moved.

hpx/parallel/container_algorithms/uninitialized_value_construct.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace ranges`

Functions

```
template<typename FwdIter, typename Sent>
FwdIter uninitialized_value_construct(FwdIter first, Sent last)
```

Constructs objects of type `typename iterator_traits<ForwardIt> ::value_type` in the uninitialized storage designated by the range by value-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel `uninitialized_value_construct` algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly `last - first` assignments.

Template Parameters

- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Sent`: The type of the source sentinel (deduced). This sentinel type must be a sentinel for `FwdIter`.

Parameters

- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.

Return The `uninitialized_value_construct` algorithm returns a returns `FwdIter`. The `uninitialized_value_construct` algorithm returns the output iterator to the element in the range, one past the last element constructed.

```
template<typename ExPolicy, typename FwdIter, typename Sent>
```

```
parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_value_construct (ExPolicy  
    &&policy,  
    FwdIter  
    first,  
    Sent  
    last)
```

Constructs objects of type typename iterator_traits<ForwardIt> ::value_type in the uninitialized storage designated by the range by value-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_value_construct* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.

The assignments in the parallel *uninitialized_value_construct* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_value_construct* algorithm returns a *hpx::future<FwdIter>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter* otherwise. The *uninitialized_value_construct* algorithm returns the iterator to the element in the source range, one past the last element constructed.

```
template<typename Rng>  
hpx::traits::range_traits<Rng>::iterator_type uninitialized_value_construct (Rng  
    &&rng)
```

Constructs objects of type typename iterator_traits<ForwardIt> ::value_type in the uninitialized storage designated by the range by value-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_value_construct* algorithm invoked without an execution policy object will execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- *rng*: Refers to the range to which will be value constructed.

Return The *uninitialized_value_construct* algorithm returns a *returns hpx::traits::range_traits<Rng> ::iterator_type*. The *uninitialized_value_construct* algorithm returns the output iterator to the element in the range, one past the last element constructed.

```
template<typename ExPolicy, typename Rng>
parallel::util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type uninitialized_value_construct(
```

Constructs objects of type *typename iterator_traits<ForwardIt> ::value_type* in the uninitialized storage designated by the range by value-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_value_construct* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs exactly *last - first* assignments.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *rng*: Refers to the range to which the value will be value consutrced

The assignments in the parallel *uninitialized_value_construct* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *uninitialized_value_construct* algorithm returns a *hpx::future<typename hpx::traits::range_traits<Rng> ::iterator_type>*, if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *typename hpx::traits::range_traits<Rng>::iterator_type* otherwise. The *uninitialized_value_construct* algorithm returns the output iterator to the element in the range, one past the last element constructed.

```
template<typename FwdIter, typename Size>
FwdIter uninitialized_value_construct_n(FwdIter first, Size count)
```

Constructs objects of type *typename iterator_traits<ForwardIt> ::value_type* in the uninitialized storage designated by the range [first, first + count) by value-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel *uninitialized_value_construct_n* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs exactly *count* assignments, if *count > 0*, no assignments otherwise.

Template Parameters

- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Size*: The type of the argument specifying the number of elements to apply *f* to.

Parameters

- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.

- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.

Return The `uninitialized_value_construct_n` algorithm returns a `FwdIter`. The `uninitialized_value_construct_n` algorithm returns the iterator to the element in the source range, one past the last element constructed.

```
template<typename ExPolicy, typename FwdIter, typename Size>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_value_construct_n(ExPolicy
    &&policy,
    FwdIter
    first,
    Size
    count)
```

Constructs objects of type `typename iterator_traits<ForwardIt> ::value_type` in the uninitialized storage designated by the range `[first, first + count]` by value-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel `uninitialized_value_construct_n` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Performs exactly `count` assignments, if `count > 0`, no assignments otherwise.

Template Parameters

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Size`: The type of the argument specifying the number of elements to apply `f` to.

Parameters

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.

The assignments in the parallel `uninitialized_value_construct_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The `uninitialized_value_construct_n` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `uninitialized_value_construct_n` algorithm returns the iterator to the element in the source range, one past the last element constructed.

hpx/parallel/container_algorithms/unique.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace ranges
```

Functions

```
template<typename FwdIter, typename Sent, typename Pred, typename Proj>
subrange_t<FwdIter, Sent> unique (FwdIter first, Sent last, Pred &&pred, Proj &&proj)
```

Eliminates all but the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.

The assignments in the parallel *unique* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first - 1* applications of the predicate *pred* and no more than twice as many applications of the projection *proj*.

Template Parameters

- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Sent**: The type of the source sentinel (deduced). This sentinel type must be a sentinel for **FwdIter**.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *unique* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an binary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter* can be dereferenced and then implicitly converted to both *Type1* and *Type2*

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *unique* algorithm returns `subrange_t<FwdIter, Sent>`. The *unique* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange.

```
template<typename ExPolicy, typename FwdIter, typename Sent, typename Pred, typename Proj>
parallel::util::detail::algorithm_result<ExPolicy, subrange_t<FwdIter, Sent>>::type unique (ExPolicy
&&pol-
icy,
FwdIter
first,
Sent
last,
Pred
&&pred,
Proj
&&proj)
```

Eliminates all but the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.

The assignments in the parallel *unique* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

Note Complexity: Performs not more than *last - first* assignments, exactly *last - first - 1* applications of the predicate *pred* and no more than twice as many applications of the projection *proj*.

Template Parameters

- *ExPolicy*: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- *FwdIter*: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- *Sent*: The type of the source sentinel (deduced). This sentinel type must be a sentinel for *FwdIter*.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *unique* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to `std::equal_to<>`
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *policy*: The execution policy to use for the scheduling of the iterations.
- *first*: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- *last*: Refers to sentinel value denoting the end of the sequence of elements the algorithm will be applied.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an binary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types *Type1* and *Type2* must be such that objects of types *FwdIter* can be dereferenced and then implicitly converted to both *Type1* and *Type2*

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

The assignments in the parallel *unique* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *unique* algorithm returns *subrange_t<FwdIter, Sent>*. The *unique* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange.

```
template<typename Rng, typename Pred, typename Proj>
subrange_t<hpx::traits::range_iterator_t<Rng>, hpx::traits::range_iterator_t<Rng>> unique (Rng
&&rng,
Pred
&&pred,
Proj
&&proj)
```

Eliminates all but the first element from every consecutive group of equivalent elements from the range *rng* and returns a past-the-end iterator for the new logical end of the range.

The assignments in the parallel *unique* algorithm invoked without an execution policy object execute in sequential order in the calling thread.

Note Complexity: Performs not more than N assignments, exactly N - 1 applications of the predicate *pred* and no more than twice as many applications of the projection *proj*, where N = std::distance(begin(rng), end(rng)).

Template Parameters

- *Rng*: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- *Pred*: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *unique* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to std::equal_to<>
- *Proj*: The type of an optional projection function. This defaults to *util::projection_identity*

Parameters

- *rng*: Refers to the sequence of elements the algorithm will be applied to.
- *pred*: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last]. This is an binary predicate which returns *true* for the required elements. The signature of this predicate should be equivalent to:

```
bool pred(const Type &a, const Type &b);
```

The signature does not need to have *const&*, but the function must not modify the objects passed to it. The type *Type* must be such that an object of type *FwdIter1* can be dereferenced and then implicitly converted to *Type*.

- *proj*: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate *is* invoked.

Return The *unique* algorithm returns *subrange_t<typename hpx::traits::range_iterator<Rng>::type,hpx::traits::range_iterator_t<Rng>>*. The *unique* algorithm returns an object {ret, last}, where ret is a past-the-end iterator for a new subrange.

hpx/parallel/util/low_level.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace util
```

Functions

```
template<typename Value, typename ...Args>
void construct_object (Value *ptr, Args&&... args)
    create an object in the memory specified by ptr
```

Template Parameters

- *Value*: : typename of the object to create
- *Args*: : parameters for the constructor

Parameters

- [in] *ptr*: : pointer to the memory where to create the object
- [in] *args*: : arguments to the constructor

```
template<typename Value>
void destroy_object (Value *ptr)
    destroy an object in the memory specified by ptr
```

Template Parameters

- **Value**: : typename of the object to create

Parameters

- [in] **ptr**: : pointer to the object to destroy

```
template<typename Iter, typename Sent>
void init (Iter first, Sent last, typename std::iterator_traits<Iter>::value_type &val)
    Initialize a range of objects with the object val moving across them
```

Return range initialized

Parameters

- [in] **r**: : range of elements not initialized
- [in] **val**: : object used for the initialization

```
template<typename Value, typename ...Args>
void construct (Value *ptr, Args&&... args)
    create an object in the memory specified by ptr
```

Template Parameters

- **Value**: : typename of the object to create
- **Args**: : parameters for the constructor

Parameters

- [in] **ptr**: : pointer to the memory where to create the object
- [in] **args**: : arguments to the constructor

```
template<typename Iter1, typename Sent1, typename Iter2>
```

```
Iter2 init_move (Iter2 it_dest, Iter1 first, Sent1 last)
```

Move objects.

Template Parameters

- **Iter**: : iterator to the elements
- **Value**: : typename of the object to create

Parameters

- [in] **itdest**: : iterator to the final place of the objects
- [in] **R**: : range to move

```
template<typename Iter, typename Sent, typename Value = typename std::iterator_traits<Iter>::value_type>
Value *uninit_move (Value *ptr, Iter first, Sent last)
```

Move objects to uninitialized memory.

Template Parameters

- **Iter**: : iterator to the elements
- **Value**: : typename of the object to construct

Parameters

- [in] **ptr**: : pointer to the memory where to create the object
- [in] **R**: : range to move

```
template<typename Iter, typename Sent>
```

```
void destroy (Iter first, Sent last)
```

Move objects to uninitialized memory.

Template Parameters

- `Iter`: iterator to the elements
- `Value`: typename of the object to construct

Parameters

- [in] `ptr`: pointer to the memory where to construct the object
- [in] `R`: range to move

```
template<typename Iter1, typename Sent1, typename Iter2, typename Compare>
Iter2 full_merge(Iter1 buf1, Sent1 end_buf1, Iter1 buf2, Sent1 end_buf2, Iter2 buf_out, Compare comp)
```

Merge two contiguous buffers pointed by `buf1` and `buf2`, and put in the buffer pointed by `buf_out`.

Parameters

- [in] `buf1`: iterator to the first element in the first buffer
- [in] `end_buf1`: final iterator of first buffer
- [in] `buf2`: iterator to the first iterator to the second buffer
- [in] `end_buf2`: final iterator of the second buffer
- [in] `buf_out`: buffer where move the elements merged
- [in] `comp`: comparison object

```
template<typename Iter, typename Sent, typename Value, typename Compare>
Value *uninit_full_merge(Iter first1, Sent last1, Iter first2, Sent last2, Value *it_out, Compare comp)
```

Merge two contiguous buffers pointed by `first1` and `first2`, and put in the uninitialized buffer pointed by `it_out`.

Parameters

- [in] `first1`: iterator to the first element in the first buffer
- [in] `last`: last iterator of the first buffer
- [in] `first2`: iterator to the first element to the second buffer
- [in] `last2`: final iterator of the second buffer
- [in] `it_out`: uninitialized buffer where move the elements merged
- [in] `comp`: comparison object

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
Iter2 half_merge(Iter1 buf1, Sent1 end_buf1, Iter2 buf2, Sent2 end_buf2, Iter2 buf_out, Compare comp)
```

: Merge two buffers. The first buffer is in a separate memory. The second buffer have a empty space before `buf2` of the same size than the (`end_buf1 - buf1`)

Remark The elements pointed by `Iter1` and `Iter2` must be the same

Parameters

- [in] `buf1`: iterator to the first element of the first buffer
- [in] `end_buf1`: iterator to the last element of the first buffer
- [in] `buf2`: iterator to the first element of the second buffer
- [in] `end_buf2`: iterator to the last element of the second buffer
- [in] `buf_out`: iterator to the first element to the buffer where put the result
- [in] `comp`: object for Compare two elements of the type pointed by the `Iter1` and `Iter2`

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3, typename Compare>
bool in_place_merge_uncontiguous(Iter1 src1, Sent1 end_src1, Iter2 src2, Sent2 end_src2, Iter3 aux, Compare comp)
```

Merge two non contiguous buffers, placing the results in the buffers for to do this use an auxiliary

buffer pointed by aux

Parameters

- [in] src1: : iterator to the first element of the first buffer
- [in] end_src1: : last iterator of the first buffer
- [in] src2: : iterator to the first element of the second buffer
- [in] end_src2: : last iterator of the second buffer
- [in] aux: : iterator to the first element of the auxiliary buffer
- [in] comp: : object for to Compare elements

Exceptions

-

```
template<typename Iter1, typename Sent1, typename Iter2, typename Compare>
bool in_place_merge(Iter1 src1, Iter1 src2, Sent1 end_src2, Iter2 buf, Compare comp)
: merge two contiguous buffers,using an auxiliary buffer pointed by buf
```

Parameters

- [in] src1: iterator to the first position of the first buffer
- [in] src2: final iterator of the first buffer and first iterator of the second buffer
- [in] end_src2: : final iterator of the second buffer
- [in] buf: : iterator to buffer used as auxiliary memory
- [in] comp: : object for to Compare elements

Exceptions

-

hpx/parallel/util/merge_four.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace util
```

Functions

```
template<typename Iter, typename Sent, typename Compare>
bool less_range(Iter it1, std::uint32_t pos1, Sent it2, std::uint32_t pos2, Compare comp)
Compare the elements pointed by it1 and it2, and if they are equals, compare their position, doing
a stable comparison.
```

Return result of the comparison

Parameters

- [in] it1: : iterator to the first element
- [in] pos1: : position of the object pointed by it1
- [in] it2: : iterator to the second element
- [in] pos2: : position of the element pointed by it2
- [in] comp: : comparison object

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
```

```
util::range<Iter1, Sent1> full_merge4(util::range<Iter1, Sent1> &rdest, util::range<Iter2,
                                         Sent2> vrange_input[4], std::uint32_t nrange_input,
                                         Compare comp)
```

Merge four ranges.

Return range with all the elements move with the size adjusted

Parameters

- [in] dest: range where move the elements merged. Their size must be greater or equal than the sum of the sizes of the ranges in the array R
- [in] R: : array of ranges to merge
- [in] nrange_input: : number of ranges in R
- [in] comp: : comparison object

```
template<typename Value, typename Iter, typename Sent, typename Compare>
util::range<Value*> uninit_full_merge4(util::range<Value*> const &dest,
                                         util::range<Iter, Sent> vrange_input[4],
                                         std::uint32_t nrange_input, Compare comp)
```

Merge four ranges and put the result in uninitialized memory.

Return range with all the elements move with the size adjusted

Parameters

- [in] dest: range where create and move the elements merged. Their size must be greater or equal than the sum of the sizes of the ranges in the array R
- [in] R: : array of ranges to merge
- [in] nrange_input: : number of ranges in vrange_input
- [in] comp: : comparison object

hpx/parallel/util/merge_vector.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace util
```

Functions

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
void merge_level4(util::range<Iter1, Sent1> dest, std::vector<util::range<Iter2, Sent2>>
                  &v_input, std::vector<util::range<Iter1, Sent1>> &v_output, Compare
                  comp)
```

Merge the ranges in the vector v_input using full_merge4. The v_output vector is used as auxiliary memory in the internal process. The final results is in the dest range. All the ranges of v_output are inside the range dest

Return range with all the elements moved

Parameters

- [in] dest: : range where move the elements merged
- [in] v_input: : vector of ranges to merge
- [in] v_output: : vector of ranges obtained

- [in] comp: : comparison object

```
template<typename Value, typename Iter, typename Sent, typename Compare>
void uninit_merge_level4(util::range<Value*> dest, std::vector<util::range<Iter, Sent>>
                        &v_input, std::vector<util::range<Value*>> &v_output,
                        Compare comp)
```

Merge the ranges over uninitialized memory,in the vector v_input using full_merge4. The v_output vector is used as auxiliary memory in the internal process. The final results is in the dest range. All the ranges of v_output are inside the range dest

Return range with all the elements moved

Parameters

- [in] dest: : range where move the elements merged
- [in] v_input: : vector of ranges to merge
- [in] v_output: : vector of ranges obtained
- [in] comp: : comparison object

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
util::range<Iter2, Sent2> merge_vector4(util::range<Iter1, Sent1> range_input,
                                                util::range<Iter2, Sent2> range_output,
                                                std::vector<util::range<Iter1, Sent1>> &v_input,
                                                std::vector<util::range<Iter2, Sent2>> &v_output,
                                                Compare comp)
```

Merge the ranges in the vector v_input using merge_level4. The v_output vector is used as auxiliary memory in the internal process The final results is in the range_output range. All the ranges of v_output are inside the range range_output All the ranges of v_input are inside the range range_input

Parameters

- [in] range_input: : range including all the ranges of v_input
-

hpx/parallel/util/nbits.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace parallel

namespace util

Functions

```
constexpr std::uint32_t nbits32(std::uint32_t num)
```

Obtain the number of bits equal or greater than num.

Return Number of bits

Parameters

- [in] num: : Number to examine

Exceptions

- none:

constexpr std::uint32_t nbits64 (std::uint64_t num)
Obtain the number of bits equal or greater than num.

Return Number of bits

Parameters

- [in] num: : Number to examine

Exceptions

- none:

Variables

hpx/parallel/util/range.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace parallel

namespace util

Typedefs

```
template<typename Iterator, typename Sentinel = Iterator>
using range = hpx::util::iterator_range<Iterator, Sentinel>
```

Functions

```
template<typename Iter, typename Sent>
range<Iter, Sent> concat (range<Iter, Sent> const &it1, range<Iter, Sent> const &it2)
    concatenate two contiguous ranges
```

Return range resulting of the concatenation

Parameters

- [in] `it1`: first range
 - [in] `it2`: second range

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2>
range<Iter2, Iter2> init_move(range<Iter2, Sent2> const &dest, range<Iter1, Sent1>
                                const &src)
```

Return range with the objects moved and the size adjusted

Parameters

- [in] dest: : range where move the objects
 - [in] src: : range from where move the objects

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2>
range<Iter2, Sent2> uninit_move(range<Iter2, Sent2> const &dest, range<Iter1, Sent1>
                                    const &src)
```

Move objects from the range src creating them in dest.

Return range with the objects moved and the size adjusted

Parameters

- [in] dest: : range where move and create the objects
- [in] src: : range from where move the objects

```
template<typename Iter, typename Sent>
void destroy_range(range<Iter, Sent> r)
    destroy a range of objects
```

Parameters

- [in] r: : range to destroy

```
template<typename Iter, typename Sent>
range<Iter, Sent> init(range<Iter, Sent> const &r, typename
                        std::iterator_traits<Iter>::value_type &val)
    initialize a range of objects with the object val moving across them
```

Return range initialized

Parameters

- [in] r: : range of elements not initialized
- [in] val: : object used for the initialization

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
bool is_mergeable(range<Iter1, Sent1> const &src1, range<Iter2, Sent2> const &src2,
                  Compare comp)
    : indicate if two ranges have a possible merge
```

Parameters

- [in] src1: : first range
- [in] src2: : second range
- [in] comp: : object for to compare elements

Exceptions

-

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3, typename Sent3>
range<Iter3, Sent3> full_merge(range<Iter3, Sent3> const &dest, range<Iter1, Sent1>
                                const &src1, range<Iter2, Sent2> const &src2, Compare comp)
```

Merge two contiguous ranges src1 and src2 , and put the result in the range dest, returning the range merged.

Return range with the elements merged and the size adjusted

Parameters

- [in] dest: : range where locate the elements merged. the size of dest must be greater or equal than the sum of the sizes of src1 and src2
- [in] src1: : first range to merge
- [in] src2: : second range to merge
- [in] comp: : comparison object

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Value, typename Compare>
range<Value*> uninit_full_merge(const range<Value*> &dest, range<Iter1, Sent1>
                                const &src1, range<Iter2, Sent2> const &src2,
                                Compare comp)
```

Merge two contiguous ranges src1 and src2 , and create and move the result in the uninitialized range dest, returning the range merged.

Return range with the elements merged and the size adjusted

Parameters

- [in] dest: : range where locate the elements merged. the size of dest must be greater or equal than the sum of the sizes of src1 and src2. Initially is uninitialized memory
- [in] src1: : first range to merge
- [in] src2: : second range to merge
- [in] comp: : comparison object

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
range<Iter2, Sent2> half_merge(range<Iter2, Sent2> const &dest, range<Iter1, Sent1>
                                const &src1, range<Iter2, Sent2> const &src2, Compare comp)
```

: Merge two buffers. The first buffer is in a separate memory

Return : range with the two buffers merged

Parameters

- [in] dest: : range where finish the two buffers merged
- [in] src1: : first range to merge in a separate memory
- [in] src2: : second range to merge, in the final part of the range where deposit the final results
- [in] comp: : object for compare two elements of the type pointed by the Iter1 and Iter2

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Iter3, typename Sent3>
bool in_place_merge_uncontiguous(range<Iter1, Sent1> const &src1, range<Iter2, Sent2> const &src2, range<Iter3, Sent3> &aux,
                                 Compare comp)
```

: merge two non contiguous buffers src1 , src2, using the range aux as auxiliary memory

Parameters

- [in] src1: : first range to merge
- [in] src2: : second range to merge
- [in] aux: : auxiliary range used in the merge
- [in] comp: : object for to compare elements

Exceptions

-

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
range<Iter1, Sent1> in_place_merge(range<Iter1, Sent1> const &src1, range<Iter1, Sent1> const &src2, range<Iter2, Sent2> &buf,
                                      Compare comp)
```

: merge two contiguous buffers (src1, src2) using buf as auxiliary memory

Parameters

- [in] src1: : first range to merge
- [in] src2: : second range to merge
- [in] buf: : auxiliary memory used in the merge

- [in] comp: : object for to compare elements

Exceptions

-

```
template<typename Iter1, typename Sent1, typename Iter2, typename Sent2, typename Compare>
void merge_flow(range<Iter1, Sent1> rng1, range<Iter2, Sent2> rbuf, range<Iter1, Sent1>
                rng2, Compare cmp)
: merge two contiguous buffers
```

Template Parameters

- Iter: : iterator to the elements
- compare: : object for to compare two elements pointed by Iter iterators

Parameters

- [in] first: : iterator to the first element
- [in] last: : iterator to the element after the last in the range
- [in] comp: : object for to compare elements

Exceptions

-

hpx/parallel/util/result_types.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace util
```

Functions

```
template<typename I1, typename I2>
I2 get_in2_element(util::in_in_result<I1, I2> &&p)
```

```
template<typename I1, typename I2>
hpx::future<I2> get_in2_element(hpx::future<util::in_in_result<I1, I2>> &&f)
```

```
template<typename I, typename O>
std::pair<I, O> get_pair(util::in_out_result<I, O> &&p)
```

```
template<typename I, typename O>
O get_second_element(util::in_out_result<I, O> &&p)
```

```
template<typename I, typename O>
hpx::future<std::pair<I, O>> get_pair(hpx::future<util::in_out_result<I, O>> &&f)
```

```
template<typename I, typename O>
hpx::future<O> get_second_element(hpx::future<util::in_out_result<I, O>> &&f)
```

```
template<typename I, typename O>
hpx::util::iterator_range<I, O> get_subrange(in_out_result<I, O> const &ior)
```

```
template<typename I, typename O>
```

```

get_subrange (hpx::future<in_out_result<I,
O>> &&iор)

template<typename I1, typename I2, typename O>
O get_third_element (util::in_out_result<I1, I2, O> &&p)

template<typename I1, typename I2, typename O>
hpx::future<O> get_third_element (hpx::future<util::in_in_out_result<I1, I2, O>> &&f)

template<typename ...Ts>
constexpr in_out_out_result<Ts...> make_in_out_out_result (hpx::tuple<Ts...>
&&t)

template<typename ...Ts>
hpx::future<in_out_out_result<Ts...>> make_in_out_out_result (hpx::future<hpx::tuple<Ts...>>
&&f)

template<typename Iterator, typename Sentinel = Iterator>
hpx::util::iterator_range<Iterator, Sentinel> make_subrange (Iterator iterator, Sentinel sentinel)

template<typename Iterator, typename Sentinel = Iterator>
hpx::future<hpx::util::iterator_range<Iterator, Sentinel>> make_subrange (hpx::future<Iterator>
&&iterator, Sentinel sentinel)

template<typename I, typename F>
struct in_fun_result

```

Public Functions

```

template<typename I2, typename F2, typename Enable = std::enable_if_t<std::is_convertible_v<I const&, I2>>
constexpr operator in_fun_result<I2, F2> () const &

template<typename I2, typename F2, typename Enable = std::enable_if_t<std::is_convertible_v<I, I2> && std::is_convertible_v<F, F2>>
constexpr operator in_fun_result<I2, F2> () &&

template<typename Archive>
void serialize (Archive &ar, unsigned)

```

Public Members

```

HPX_NO_UNIQUE_ADDRESS I hpx::parallel::util::in_fun_result::in
HPX_NO_UNIQUE_ADDRESS F hpx::parallel::util::in_fun_result::fun

template<typename I1, typename I2, typename O>
struct in_in_out_result

```

Public Functions

```
template<typename II1, typename II2, typename O1, typename Enable = typename std::enable_if_t<std::is_
constexpr operator in_in_out_result<II1, II2, O1>() const &

template<typename II2, typename II1, typename O1, typename Enable = typename std::enable_if_t<std::is_
constexpr operator in_in_out_result<II1, II2, O1>() &&

template<typename Archive>
void serialize(Archive &ar, unsigned)
```

Public Members

```
HPX_NO_UNIQUE_ADDRESS I1 hpx::parallel::util::in_in_out_result::in1
HPX_NO_UNIQUE_ADDRESS I2 hpx::parallel::util::in_in_out_result::in2
HPX_NO_UNIQUE_ADDRESS O hpx::parallel::util::in_in_out_result::out

template<typename I1, typename I2>
struct in_in_result
```

Public Functions

```
template<typename II1, typename II2, typename Enable = std::enable_if_t<std::is_convertible_v<I1 const &,
constexpr operator in_in_result<II1, II2>() const &

template<typename II1, typename II2, typename Enable = std::enable_if_t<std::is_convertible_v<I1, II> &&
constexpr operator in_in_result<II1, II2>() &&

template<typename Archive>
void serialize(Archive &ar, unsigned)
```

Public Members

```
HPX_NO_UNIQUE_ADDRESS I1 hpx::parallel::util::in_in_result::in1
HPX_NO_UNIQUE_ADDRESS I2 hpx::parallel::util::in_in_result::in2

template<typename I, typename O1, typename O2>
struct in_out_out_result
```

Public Functions

```
template<typename II, typename OO1, typename OO2, typename Enable = typename std::enable_if_t<std::is_
constexpr operator in_out_out_result<II, OO1, OO2>() const &

template<typename II, typename OO1, typename OO2, typename Enable = typename std::enable_if_t<std::is_
constexpr operator in_out_out_result<II, OO1, OO2>() &&

template<typename Archive>
void serialize(Archive &ar, unsigned)
```

Public Members

```
HPX_NO_UNIQUE_ADDRESS I hpx::parallel::util::in_out_out_result::in
HPX_NO_UNIQUE_ADDRESS O1 hpx::parallel::util::in_out_out_result::out1
HPX_NO_UNIQUE_ADDRESS O2 hpx::parallel::util::in_out_out_result::out2

template<typename I, typename O>
struct in_out_result
```

Public Functions

```
template<typename I2, typename O2, typename Enable = std::enable_if_t<std::is_convertible_v<I const&, I2>>>
constexpr operator in_out_result<I2, O2>() const &

template<typename I2, typename O2, typename Enable = std::enable_if_t<std::is_convertible_v<I, I2> && std::
constexpr operator in_out_result<I2, O2>() &&

template<typename Archive>
void serialize(Archive &ar, unsigned)
```

Public Members

```
HPX_NO_UNIQUE_ADDRESS I hpx::parallel::util::in_out_result::in
HPX_NO_UNIQUE_ADDRESS O hpx::parallel::util::in_out_result::out

template<typename T>
struct min_max_result
```

Public Functions

```
template<typename T2, typename Enable = std::enable_if_t<std::is_convertible_v<T const&, T>>>
constexpr operator min_max_result<T2>() const &

template<typename T2, typename Enable = std::enable_if_t<std::is_convertible_v<T, T2>>>
constexpr operator min_max_result<T2>() &&

template<typename Archive>
void serialize(Archive &ar, unsigned)
```

Public Members

```
HPX_NO_UNIQUE_ADDRESS T hpx::parallel::util::min_max_result::min
HPX_NO_UNIQUE_ADDRESS T hpx::parallel::util::min_max_result::max
```

asio

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/asio/asio_util.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

TypeDefs

```
using endpoint_iterator_type = asio::ip::tcp::resolver::iterator
```

Functions

```
bool get_endpoint (std::string const &addr, std::uint16_t port, asio::ip::tcp::endpoint &ep)
```

```
std::string get_endpoint_name (asio::ip::tcp::endpoint const &ep)
```

```
asio::ip::tcp::endpoint resolve_hostname (std::string const &hostname, std::uint16_t port,  
asio::io_context &io_service)
```

```
std::string resolve_public_ip_address ()
```

```
std::string cleanup_ip_address (std::string const &addr)
```

```
endpoint_iterator_type connect_begin (std::string const &address, std::uint16_t port,  
asio::io_context &io_service)
```

```
template<typename Locality>  
endpoint_iterator_type connect_begin (Locality const &loc, asio::io_context &io_service)  
    Returns an iterator which when dereferenced will give an endpoint suitable for a call to connect()  
    related to this locality.
```

```
endpoint_iterator_type connect_end ()
```

```
endpoint_iterator_type accept_begin (std::string const &address, std::uint16_t port,  
asio::io_context &io_service)
```

```
template<typename Locality>  
endpoint_iterator_type accept_begin (Locality const &loc, asio::io_context &io_service)  
    Returns an iterator which when dereferenced will give an endpoint suitable for a call to accept() related  
    to this locality.
```

```
endpoint_iterator_type accept_end ()
```

```
bool split_ip_address (std::string const &v, std::string &host, std::uint16_t &port)
```

assertion

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/assertion/evaluate_assert.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/modules/assertion.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_ASSERT (*expr*)

This macro asserts that *expr* evaluates to true.

If *expr* evaluates to false, The source location and *msg* is being printed along with the expression and additional. Afterwards the program is being aborted. The assertion handler can be customized by calling `hpx::assertion::set_assertion_handler()`.

Parameters

- *expr*: The expression to assert on. This can either be an expression that's convertible to bool or a callable which returns bool
- *msg*: The optional message that is used to give further information if the assert fails. This should be convertible to a `std::string`

Asserts are enabled if `HPX_DEBUG` is set. This is the default for `CMAKE_BUILD_TYPE=Debug`

HPX_ASSERT_MSG (*expr*, *msg*)

See `HPX_ASSERT`

namespace hpx

namespace assertion

TypeDefs

```
using assertion_handler = void (*) (hpx::source_location const &loc, const char *expr,
                                     std::string const &msg)
```

The signature for an assertion handler.

Functions

```
void set_assertion_handler(assertion_handler handler)
```

Set the assertion handler to be used within a program. If the handler has been set already once, the call to this function will be ignored.

Note This function is not thread safe

async_base

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/async_base/launch_policy.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
struct launch : public detail::policy_holder<>  
#include <launch_policy.hpp> Launch policies for hpx::async etc.
```

Public Functions

```
constexpr launch()
```

Default constructor. This creates a launch policy representing all possible launch modes

```
constexpr launch(detail::async_policy p)
```

Create a launch policy representing asynchronous execution.

```
constexpr launch(detail::fork_policy p)
```

Create a launch policy representing asynchronous execution. The new thread is executed in a preferred way

```
constexpr launch(detail::sync_policy p)
```

Create a launch policy representing synchronous execution.

```
constexpr launch(detail::deferred_policy p)
```

Create a launch policy representing deferred execution.

```
constexpr launch(detail::apply_policy p)
```

Create a launch policy representing fire and forget execution.

```
template<typename F>
```

```
constexpr launch(detail::select_policy<F> const &p)
```

Create a launch policy representing fire and forget execution.

```
template<typename Launch, typename Enable = std::enable_if_t<hpx::traits::is_launch_policy_v<Launch>>>
```

```
constexpr launch(Launch l, threads::thread_priority priority, threads::thread_stacksize stack-size, threads::thread_schedule_hint hint)
```

Public Static Attributes

const detail::async_policy **async**

Predefined launch policy representing asynchronous execution.

const detail::fork_policy **fork**

Predefined launch policy representing asynchronous execution. The new thread is executed in a preferred way

const detail::sync_policy **sync**

Predefined launch policy representing synchronous execution.

const detail::deferred_policy **deferred**

Predefined launch policy representing deferred execution.

const detail::apply_policy **apply**

Predefined launch policy representing fire and forget execution.

const detail::select_policy_generator **select**

Predefined launch policy representing delayed policy selection.

Friends

launch **tag_invoke** (*hpx::execution::experimental*::with_priority_t, launch **const** &*policy*, threads::thread_priority *priority*)

friend constexpr *hpx::threads::thread_priority tag_invoke* (*hpx::execution::experimental*::get_priority_t, launch **const** &*policy*)

launch **tag_invoke** (*hpx::execution::experimental*::with_stacksize_t, launch **const** &*policy*, threads::thread_stacksize *stacksize*)

friend constexpr *hpx::threads::thread_stacksize tag_invoke* (*hpx::execution::experimental*::get_stacksize_t, launch **const** &*policy*)

launch **tag_invoke** (*hpx::execution::experimental*::with_hint_t, launch **const** &*policy*, threads::thread_schedule_hint *hint*)

friend constexpr *hpx::threads::thread_schedule_hint tag_invoke* (*hpx::execution::experimental*::get_hint_t, launch **const** &*policy*)

async_combinators

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/async_combinators/split_future.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

```
template<typename ...Ts>
tuple<future<Ts...>> split_future (future<tuple<Ts...>> &&f)
```

The function *split_future* is an operator allowing to split a given future of a sequence of values (any tuple, std::pair, or std::array) into an equivalent container of futures where each future represents one of the values from the original future. In some sense this function provides the inverse operation of *when_all*.

Return Returns an equivalent container (same container type as passed as the argument) of futures, where each future refers to the corresponding value in the input parameter. All of the returned futures become ready once the input future has become ready. If the input future is exceptional, all output futures will be exceptional as well.

Note The following cases are special:

```
tuple<future<void>> split_future(future<tuple<>> && f);
array<future<void>, 1> split_future(future<array<T, 0>> && f);
```

here the returned futures are directly representing the futures which were passed to the function.

Parameters

- *f*: [in] A future holding an arbitrary sequence of values stored in a tuple-like container. This facility supports *hpx::tuple<>*, *std::pair<T1, T2>*, and *std::array<T, N>*

```
template<typename T>
std::vector<future<T>> split_future (future<std::vector<T>> &&f, std::size_t size)
```

The function *split_future* is an operator allowing to split a given future of a sequence of values (any std::vector) into a std::vector of futures where each future represents one of the values from the original std::vector. In some sense this function provides the inverse operation of *when_all*.

Return Returns a std::vector of futures, where each future refers to the corresponding value in the input parameter. All of the returned futures become ready once the input future has become ready. If the input future is exceptional, all output futures will be exceptional as well.

Parameters

- *f*: [in] A future holding an arbitrary sequence of values stored in a std::vector.
- *size*: [in] The number of elements the vector will hold once the input future has become ready

hpx/async_combinators/wait_all.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename InputIter>
void wait_all (InputIter first, InputIter last)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Note The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

Note The function `wait_all` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_all_nothrow` instead.

Parameters

- `first`: The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `wait_all` should wait.
- `last`: The iterator pointing to the last element of a sequence of `future` or `shared_future` objects for which `wait_all` should wait.

```
template<typename R>
void wait_all (std::vector<future<R>> &&futures)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Note The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

Note The function `wait_all` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_all_nothrow` instead.

Parameters

- `futures`: A vector or array holding an arbitrary amount of `future` or `shared_future` objects for which `wait_all` should wait.

```
template<typename R, std::size_t N>
void wait_all (std::array<future<R>, N> &&futures)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Note The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

Note The function `wait_all` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_all_nothrow` instead.

Parameters

- `futures`: A vector or array holding an arbitrary amount of `future` or `shared_future` objects for which `wait_all` should wait.

```
template<typename ...T>
```

```
void wait_all(T&&... futures)
```

The function *wait_all* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Note The function *wait_all* returns after all futures have become ready. All input futures are still valid after *wait_all* returns.

Note The function *wait_all* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_all_nothrow* instead.

Parameters

- *futures*: An arbitrary number of *future* or *shared_future* objects, possibly holding different types for which *wait_all* should wait.

```
template<typename InputIter>
void wait_all_n(InputIter begin, std::size_t count)
```

The function *wait_all_n* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Return The function *wait_all_n* will return an iterator referring to the first element in the input sequence after the last processed element.

Note The function *wait_all_n* returns after all futures have become ready. All input futures are still valid after *wait_all_n* returns.

Note The function *wait_all_n* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_all_n_nothrow* instead.

Parameters

- *begin*: The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *wait_all_n* should wait.
- *count*: The number of elements in the sequence starting at *first*.

hpx/async_combinators/wait_any.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename InputIter>
void wait_any(InputIter first, InputIter last)
```

The function *wait_any* is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note The function *wait_any* returns after at least one future has become ready. All input futures are still valid after *wait_any* returns.

Note The function *wait_any* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_any_nothrow* instead.

Parameters

- **first:** [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *wait_any* should wait.
- **last:** [in] The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which *wait_any* should wait.

```
template<typename R>
void wait_any(std::vector<future<R>> &futures)
```

The function *wait_any* is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note The function *wait_any* returns after at least one future has become ready. All input futures are still valid after *wait_any* returns.

Note The function *wait_any* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_any_nothrow* instead.

Parameters

- **futures:** [in] A vector holding an arbitrary amount of *future* or *shared_future* objects for which *wait_any* should wait.

```
template<typename R, std::size_t N>
void wait_any(std::array<future<R>, N> &futures)
```

The function *wait_any* is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note The function *wait_any* returns after at least one future has become ready. All input futures are still valid after *wait_any* returns.

Note The function *wait_any* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_any_nothrow* instead.

Parameters

- **futures:** [in] An array holding an arbitrary amount of *future* or *shared_future* objects for which *wait_any* should wait.

```
template<typename ...T>
void wait_any(T&&... futures)
```

The function *wait_any* is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note The function *wait_any* returns after at least one future has become ready. All input futures are still valid after *wait_any* returns.

Note The function *wait_any* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_any_nothrow* instead.

Parameters

- **futures:** [in] An arbitrary number of *future* or *shared_future* objects, possibly holding different types for which *wait_any* should wait.

```
template<typename InputIter>
```

```
void wait_any_n (InputIter first, std::size_t count)
```

The function *wait_any_n* is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note The function *wait_any_n* returns after at least one future has become ready. All input futures are still valid after *wait_any_n* returns.

Note The function *wait_any_n* will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use *wait_any_n_nothrow* instead.

Parameters

- *first*: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *wait_any_n* should wait.
- *count*: [in] The number of elements in the sequence starting at *first*.

hpx/async_combinators/wait_each.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename F, typename Future>
void wait_each (F &&f, std::vector<Future> &&futures)
```

The function *wait_each* is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready. *wait_each* returns after all futures have been become ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that *std::size_t* is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

Parameters

- *f*: The function which will be called for each of the input futures once the future has become ready.
- *futures*: A vector holding an arbitrary amount of *future* or *shared_future* objects for which *wait_each* should wait.

```
template<typename F, typename Iterator>
void wait_each (F &&f, Iterator begin, Iterator end)
```

The function *wait_each* is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready. *wait_each* returns after all futures have been become ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that *std::size_t* is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

Parameters

- *f*: The function which will be called for each of the input futures once the future has become ready.
- *begin*: The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *wait_each* should wait.
- *end*: The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which *wait_each* should wait.

```
template<typename F, typename ...T>
void wait_each (F &&f, T &&... futures)
```

The function *wait_each* is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready. *wait_each* returns after all futures have been become ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that *std::size_t* is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

Parameters

- *f*: The function which will be called for each of the input futures once the future has become ready.
- *futures*: An arbitrary number of *future* or *shared_future* objects, possibly holding different types for which *wait_each* should wait.

```
template<typename F, typename Iterator>
void wait_each_n (F &&f, Iterator begin, std::size_t count)
```

The function *wait_each* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that *std::size_t* is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

Parameters

- *f*: The function which will be called for each of the input futures once the future has become ready.
- *begin*: The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *wait_each_n* should wait.
- *count*: The number of elements in the sequence starting at *first*.

hpx/async_combinators/wait_some.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

```
template<typename InputIter>
void wait_some (std::size_t n, InputIter first, InputIter last)
```

The function `wait_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The function `wait_some` returns after n futures have become ready. All input futures are still valid after `wait_some` returns.

Note The function `wait_some` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_some_nothrow` instead.

Parameters

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the function to return.
- `first`: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which `when_all` should wait.
- `last`: [in] The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which `when_all` should wait.

```
template<typename R>
void wait_some (std::size_t n, std::vector<future<R>> &&futures)
```

The function `wait_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The function `wait_some` returns after n futures have become ready. All input futures are still valid after `wait_some` returns.

Note The function `wait_some` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_some_nothrow` instead.

Parameters

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `futures`: [in] A vector holding an arbitrary amount of *future* or *shared_future* objects for which `wait_some` should wait.

```
template<typename R, std::size_t N>
void wait_some (std::size_t n, std::array<future<R>, N> &&futures)
```

The function `wait_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The function `wait_some` returns after n futures have become ready. All input futures are still valid after `wait_some` returns.

Note The function `wait_some` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_some_nothrow` instead.

Parameters

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `futures`: [in] An array holding an arbitrary amount of `future` or `shared_future` objects for which `wait_some` should wait.

```
template<typename ...T>
void wait_some (std::size_t n, T&&... futures)
```

The function `wait_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The function `wait_all` returns after n futures have become ready. All input futures are still valid after `wait_some` returns.

Note The function `wait_some` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_some_nothrow` instead.

Parameters

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `wait_some` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
template<typename InputIter>
void wait_some_n (std::size_t n, InputIter first, std::size_t count)
```

The function `wait_some_n` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The function `wait_some_n` returns after n futures have become ready. All input futures are still valid after `wait_some_n` returns.

Note The function `wait_some_n` will rethrow any exceptions captured by the futures while becoming ready. If this behavior is undesirable, use `wait_some_n_nothrow` instead.

Parameters

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_all` should wait.
- `count`: [in] The number of elements in the sequence starting at `first`.

hpx/async_combinators/when_all.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

template<typename **InputIter**, typename **Container** = vector<future<typename std::iterator_traits<**InputIter**>::value_type>>
hpx::future<**Container**> **when_all** (**InputIter** first, **InputIter** last)

The function *when_all* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

Return Returns a future holding the same list of futures as has been passed to *when_all*.

- **future<Container<future<R>>>**: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Note Calling this version of *when_all* where *first == last*, returns a future with an empty container that is immediately ready. Each future and *shared_future* is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_all* will not throw an exception, but the futures held in the output collection may.

Parameters

- **first**: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.
- **last**: [in] The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.

template<typename **Range**>
hpx::future<**Range**> **when_all** (**Range** &&values)

The function *when_all* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

Return Returns a future holding the same list of futures as has been passed to *when_all*.

- **future<Container<future<R>>>**: If the input cardinality is unknown at compile time and the futures are all of the same type.

Note Calling this version of *when_all* where the input container is empty, returns a future with an empty container that is immediately ready. Each future and *shared_future* is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_all* will not throw an exception, but the futures held in the output collection may.

Parameters

- **values**: [in] A range holding an arbitrary amount of *future* or *shared_future* objects for which *when_all* should wait.

template<typename ...T>

`hpx::future<hpx::tuple<hpx::future<T>...>> when_all (T&&... futures)`

The function `when_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

Return Returns a future holding the same list of futures as has been passed to `when_all`.

- `future<tuple<future<T0>, future<T1>, future<T2>...>>`: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.
- `future<tuple<>>` if `when_all` is called with zero arguments. The returned future will be initially ready.

Note Each future and `shared_future` is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by `when_all` will not throw an exception, but the futures held in the output collection may.

Parameters

- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `when_all` should wait.

`template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>> hpx::future<Container> when_all_n (InputIter begin, std::size_t count)`

The function `when_all_n` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

Return Returns a future holding the same list of futures as has been passed to `when_all_n`.

- `future<Container<future<R>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output vector will be the same as given by the input iterator.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note None of the futures in the input sequence are invalidated.

Parameters

- `begin`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `wait_all_n` should wait.
- `count`: [in] The number of elements in the sequence starting at `first`.

Exceptions

- `This`: function will throw errors which are encountered while setting up the requested operation only. Errors encountered while executing the operations delivering the results to be stored in the futures are reported through the futures themselves.

hpx/async_combinators/when_any.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>
future<when_any_result<Container>>> when_any (InputIter first, InputIter last)
```

The function `when_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

Return Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<Container<future<R>>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Parameters

- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_any` should wait.
- `last`: [in] The iterator pointing to the last element of a sequence of `future` or `shared_future` objects for which `when_any` should wait.

```
template<typename Range>
future<when_any_result<Range>>> when_any (Range &values)
```

The function `when_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

Return Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<Container<future<R>>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Parameters

- `values`: [in] A range holding an arbitrary amount of `futures` or `shared_future` objects for which `when_any` should wait.

```
template<typename ...T>
future<when_any_result<tuple<future<T>...>>> when_any (T &&... futures)
```

The function `when_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

Return Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future..

- `future<when_any_result<tuple<future<T0>, future<T1>...>>>`: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.
- `future<when_any_result<tuple<>>>` if `when_any` is called with zero arguments. The returned future will be initially ready.

Parameters

- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `when_any` should wait.

```
template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>
future<when_any_result<Container>> when_any_n(InputIter first, std::size_t count)
```

The function `when_any_n` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

Return Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<Container<future<R>>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Note None of the futures in the input sequence are invalidated.

Parameters

- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_any_n` should wait.
- `count`: [in] The number of elements in the sequence starting at `first`.

```
template<typename Sequence>
struct when_any_result
#include <when_any.hpp> Result type for when_any, contains a sequence of futures and an index pointing to a ready future.
```

Public Members

`std::size_t index`

The index of a future which has become ready.

Sequence `futures`

The sequence of futures as passed to `hpx::when_any`

hpx/async_combinators/when_each.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

```
template<typename F, typename Future>
future<void> when_each (F &&f, std::vector<Future> &&futures)
```

The function `when_each` is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a `future` to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the `future` as the second parameter. The first parameter will correspond to the index of the current `future` in the collection.

Return Returns a future representing the event of all input futures being ready.

Parameters

- `f`: The function which will be called for each of the input futures once the future has become ready.
- `futures`: A vector holding an arbitrary amount of `future` or `shared_future` objects for which `wait_each` should wait.

```
template<typename F, typename Iterator>
future<Iterator> when_each (F &&f, Iterator begin, Iterator end)
```

The function `when_each` is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a `future` to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the `future` as the second parameter. The first parameter will correspond to the index of the current `future` in the collection.

Return Returns a future representing the event of all input futures being ready.

Parameters

- `f`: The function which will be called for each of the input futures once the future has become ready.
- `begin`: The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `wait_each` should wait.
- `end`: The iterator pointing to the last element of a sequence of `future` or `shared_future` objects for which `wait_each` should wait.

```
template<typename F, typename ...Ts>
```

```
future<void> when_each (F &&f, Ts&&... futures)
```

The function *when_each* is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that *std::size_t* is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

Return Returns a future representing the event of all input futures being ready.

Parameters

- *f*: The function which will be called for each of the input futures once the future has become ready.
- *futures*: An arbitrary number of *future* or *shared_future* objects, possibly holding different types for which *wait_each* should wait.

```
template<typename F, typename Iterator>
future<Iterator> when_each_n (F &&f, Iterator begin, std::size_t count)
```

The function *when_each* is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that *std::size_t* is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

Return Returns a future holding the iterator pointing to the first element after the last one.

Parameters

- *f*: The function which will be called for each of the input futures once the future has become ready.
- *begin*: The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *wait_each_n* should wait.
- *count*: The number of elements in the sequence starting at *first*.

hpx/async_combinators/when_some.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>
future<when_some_result<Container>> when_some (std::size_t n, Iterator first, Iterator last)
```

The function *when_some* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The future returned by the function *when_some* becomes ready when at least *n* argument futures have become ready.

Return Returns a *when_some_result* holding the same list of futures as has been passed to *when_some* and indices pointing to ready futures.

- *future<when_some_result<Container<future<R>>>>*: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Note Calling this version of *when_some* where *first == last*, returns a future with an empty container that is immediately ready. Each future and *shared_future* is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_some* will not throw an exception, but the futures held in the output collection may.

Parameters

- *n*: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- *first*: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.
- *last*: [in] The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.

```
template<typename Range>
future<when_some_result<Range>> when_some (std::size_t n, Range &&futures)
```

The function *when_some* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

Note The future returned by the function *when_some* becomes ready when at least *n* argument futures have become ready.

Return Returns a *when_some_result* holding the same list of futures as has been passed to *when_some* and indices pointing to ready futures.

- *future<when_some_result<Container<future<R>>>>*: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Note Each future and *shared_future* is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_some* will not throw an exception, but the futures held in the output collection may.

Parameters

- *n*: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.

- **futures:** [in] A container holding an arbitrary amount of *future* or *shared_future* objects for which *when_some* should wait.

```
template<typename ...Ts>
future<when_some_result<tuple<future<T>...>>> when_some (std::size_t n, Ts&&... futures)
```

The function *when_some* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after *n* of them finished executing.

Note The future returned by the function *when_some* becomes ready when at least *n* argument futures have become ready.

Return Returns a *when_some_result* holding the same list of futures as has been passed to *when_some* and an index pointing to a ready future..

- *future<when_some_result<tuple<future<T0>, future<T1>...>>>*: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.
- *future<when_some_result<tuple<>>>* if *when_some* is called with zero arguments. The returned future will be initially ready.

Note Each future and *shared_future* is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_some* will not throw an exception, but the futures held in the output collection may.

Parameters

- *n*: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- **futures:** [in] An arbitrary number of *future* or *shared_future* objects, possibly holding different types for which *when_some* should wait.

```
template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>
future<when_some_result<Container>> when_some_n (std::size_t n, Iterator first, std::size_t count)
```

The function *when_some_n* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after *n* of them finished executing.

Note The future returned by the function *when_some_n* becomes ready when at least *n* argument futures have become ready.

Return Returns a *when_some_result* holding the same list of futures as has been passed to *when_some* and indices pointing to ready futures.

- *future<when_some_result<Container<future<R>>>>*: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Note Calling this version of *when_some_n* where *count == 0*, returns a future with the same elements as the arguments that is immediately ready. Possibly none of the futures in that container are ready. Each future and *shared_future* is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_some_n* will not throw an exception, but the futures held in the output collection may.

Parameters

- *n*: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.

- `first`: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.
- `count`: [in] The number of elements in the sequence starting at *first*.

```
template<typename Sequence>
struct when_some_result
```

`#include <when_some.hpp>` Result type for *when_some*, contains a sequence of futures and indices pointing to ready futures.

Public Members

`std::vector<std::size_t> indices`

List of indices of futures that have become ready.

Sequence `futures`

The sequence of futures as passed to *hpx::when_some*.

async_cuda

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/async_cuda/cublas_executor.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/async_cuda/cuda_executor.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

`namespace hpx`

```
namespace cuda
```

```
namespace experimental
```

```
struct cuda_executor : public hpx::cuda::experimental::cuda_executor_base
```

Public Functions

```
cuda_executor(std::size_t device, bool event_mode = true)
```

```
~cuda_executor()
```

```
template<typename F, typename ...Ts>
decltype(auto) post(F &&f, Ts&&... ts)
```

```
template<typename F, typename ...Ts>
decltype(auto) async_execute(F &&f, Ts&&... ts)
```

Protected Functions

```
template<typename R, typename ...Params, typename ...Args>
void apply(R (*cuda_function)) Params...
    , Args&&... args
```

```
template<typename R, typename ...Params, typename ...Args>
hpx::future<void> async(R (*cuda_kernel)) Params...
    , Args&&... args
```

```
struct cuda_executor_base
```

Subclassed by *hpx::cuda::experimental::cuda_executor*

Public Types

```
using future_type = hpx::future<void>
```

Public Functions

```
cuda_executor_base(std::size_t device, bool event_mode)
future_type get_future()
```

Protected Attributes

```
int device_
bool event_mode_
cudaStream_t stream_
std::shared_ptr<hpx::cuda::experimental::target> target_
```

async_mpi

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/async_mpi/mpi_executor.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace mpi
```

```
namespace experimental
```

```
struct executor
```

Public Types

```
using execution_category = hpx::execution::parallel_execution_tag
using executor_parameters_type = hpx::execution::static_chunk_size
```

Public Functions

```
constexpr executor(MPI_Comm communicator = MPI_COMM_WORLD)

template<typename F, typename ...Ts>
decltype(auto) async_execute (F &&f, Ts&&... ts) const

std::size_t in_flight_estimate () const
```

Private Members

```
MPI_Comm communicator_
```

hpx/async_mpi/transform_mpi.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace mpi
```

```
namespace experimental
```

Variables

```
hpx::mpi::experimental::transform_mpi_t transform_mpi

struct transform_mpi_t : public hpx::functional::detail::tagFallback<transform_mpi_t>
```

Friends

```
template<typename Sender, typename F>
friend constexpr auto tagFallback_invoke (transform_mpi_t, Sender &&s, F
                                         &&f)

template<typename F>
friend constexpr auto tagFallback_invoke (transform_mpi_t, F &&f)
```

cache

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`hpx/cache/local_cache.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

```
template<typename Key, typename Entry, typename UpdatePolicy = std::less<Entry>, typename InsertPolicy = policies::always>
class local_cache
#include <hpx/cache/local_cache.hpp> The local_cache implements the basic functionality
needed for a local (non-distributed) cache.
```

Template Parameters

- **Key:** The type of the keys to use to identify the entries stored in the cache
- **Entry:** The type of the items to be held in the cache, must model the CacheEntry concept
- **UpdatePolicy:** A (optional) type specifying a (binary) function object used to sort the cache entries based on their ‘age’. The ‘oldest’ entries (according to this sorting criteria) will be discarded first if the maximum capacity of the cache is reached. The default is `std::less<Entry>`. The function object will be invoked using 2 entry instances of the type `Entry`. This type must model the `UpdatePolicy` model.
- **InsertPolicy:** A (optional) type specifying a (unary) function object used to allow global decisions whether a particular entry should be added to the cache or not. The default is `policies::always`, imposing no global insert related criteria on the cache. The function object will be invoked using the entry instance to be inserted into the cache. This type must model the `InsertPolicy` model.
- **CacheStorage:** A (optional) container type used to store the cache items. The container must be an associative and STL compatible container. The default is a `std::map<Key, Entry>`.
- **Statistics:** A (optional) type allowing to collect some basic statistics about the operation of the cache instance. The type must conform to the `CacheStatistics` concept. The default value is the type `statistics::no_statistics` which does not collect any numbers, but provides empty stubs allowing the code to compile.

Public Types

```
template<>
using key_type = Key
template<>
using entry_type = Entry
template<>
using update_policy_type = UpdatePolicy
template<>
```

```
using insert_policy_type = InsertPolicy
template<>
using storage_type = CacheStorage
template<>
using statistics_type = Statistics
template<>
using value_type = typename entry_type::value_type
template<>
using size_type = typename storage_type::size_type
template<>
using storage_value_type = typename storage_type::value_type
```

Public Functions

local_cache(size_type *max_size* = 0, update_policy_type **const** &*up* = update_policy_type(), insert_policy_type **const** &*ip* = insert_policy_type())
Construct an instance of a *local_cache*.

Parameters

- *max_size*: [in] The maximal size this cache is allowed to reach any time. The default is zero (no size limitation). The unit of this value is usually determined by the unit of the values returned by the entry's *get_size* function.
- *up*: [in] An instance of the *UpdatePolicy* to use for this cache. The default is to use a default constructed instance of the type as defined by the *UpdatePolicy* template parameter.
- *ip*: [in] An instance of the *InsertPolicy* to use for this cache. The default is to use a default constructed instance of the type as defined by the *InsertPolicy* template parameter.

local_cache(*local_cache* &&*other*)

constexpr size_type **size**() **const**

Return current size of the cache.

Return The current size of this cache instance.

constexpr size_type **capacity**() **const**

Access the maximum size the cache is allowed to grow to.

Note The unit of this value is usually determined by the unit of the return values of the entry's function *entry::get_size*.

Return The maximum size this cache instance is currently allowed to reach. If this number is zero the cache has no limitation with regard to a maximum size.

bool **reserve**(size_type *max_size*)

Change the maximum size this cache can grow to.

Return This function returns *true* if successful. It returns *false* if the new *max_size* is smaller than the current limit and the cache could not be shrunk to the new maximum size.

Parameters

- *max_size*: [in] The new maximum size this cache will be allowed to grow to.

```
bool holds_key (key_type const &k) const
```

Check whether the cache currently holds an entry identified by the given key.

Note This function does not call the entry's function *entry::touch*. It just checks if the cache contains an entry corresponding to the given key.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- k: [in] The key for the entry which should be looked up in the cache.

```
bool get_entry (key_type const &k, key_type &realkey, entry_type &val)
```

Get a specific entry identified by the given key.

Note The function will call the entry's *entry::touch* function if the value corresponding to the provided key is found in the cache.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- k: [in] The key for the entry which should be retrieved from the cache.
- val: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

```
bool get_entry (key_type const &k, entry_type &val)
```

Get a specific entry identified by the given key.

Note The function will call the entry's *entry::touch* function if the value corresponding to the provided key is found in the cache.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- k: [in] The key for the entry which should be retrieved from the cache.
- val: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

```
bool get_entry (key_type const &k, value_type &val)
```

Get a specific entry identified by the given key.

Note The function will call the entry's *entry::touch* function if the value corresponding to the provided is found in the cache.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- k: [in] The key for the entry which should be retrieved from the cache
- val: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding value.

```
bool insert (key_type const &k, value_type const &val)
```

Insert a new element into this cache.

Note This function invokes both, the insert policy as provided to the constructor and the function *entry::insert* of the newly constructed entry instance. If either of these functions returns *false* the key/value pair doesn't get inserted into the cache and the *insert* function will return *false*. Other reasons for this function to fail (return *false*) are a) the key/value pair is already

held in the cache or b) inserting the new value into the cache maxed out its capacity and it was not possible to free any of the existing entries.

Return This function returns *true* if the entry has been successfully added to the cache, otherwise it returns *false*.

Parameters

- *k*: [in] The key for the entry which should be added to the cache.
- *value*: [in] The value which should be added to the cache.

```
bool insert (key_type const &k, value_type &&val)
```

```
template<typename Entry_, std::enable_if_t<std::is_convertible_v<std::decay_t<Entry_>, entry_type>, int> = 0>
bool insert (key_type const &k, Entry_ &&e)
```

Insert a new entry into this cache.

Note This function invokes both, the insert policy as provided to the constructor and the function *entry::insert* of the provided entry instance. If either of these functions returns false the key/value pair doesn't get inserted into the cache and the *insert* function will return *false*. Other reasons for this function to fail (return *false*) are a) the key/value pair is already held in the cache or b) inserting the new value into the cache maxed out its capacity and it was not possible to free any of the existing entries.

Return This function returns *true* if the entry has been successfully added to the cache, otherwise it returns *false*.

Parameters

- *k*: [in] The key for the entry which should be added to the cache.
- *value*: [in] The entry which should be added to the cache.

```
template<typename Value, std::enable_if_t<std::is_convertible_v<std::decay_t<Value>, value_type>, int> = 0>
bool update (key_type const &k, Value &&val)
```

Update an existing element in this cache.

Note The function will call the entry's *entry::touch* function if the indexed value is found in the cache.

Note The difference to the other overload of the *insert* function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

Return This function returns *true* if the entry has been successfully updated, otherwise it returns *false*. If the entry currently is not held by the cache it is added and the return value reflects the outcome of the corresponding insert operation.

Parameters

- *k*: [in] The key for the value which should be updated in the cache.
- *value*: [in] The value which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.

```
template<typename F, typename Value, typename = std::enable_if_t<std::is_convertible_v<std::decay_t<Value>,
```

```
bool update_if (key_type const &k, Value &&val, F &&f)
```

Update an existing element in this cache.

Note The function will call the entry's *entry::touch* function if the indexed value is found in the cache.

Note The difference to the other overload of the *insert* function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

Return This function returns *true* if the entry has been successfully updated, otherwise it returns *false*. If the entry currently is not held by the cache it is added and the return value reflects the outcome of the corresponding insert operation.

Parameters

- *k*: [in] The key for the value which should be updated in the cache.
- *value*: [in] The value which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.
- *f*: [in] A callable taking two arguments, *k* and the key found in the cache (in that order). If *f* returns true, then the update will continue. If *f* returns false, then the update will not succeed.

```
template<typename Entry_, std::enable_if_t<std::is_convertible_v<std::decay_t<Entry>, entry_type>, int> = 0
bool update (key_type const &k, Entry_&&e)
```

Update an existing entry in this cache.

Note The function will call the entry's *entry::touch* function if the indexed value is found in the cache.

Note The difference to the other overload of the *insert* function is that this overload replaces the whole cache entry, while the other overload replaces the cached value only, leaving the cache entry properties untouched.

Return This function returns *true* if the entry has been successfully updated, otherwise it returns *false*. If the entry currently is not held by the cache it is added and the return value reflects the outcome of the corresponding insert operation.

Parameters

- *k*: [in] The key for the entry which should be updated in the cache.
- *value*: [in] The entry which should be used as a replacement for the existing entry in the cache. Any existing entry is first removed and then this entry is added.

```
template<typename Func = policies::always<storage_value_type>>
size_type erase (Func &&ep = Func())
```

Remove stored entries from the cache for which the supplied function object returns true.

Return This function returns the overall size of the removed entries (which is the sum of the values returned by the *entry::get_size* functions of the removed entries).

Parameters

- *ep*: [in] This parameter has to be a (unary) function object. It is invoked for each of the entries currently held in the cache. An entry is considered for removal from the cache whenever the value returned from this invocation is *true*. Even then the entry might not be removed from the cache as its *entry::remove* function might return false.

```
size_type erase ()
```

Remove all stored entries from the cache.

Note All entries are considered for removal, but in the end an entry might not be removed from the cache as its *entry::remove* function might return false. This function is very useful for instance in conjunction with an entry's *entry::remove* function enforcing additional criteria like entry expiration, etc.

Return This function returns the overall size of the removed entries (which is the sum of the values returned by the *entry::get_size* functions of the removed entries).

```
void clear ()
```

Clear the cache.

Unconditionally removes all stored entries from the cache.

```
constexpr statistics_type const &get_statistics() const
    Allow to access the embedded statistics instance.
```

Return This function returns a reference to the statistics instance embedded inside this cache
statistics_type &**get_statistics**()

Protected Functions

```
bool free_space(long num_free)
```

Private Types

```
template<>
using iterator = typename storage_type::iterator

template<>
using const_iterator = typename storage_type::const_iterator

template<>
using heap_type = std::deque<iterator>

template<>
using heap_iterator = typename heap_type::iterator

template<>
using adapted_update_policy_type = adapt<UpdatePolicy, iterator>

template<>
using update_on_exit = typename statistics_type::update_on_exit
```

Private Members

```
size_type max_size_
size_type current_size_
storage_type store_
heap_type entry_heap_
adapted_update_policy_type update_policy_
insert_policy_type insert_policy_
statistics_type statistics_

template<typename Func, typename Iterator>
struct adapt
```

Public Functions

```
template<>
adapt (Func const &f)  

template<>
adapt (Func &&f)  

template<>
bool operator() (Iterator const &lhs, Iterator const &rhs) const
```

Public Members

```
template<>
Func f_
```

`hpx/cache/lru_cache.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

```
template<typename Key, typename Entry, typename Statistics = statistics::no_statisticsclass lru_cache
#include <hpx/cache/lru_cache.hpp> The lru_cache implements the basic functionality needed
for a local (non-distributed) LRU cache.
```

Template Parameters

- **Key**: The type of the keys to use to identify the entries stored in the cache
- **Entry**: The type of the items to be held in the cache.
- **Statistics**: A (optional) type allowing to collect some basic statistics about the operation of the cache instance. The type must conform to the CacheStatistics concept. The default value is the type `statistics::no_statistics` which does not collect any numbers, but provides empty stubs allowing the code to compile.

Public Types

```
template<>
using key_type = Key
template<>
using entry_type = Entry
template<>
using statistics_type = Statistics
template<>
```

```
using entry_pair = std::pair<key_type, entry_type>
template<>
using storage_type = std::list<entry_pair>
template<>
using map_type = std::map<Key, typename storage_type::iterator>
template<>
using size_type = std::size_t
```

Public Functions

lru_cache (size_type *max_size* = 0)

Construct an instance of a *lru_cache*.

Parameters

- *max_size*: [in] The maximal size this cache is allowed to reach any time. The default is zero (no size limitation). The unit of this value is usually determined by the unit of the values returned by the entry's *get_size* function.

lru_cache (*lru_cache* &&*other*)

constexpr size_type **size** () **const**

Return current size of the cache.

Return The current size of this cache instance.

constexpr size_type **capacity** () **const**

Access the maximum size the cache is allowed to grow to.

Note The unit of this value is usually determined by the unit of the return values of the entry's function *entry::get_size*.

Return The maximum size this cache instance is currently allowed to reach. If this number is zero the cache has no limitation with regard to a maximum size.

void reserve (size_type *max_size*)

Change the maximum size this cache can grow to.

Parameters

- *max_size*: [in] The new maximum size this cache will be allowed to grow to.

bool holds_key (key_type **const** &*key*) **const**

Check whether the cache currently holds an entry identified by the given key.

Note This function does not call the entry's function *entry::touch*. It just checks if the cache contains an entry corresponding to the given key.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- *k*: [in] The key for the entry which should be looked up in the cache.

```
bool get_entry (key_type const &key, key_type &realkey, entry_type &entry)
    Get a specific entry identified by the given key.
```

Note The function will “touch” the entry and mark it as recently used if the key was found in the cache.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- **key:** [in] The key for the entry which should be retrieved from the cache.
- **entry:** [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

```
bool get_entry (key_type const &key, entry_type const &entry)
    Get a specific entry identified by the given key.
```

Note The function will “touch” the entry and mark it as recently used if the key was found in the cache.

Return This function returns *true* if the cache holds the referenced entry, otherwise it returns *false*.

Parameters

- **key:** [in] The key for the entry which should be retrieved from the cache.
- **entry:** [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

```
template<typename Entry_, typename = std::enable_if_t<std::is_convertible_v<std::decay_t<Entry_>, entry_type>
bool insert (key_type const &key, Entry_ &&entry)
```

Insert a new entry into this cache.

Note This function assumes that the entry is not in the cache already. Inserting an already existing entry is considered undefined behavior

Parameters

- **key:** [in] The key for the entry which should be added to the cache.
- **entry:** [in] The entry which should be added to the cache.

```
template<typename Entry_, typename = std::enable_if_t<std::is_convertible_v<std::decay_t<Entry_>, entry_type>
void update (key_type const &key, Entry_ &&entry)
```

Update an existing element in this cache.

Note The function will “touch” the entry and mark it as recently used if the key was found in the cache.

Note The difference to the other overload of the *insert* function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

Parameters

- **key:** [in] The key for the value which should be updated in the cache.
- **entry:** [in] The entry which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.

```
template<typename F, typename Entry_, std::enable_if_t<std::is_convertible_v<std::decay_t<Entry_>, entry_type>
bool update_if (key_type const &key, Entry_ &&entry, F &&f)
```

Update an existing element in this cache.

Note The function will “touch” the entry and mark it as recently used if the key was found in the cache.

Note The difference to the other overload of the *insert* function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

Return This function returns *true* if the entry has been successfully updated, otherwise it returns *false*. If the entry currently is not held by the cache it is added and the return value reflects the outcome of the corresponding insert operation.

Parameters

- `key`: [in] The key for the value which should be updated in the cache.
- `entry`: [in] The value which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.
- `f`: [in] A callable taking two arguments, *k* and the key found in the cache (in that order). If *f* returns true, then the update will continue. If *f* returns false, then the update will not succeed.

```
template<typename Func>
size_type erase (Func const &ep)
```

Remove stored entries from the cache for which the supplied function object returns true.

Return This function returns the overall size of the removed entries (which is the sum of the values returned by the *entry::get_size* functions of the removed entries).

Parameters

- `ep`: [in] This parameter has to be a (unary) function object. It is invoked for each of the entries currently held in the cache. An entry is considered for removal from the cache whenever the value returned from this invocation is *true*.

```
size_type erase ()
```

Remove all stored entries from the cache.

Return This function returns the overall size of the removed entries (which is the sum of the values returned by the *entry::get_size* functions of the removed entries).

```
size_type clear ()
```

Clear the cache.

Unconditionally removes all stored entries from the cache.

```
constexpr statistics_type const &get_statistics () const
```

Allow to access the embedded statistics instance.

Return This function returns a reference to the statistics instance embedded inside this cache

```
statistics_type &get_statistics ()
```

Private Types

```
template<>
using update_on_exit = typename statistics_type::update_on_exit
```

Private Functions

```
template<typename Entry_, typename = std::enable_if_t<std::is_convertible_v<std::decay_t<Entry_>, entry_type>>
void insert_nonexist(key_type const &key, Entry_ &&entry)

void touch (typename storage_type::iterator it)

void evict ()
```

Private Members

```
size_type max_size_
size_type current_size_ = 0
storage_type storage_
map_type map_
statistics_type statistics_
```

hpx/cache/entries/entry.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

```
namespace entries
```

```
class entry
```

```
#include <hpx/cache/entries/entry.hpp>
```

Template Parameters

- Value: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.
- Derived: The (optional) type for which this type is used as a base class.

Public Types

`using value_type = Value`

Public Functions

`entry()`

Any cache entry has to be default constructible.

`entry(value_type const &val)`

Construct a new instance of a cache entry holding the given value.

`entry(value_type &&val)`

Construct a new instance of a cache entry holding the given value.

`constexpr bool touch() const`

The function `touch` is called by a cache holding this instance whenever it has been requested (touched).

Note It is possible to change the entry in a way influencing the sort criteria mandated by the `UpdatePolicy`. In this case the function should return `true` to indicate this to the cache, forcing to reorder the cache entries.

Note This function is part of the `CacheEntry` concept

Return This function should return `true` if the cache needs to update its internal heap.

Usually this is needed if the entry has been changed by `touch()` in a way influencing the sort order as mandated by the cache's `UpdatePolicy`

`constexpr bool insert() const`

The function `insert` is called by a cache whenever it is about to be inserted into the cache.

Note This function is part of the `CacheEntry` concept

Return This function should return `true` if the entry should be added to the cache, otherwise it should return `false`.

`constexpr bool remove() const`

The function `remove` is called by a cache holding this instance whenever it is about to be removed from the cache.

Note This function is part of the `CacheEntry` concept

Return The return value can be used to avoid removing this instance from the cache. If the value is `true` it is ok to remove the entry, other wise it will stay in the cache.

`constexpr std::size_t get_size() const`

Return the 'size' of this entry. By default the size of each entry is just one (1), which is sensible if the cache has a limit (capacity) measured in number of entries.

`value_type &get()`

Get a reference to the stored data value.

Note This function is part of the `CacheEntry` concept

`constexpr value_type const &get() const`

Private Members

`value_type value_`

Friends

`bool operator< (entry const &lhs, entry const &rhs)`

Forwarding operator< allowing to compare entries instead of the values.

hpx/cache/entries/fifo_entry.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace util`

`namespace cache`

`namespace entries`

```
template<typename Value>
class fifo_entry : public hpx::util::cache::entries::entry<Value,fifo_entry<Value>>
#include <hpx/cache/entries/fifo_entry.hpp> The fifo_entry type can be used to store arbitrary
values in a cache. Using this type as the cache's entry type makes sure that the least recently
inserted entries are discarded from the cache first.
```

Note The `fifo_entry` conforms to the CacheEntry concept.

Note This type can be used to model a ‘last in first out’ cache policy if it is used with a `std::greater` as the caches’ UpdatePolicy (instead of the default `std::less`).

Template Parameters

- `Value`: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.

Public Functions

`fifo_entry()`

Any cache entry has to be default constructible.

`fifo_entry (Value const &val)`

Construct a new instance of a cache entry holding the given value.

`fifo_entry (Value &&val)`

Construct a new instance of a cache entry holding the given value.

`constexpr bool insert ()`

The function `insert` is called by a cache whenever it is about to be inserted into the cache.

Note This function is part of the CacheEntry concept

Return This function should return *true* if the entry should be added to the cache, otherwise it should return *false*.

```
constexpr time_point const &get_creation_time() const
```

Private Types

```
template<>
using base_type = entry<Value, fifo_entry<Value>>

template<>
using time_point = std::chrono::steady_clock::time_point
```

Private Members

```
time_point insertion_time_
```

Friends

```
bool operator< (fifo_entry const &lhs, fifo_entry const &rhs)
```

Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has been created earlier (FIFO).

hpx/cache/entries/lfu_entry.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

```
namespace entries
```

```
template<typename Value>
class lfu_entry : public hpx::util::cache::entries::entry<Value, lfu_entry<Value>>
#include <hpx/cache/entries/lfu_entry.hpp> The lfu_entry type can be used to store arbitrary values in a cache. Using this type as the cache’s entry type makes sure that the least frequently used entries are discarded from the cache first.
```

Note The `lfu_entry` conforms to the CacheEntry concept.

Note This type can be used to model a ‘most frequently used’ cache policy if it is used with a `std::greater` as the caches’ `UpdatePolicy` (instead of the default `std::less`).

Template Parameters

- `Value`: The data type to be stored in a cache. It has to be default constructible, copy constructible and `less_than_comparable`.

Public Functions

`lfu_entry()`

Any cache entry has to be default constructible.

`lfu_entry(Value const &val)`

Construct a new instance of a cache entry holding the given value.

`lfu_entry(Value &&val)`

Construct a new instance of a cache entry holding the given value.

`bool touch()`

The function `touch` is called by a cache holding this instance whenever it has been requested (touched).

In the case of the LFU entry we store the reference count tracking the number of times this entry has been requested. This which will be used to compare the age of an entry during the invocation of the `operator<()`.

Return This function should return true if the cache needs to update it's internal heap.

Usually this is needed if the entry has been changed by `touch()` in a way influencing the sort order as mandated by the cache's `UpdatePolicy`

```
constexpr unsigned long const &get_access_count() const
```

Private Types

```
template<>
```

```
using base_type = entry<Value, lfu_entry<Value>>
```

Private Members

```
unsigned long ref_count_ = 0
```

Friends

```
bool operator<(lfu_entry const &lhs, lfu_entry const &rhs)
```

Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has been accessed less frequently (LFU).

hpx/cache/entries/lru_entry.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

namespace entries

```
template<typename Value>
class lru_entry : public hpx::util::cache::entries::entry<Value, lru_entry<Value>>
#include <hpx/cache/entries/lru_entry.hpp> The lru_entry type can be used to store arbitrary
values in a cache. Using this type as the cache's entry type makes sure that the least recently
used entries are discarded from the cache first.
```

Note The `lru_entry` conforms to the CacheEntry concept.

Note This type can be used to model a ‘most recently used’ cache policy if it is used with a `std::greater` as the caches’ UpdatePolicy (instead of the default `std::less`).

Template Parameters

- `Value`: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.

Public Functions**lru_entry()**

Any cache entry has to be default constructible.

lru_entry (Value const &val)

Construct a new instance of a cache entry holding the given value.

lru_entry (Value &&val)

Construct a new instance of a cache entry holding the given value.

bool touch()

The function `touch` is called by a cache holding this instance whenever it has been requested (touched).

In the case of the LRU entry we store the time of the last access which will be used to compare the age of an entry during the invocation of the `operator<()`.

Return This function should return true if the cache needs to update its internal heap.

Usually this is needed if the entry has been changed by `touch()` in a way influencing the sort order as mandated by the cache’s UpdatePolicy

constexpr time_point const &get_access_time() const

Returns the last access time of the entry.

Private Types

```
template<>
using base_type = entry<Value, lru_entry<Value>>
```

```
template<>
using time_point = std::chrono::steady_clock::time_point
```

Private Members

time_point **access_time_**

Friends

bool **operator<** (lru_entry **const** &lhs, lru_entry **const** &rhs)
 Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has been accessed less recently (LRU).

hpx/cache/entries/size_entry.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace util

namespace cache

namespace entries

class size_entry

#include <hpx/cache/entries/size_entry.hpp> The **size_entry** type can be used to store values in a cache which have a size associated (such as files, etc.). Using this type as the cache’s entry type makes sure that the entries with the biggest size are discarded from the cache first.

Note The **size_entry** conforms to the CacheEntry concept.

Note This type can be used to model a ‘discard smallest first’ cache policy if it is used with a `std::greater` as the caches’ UpdatePolicy (instead of the default `std::less`).

Template Parameters

- **Value:** The data type to be stored in a cache. It has to be default constructible, copy constructible and `less_than_comparable`.
- **Derived:** The (optional) type for which this type is used as a base class.

Public Functions

size_entry()

Any cache entry has to be default constructible.

size_entry (Value **const &val, std::size_t size = 0)**

Construct a new instance of a cache entry holding the given value.

size_entry (Value &&val, std::size_t size = 0)

Construct a new instance of a cache entry holding the given value.

constexpr std::size_t get_size () const

Return the ‘size’ of this entry.

Private Types

```
using derived_type = typename detail::size_derived<Value, Derived>::type
using base_type = entry<Value, derived_type>
```

Private Members

```
std::size_t size_ = 0
```

Friends

```
friend constexpr bool operator<(size_entry const &lhs, size_entry const &rhs)
```

Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has a bigger size.

hpx/cache/statistics/local_statistics.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

```
namespace statistics
```

```
class local_statistics : public hpx::util::cache::statistics::no_statistics
```

Public Functions

```
local_statistics()

std::size_t get_and_reset(std::size_t &value, bool reset)

constexpr std::size_t hits() const

constexpr std::size_t misses() const

constexpr std::size_t insertions() const

constexpr std::size_t evictions() const

std::size_t hits(bool reset)

std::size_t misses(bool reset)

std::size_t insertions(bool reset)
```

```
std::size_t evictions (bool reset)

void got_hit ()
    The function got_hit will be called by a cache instance whenever a entry got touched.

void got_miss ()
    The function got_miss will be called by a cache instance whenever a requested entry has not
    been found in the cache.

void got_insertion ()
    The function got_insertion will be called by a cache instance whenever a new entry has been
    inserted.

void got_eviction ()
    The function got_eviction will be called by a cache instance whenever an entry has been
    removed from the cache because a new inserted entry let the cache grow beyond its capacity.

void clear ()
    Reset all statistics.
```

Private Members

```
std::size_t hits_ = 0
std::size_t misses_ = 0
std::size_t insertions_ = 0
std::size_t evictions_ = 0
```

hpx/cache/statistics/no_statistics.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_CACHE_METHOD_UNSCOPED_ENUM_DEPRECATED_MSG
```

```
namespace hpx
```

```
namespace util
```

```
namespace cache
```

```
namespace statistics
```

Enums

```
enum method
    Values:
        get_entry = 0
        insert_entry = 1
        update_entry = 2
        erase_entry = 3
```

Variables

```
constexpr method method_get_entry = method::get_entry
constexpr method method_insert_entry = method::insert_entry
constexpr method method_update_entry = method::update_entry
constexpr method method_erase_entry = method::erase_entry

class no_statistics
    Subclassed by hpx::util::cache::statistics::local\_statistics
```

Public Functions

constexpr void got_hit() const

The function *got_hit* will be called by a cache instance whenever a entry got touched.

constexpr void got_miss() const

The function *got_miss* will be called by a cache instance whenever a requested entry has not been found in the cache.

constexpr void got_insertion() const

The function *got_insertion* will be called by a cache instance whenever a new entry has been inserted.

constexpr void got_eviction() const

The function *got_eviction* will be called by a cache instance whenever an entry has been removed from the cache because a new inserted entry let the cache grow beyond its capacity.

constexpr void clear() const

Reset all statistics.

constexpr std::int64_t get_get_entry_count(bool) const

The function *get_get_entry_count* returns the number of invocations of the *get_entry()* API function of the cache.

constexpr std::int64_t get_insert_entry_count(bool) const

The function *get_insert_entry_count* returns the number of invocations of the *insert_entry()* API function of the cache.

constexpr std::int64_t get_update_entry_count(bool) const

The function *get_update_entry_count* returns the number of invocations of the *update_entry()* API function of the cache.

```
constexpr std::int64_t get_erase_entry_count (bool) const
```

The function *get_erase_entry_count* returns the number of invocations of the *erase()* API function of the cache.

```
constexpr std::int64_t get_get_entry_time (bool) const
```

The function *get_get_entry_time* returns the overall time spent executing of the *get_entry()* API function of the cache.

```
constexpr std::int64_t get_insert_entry_time (bool) const
```

The function *get_insert_entry_time* returns the overall time spent executing of the *insert_entry()* API function of the cache.

```
constexpr std::int64_t get_update_entry_time (bool) const
```

The function *get_update_entry_time* returns the overall time spent executing of the *update_entry()* API function of the cache.

```
constexpr std::int64_t get_erase_entry_time (bool) const
```

The function *get_erase_entry_time* returns the overall time spent executing of the *erase()* API function of the cache.

```
struct update_on_exit
```

#include <no_statistics.hpp> Helper class to update timings and counts on function exit.

Public Functions

```
constexpr update_on_exit (no_statistics const&, method)
```

config

See *Public API* for a list of names and headers that are part of the public *HPX API*.

hpx/config/endian.hpp

See *Public API* for a list of names and headers that are part of the public *HPX API*.

coroutines

See *Public API* for a list of names and headers that are part of the public *HPX API*.

hpx/coroutines/thread_enums.hpp

See *Public API* for a list of names and headers that are part of the public *HPX API*.

```
namespace hpx
```

```
namespace threads
```

Enums

enum thread_schedule_state

The *thread_schedule_state* enumerator encodes the current state of a *thread* instance

Values:

unknown = 0

active = 1

 thread is currently active (running, has resources)

Pending = 2

 thread is pending (ready to run, but no hardware resource available)

suspended = 3

 thread has been suspended (waiting for synchronization event, but still known and under control of the thread-manager)

depleted = 4

 thread has been depleted (deeply suspended, it is not known to the thread-manager)

terminated = 5

 thread has been stopped and may be garbage collected

staged = 6

 this is not a real thread state, but allows to reference staged task descriptions, which eventually will be converted into thread objects

Pending_do_not_schedule = 7

Pending_boost = 8

enum thread_priority

This enumeration lists all possible thread-priorities for HPX threads.

Values:

unknown = -1

default = 0

 Will assign the priority of the task to the default (normal) priority.

low = 1

 Task goes onto a special low priority queue and will not be executed until all high/normal priority tasks are done, even if they are added after the low priority task.

normal = 2

 Task will be executed when it is taken from the normal priority queue, this is usually a first-in-first-out ordering of tasks (depending on scheduler choice). This is the default priority.

high_recursive = 3

 The task is a high priority task and any child tasks spawned by this task will be made high priority as well - unless they are specifically flagged as non default priority.

boost = 4

 Same as *thread_priority_high* except that the thread will fall back to *thread_priority_normal* if resumed after being suspended.

high = 5

 Task goes onto a special high priority queue and will be executed before normal/low priority tasks are taken (some schedulers modify the behavior slightly and the documentation for those should be consulted).

bound = 6

Task goes onto a special high priority queue and will never be stolen by another thread after initial assignment. This should be used for thread placement tasks such as OpenMP type for loops.

enum `thread_restart_state`

The `thread_restart_state` enumerator encodes the reason why a thread is being restarted

Values:

unknown = 0**signaled** = 1

The thread has been signaled.

timeout = 2

The thread has been reactivated after a timeout

terminate = 3

The thread needs to be terminated.

abort = 4

The thread needs to be aborted.

enum `thread_stacksize`

A `thread_stacksize` references any of the possible stack-sizes for HPX threads.

Values:

unknown = -1**small_** = 1

use small stack size (the underscore is to work around small being defined to char on Windows)

medium = 2

use medium sized stack size

large = 3

use large stack size

huge = 4

use very large stack size

nostack = 5

this thread does not suspend (does not need a stack)

current = 6

use size of current thread's stack

default_ = *small_*

use default stack size

minimal = *small_*

use minimally stack size

maximal = *huge*

use maximally stack size

enum `thread_schedule_hint_mode`

The type of hint given when creating new tasks.

Values:

none = 0

A hint that leaves the choice of scheduling entirely up to the scheduler.

thread = 1

A hint that tells the scheduler to prefer scheduling a task on the local thread number associated with this hint. Local thread numbers are indexed from zero. It is up to the scheduler to decide how to interpret thread numbers that are larger than the number of threads available to the scheduler. Typically thread numbers will wrap around when too large.

numa = 2

A hint that tells the scheduler to prefer scheduling a task on the NUMA domain associated with this hint. NUMA domains are indexed from zero. It is up to the scheduler to decide how to interpret NUMA domain indices that are larger than the number of available NUMA domains to the scheduler. Typically indices will wrap around when too large.

Functions

`std::ostream &operator<< (std::ostream &os, thread_schedule_state const t)`

`char const *get_thread_state_name (thread_schedule_state state)`

Returns the name of the given state.

Get the readable string representing the name of the given `thread_state` constant.

Parameters

- `state`: this represents the thread state.

`std::ostream &operator<< (std::ostream &os, thread_priority const t)`

`char const *get_thread_priority_name (thread_priority priority)`

Return the thread priority name.

Get the readable string representing the name of the given `thread_priority` constant.

Parameters

- `this`: represents the thread priority.

`std::ostream &operator<< (std::ostream &os, thread_restart_state const t)`

`char const *get_thread_state_ex_name (thread_restart_state state)`

Get the readable string representing the name of the given `thread_restart_state` constant.

`char const *get_thread_state_name (thread_state state)`

Get the readable string representing the name of the given `thread_state` constant.

`std::ostream &operator<< (std::ostream &os, thread_stacksize const t)`

`char const *get_stack_size_enum_name (thread_stacksize size)`

Returns the stack size name.

Get the readable string representing the given stack size constant.

Parameters

- `size`: this represents the stack size

`struct thread_schedule_hint`

`#include <thread_enums.hpp>` A hint given to a scheduler to guide where a task should be scheduled.

A scheduler is free to ignore the hint, or modify the hint to suit the resources available to the scheduler.

Public Functions

constexpr thread_schedule_hint()

Construct a default hint with mode `thread_schedule_hint_mode::none`.

constexpr thread_schedule_hint(`std::int16_t thread_hint`)

Construct a hint with mode `thread_schedule_hint_mode::thread` and the given hint as the local thread number.

constexpr thread_schedule_hint(`thread_schedule_hint_mode mode, std::int16_t hint`)

Construct a hint with the given mode and hint. The numerical hint is unused when the mode is `thread_schedule_hint_mode::none`.

Public Members

`std::int16_t hint`

The hint associated with the mode. The interpretation of this hint depends on the given mode.

`thread_schedule_hint_mode mode`

The mode of the scheduling hint.

hpx/coroutines/thread_id_type.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

template<>
struct hash<::hpx::threads::thread_id>

Public Functions

`std::size_t operator() (::hpx::threads::thread_id const &v) const`

template<>
struct hash<::hpx::threads::thread_id_ref>

Public Functions

`std::size_t operator() (::hpx::threads::thread_id_ref const &v) const`

namespace hpx

namespace threads

Enums

```
enum thread_id_addrf  
Values:
```

yes

no

Variables

```
constexpr const thread_id invalid_thread_id  
struct thread_id
```

Public Functions

```
thread_id()  
thread_id(thread_id const&)  
thread_id &operator=(thread_id const&)  
constexpr thread_id(thread_id &&rhs)  
constexpr thread_id &operator=(thread_id &&rhs)  
constexpr thread_id(thread_id_repr const &thrd)  
constexpr thread_id &operator=(thread_id_repr const &rhs)  
constexpr operator bool() const  
constexpr thread_id_repr get() const  
constexpr void reset()
```

Private Types

```
using thread_id_repr = void*
```

Private Members

```
thread_id_repr thrd_ = nullptr
```

Friends

```

friend constexpr bool operator==(std::nullptr_t, thread_id const &rhs)
friend constexpr bool operator!=(std::nullptr_t, thread_id const &rhs)
friend constexpr bool operator==(thread_id const &lhs, std::nullptr_t)
friend constexpr bool operator!=(thread_id const &lhs, std::nullptr_t)
friend constexpr bool operator==(thread_id const &lhs, thread_id const &rhs)
friend constexpr bool operator!=(thread_id const &lhs, thread_id const &rhs)
friend constexpr bool operator<(thread_id const &lhs, thread_id const &rhs)
friend constexpr bool operator>(thread_id const &lhs, thread_id const &rhs)
friend constexpr bool operator<=(thread_id const &lhs, thread_id const &rhs)
friend constexpr bool operator>=(thread_id const &lhs, thread_id const &rhs)
template<typename Char, typename Traits>
std::basic_ostream<Char, Traits> &operator<<(std::basic_ostream<Char, Traits> &os,
thread_id const &id)
void format_value(std::ostream &os, boost::string_ref spec, thread_id const &id)

```

```
struct thread_id_ref
```

Public Types

```
using thread_repr = detail::thread_data_reference_counting
```

Public Functions

```

thread_id_ref()
thread_id_ref(thread_id_ref const&)
thread_id_ref &operator=(thread_id_ref const&)
thread_id_ref(thread_id_ref &&rhs)
thread_id_ref &operator=(thread_id_ref &&rhs)
thread_id_ref(thread_id_repr const &thrd)
thread_id_ref(thread_id_repr &&thrd)
thread_id_ref &operator=(thread_id_repr const &rhs)
thread_id_ref &operator=(thread_id_repr &&rhs)
thread_id_ref(thread_id_repr *thrd, thread_id_addr addr = thread_id_addr::yes)
thread_id_ref &operator=(thread_id_repr *rhs)

```

```
thread_id_ref(thread_id const &noref)
thread_id_ref(thread_id &&noref)
thread_id_ref &operator=(thread_id const &noref)
thread_id_ref &operator=(thread_id &&noref)

operator bool() const

thread_id noref() const

thread_id_repr &get() &
thread_id_repr &&get() &&
thread_id_repr const &get() const &

void reset()

void reset(thread_repr *thrd, bool add_ref = true)

constexpr thread_repr *detach()
```

Private Types

```
using thread_id_repr = hpx::intrusive_ptr<detail::thread_data_reference_counting>
```

Private Members

```
thread_id_repr thrd_
```

Friends

```
bool operator==(std::nullptr_t, thread_id_ref const &rhs)
bool operator!=(std::nullptr_t, thread_id_ref const &rhs)
bool operator==(thread_id_ref const &lhs, std::nullptr_t)
bool operator!=(thread_id_ref const &lhs, std::nullptr_t)
bool operator==(thread_id_ref const &lhs, thread_id_ref const &rhs)
bool operator!=(thread_id_ref const &lhs, thread_id_ref const &rhs)
bool operator<(thread_id_ref const &lhs, thread_id_ref const &rhs)
bool operator>(thread_id_ref const &lhs, thread_id_ref const &rhs)
bool operator<=(thread_id_ref const &lhs, thread_id_ref const &rhs)
bool operator>=(thread_id_ref const &lhs, thread_id_ref const &rhs)

template<typename Char, typename Traits>
std::basic_ostream<Char, Traits> &operator<< (std::basic_ostream<Char, Traits> &os,
                                                thread_id_ref const &id)
```

```
void format_value(std::ostream &os, boost::string_ref spec, thread_id_ref const &id)  
namespace std
```

```
template<>  
struct hash<::hpx::threads::thread_id>
```

Public Functions

```
std::size_t operator() (::hpx::threads::thread_id const &v) const  
template<>  
struct hash<::hpx::threads::thread_id_ref>
```

Public Functions

```
std::size_t operator() (::hpx::threads::thread_id_ref const &v) const
```

debugging

See *Public API* for a list of names and headers that are part of the public *HPX API*.

hpx/debugging/print.hpp

See *Public API* for a list of names and headers that are part of the public *HPX API*.

Defines

HPX_DP_LAZY (*Expr, printer*)

errors

See *Public API* for a list of names and headers that are part of the public *HPX API*.

hpx/errors/error.hpp

See *Public API* for a list of names and headers that are part of the public *HPX API*.

```
namespace hpx
```

Enums

enum error

Possible error conditions.

This enumeration lists all possible error conditions which can be reported from any of the API functions.

Values:

success = 0

The operation was successful.

no_success = 1

The operation did failed, but not in an unexpected manner.

not_implemented = 2

The operation is not implemented.

out_of_memory = 3

The operation caused an out of memory condition.

bad_action_code = 4

bad_component_type = 5

The specified component type is not known or otherwise invalid.

network_error = 6

A generic network error occurred.

version_too_new = 7

The version of the network representation for this object is too new.

version_too_old = 8

The version of the network representation for this object is too old.

version_unknown = 9

The version of the network representation for this object is unknown.

unknown_component_address = 10

duplicate_component_address = 11

The given global id has already been registered.

invalid_status = 12

The operation was executed in an invalid status.

bad_parameter = 13

One of the supplied parameters is invalid.

internal_server_error = 14

service_unavailable = 15

bad_request = 16

repeated_request = 17

lock_error = 18

duplicate_console = 19

There is more than one console locality.

no_registered_console = 20

There is no registered console locality available.

startup_timed_out = 21

```
uninitialized_value = 22
bad_response_type = 23
deadlock = 24
assertion_failure = 25
null_thread_id = 26
    Attempt to invoke a API function from a non-HPX thread.
invalid_data = 27
yield_aborted = 28
    The yield operation was aborted.
dynamic_link_failure = 29
commandline_option_error = 30
    One of the options given on the command line is erroneous.
serialization_error = 31
    There was an error during serialization of this object.
unhandled_exception = 32
    An unhandled exception has been caught.
kernel_error = 33
    The OS kernel reported an error.
broken_task = 34
    The task associated with this future object is not available anymore.
task_moved = 35
    The task associated with this future object has been moved.
task_already_started = 36
    The task associated with this future object has already been started.
future_already_retrieved = 37
    The future object has already been retrieved.
promise_already_satisfied = 38
    The value for this future object has already been set.
future_does_not_support_cancellation = 39
    The future object does not support cancellation.
future_can_not_be_cancelled = 40
    The future can't be canceled at this time.
no_state = 41
    The future object has no valid shared state.
broken.promise = 42
    The promise has been deleted.
thread_resource_error = 43
future_cancelled = 44
thread_cancelled = 45
thread_not_interruptable = 46
```

```
duplicate_component_id = 47
    The component type has already been registered.

unknown_error = 48
    An unknown error occurred.

bad_plugin_type = 49
    The specified plugin type is not known or otherwise invalid.

filesystem_error = 50
    The specified file does not exist or other filesystem related error.

bad_function_call = 51
    equivalent of std::bad_function_call

task_canceled_exception = 52
    parallel::v2::task_canceled_exception

task_block_not_active = 53
    task_region is not active

out_of_range = 54
    Equivalent to std::out_of_range.

length_error = 55
    Equivalent to std::length_error.

migration_needs_retry = 56
    migration failed because of global race, retry
```

hpx/errors/error_code.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Unnamed Group

```
error_code make_error_code(error e, throwmode mode = throwmode::plain)
    Returns a new error_code constructed from the given parameters.

error_code make_error_code(error e, char const *func, char const *file, long line, throwmode
    mode = throwmode::plain)

error_code make_error_code(error e, char const *msg, throwmode mode = throwmode::plain)
    Returns error_code(e, msg, mode).

error_code make_error_code(error e, char const *msg, char const *func, char const *file, long
    line, throwmode mode = throwmode::plain)

error_code make_error_code(error e, std::string const &msg, throwmode mode = throwmode::plain)
    Returns error_code(e, msg, mode).

error_code make_error_code(error e, std::string const &msg, char const *func, char const
    *file, long line, throwmode mode = throwmode::plain)

error_code make_error_code(std::exception_ptr const &e)
```

Functions

`std::error_category const &get_hpx_category()`

Returns generic HPX error category used for new errors.

`std::error_category const &get_hpx_rethrow_category()`

Returns generic HPX error category used for errors re-thrown after the exception has been de-serialized.

`error_code make_success_code (throwmode mode = throwmode::plain)`

Returns `error_code`(`hpx::success`, “success”, mode).

`class error_code : public error_code`

`#include <error_code.hpp>` A `hpx::error_code` represents an arbitrary error condition.

The class `hpx::error_code` describes an object used to hold error code values, such as those originating from the operating system or other low-level application program interfaces.

Note Class `hpx::error_code` is an adjunct to error reporting by exception

Public Functions

`error_code (throwmode mode = throwmode::plain)`

Construct an object of type `error_code`.

Parameters

- mode: The parameter mode specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if mode is `plain`, this is the default) or to the category `hpx_category_rethrow` (if mode is `rethrow`).

Exceptions

- nothing:

`error_code (error e, throwmode mode = throwmode::plain)`

Construct an object of type `error_code`.

Parameters

- e: The parameter e holds the `hpx::error` code the new exception should encapsulate.
- mode: The parameter mode specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if mode is `plain`, this is the default) or to the category `hpx_category_rethrow` (if mode is `rethrow`).

Exceptions

- nothing:

`error_code (error e, char const *func, char const *file, long line, throwmode mode = throwmode::plain)`

Construct an object of type `error_code`.

Parameters

- e: The parameter e holds the `hpx::error` code the new exception should encapsulate.
- func: The name of the function where the error was raised.
- file: The file name of the code where the error was raised.
- line: The line number of the code line where the error was raised.

- mode: The parameter mode specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if mode is *plain*, this is the default) or to the category `hpx_category_rethrow` (if mode is *rethrow*).

Exceptions

- nothing:

error_code (`error e, char const *msg, throwmode mode = throwmode::plain`)

Construct an object of type `error_code`.

Parameters

- e: The parameter e holds the hpx::error code the new exception should encapsulate.
- msg: The parameter msg holds the error message the new exception should encapsulate.
- mode: The parameter mode specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if mode is *plain*, this is the default) or to the category `hpx_category_rethrow` (if mode is *rethrow*).

Exceptions

- `std::bad_alloc`: (if allocation of a copy of the passed string fails).

error_code (`error e, char const *msg, char const *func, char const *file, long line, throwmode mode = throwmode::plain`)

Construct an object of type `error_code`.

Parameters

- e: The parameter e holds the hpx::error code the new exception should encapsulate.
- msg: The parameter msg holds the error message the new exception should encapsulate.
- func: The name of the function where the error was raised.
- file: The file name of the code where the error was raised.
- line: The line number of the code line where the error was raised.
- mode: The parameter mode specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if mode is *plain*, this is the default) or to the category `hpx_category_rethrow` (if mode is *rethrow*).

Exceptions

- `std::bad_alloc`: (if allocation of a copy of the passed string fails).

error_code (`error e, std::string const &msg, throwmode mode = throwmode::plain`)

Construct an object of type `error_code`.

Parameters

- e: The parameter e holds the hpx::error code the new exception should encapsulate.
- msg: The parameter msg holds the error message the new exception should encapsulate.
- mode: The parameter mode specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if mode is *plain*, this is the default) or to the category `hpx_category_rethrow` (if mode is *rethrow*).

Exceptions

- `std::bad_alloc`: (if allocation of a copy of the passed string fails).

error_code (`error e, std::string const &msg, char const *func, char const *file, long line, throwmode mode = throwmode::plain`)

Construct an object of type `error_code`.

Parameters

- e: The parameter e holds the hpx::error code the new exception should encapsulate.
- msg: The parameter msg holds the error message the new exception should encapsulate.

- `func`: The name of the function where the error was raised.
- `file`: The file name of the code where the error was raised.
- `line`: The line number of the code line where the error was raised.
- `mode`: The parameter `mode` specifies whether the constructed `hpx::error_code` belongs to the error category `hpx_category` (if `mode` is `plain`, this is the default) or to the category `hpx_category_rethrow` (if `mode` is `rethrow`).

Exceptions

- `std::bad_alloc`: (if allocation of a copy of the passed string fails).

`std::string get_message() const`

Return a reference to the error message stored in the `hpx::error_code`.

Exceptions

- `nothing`:

`void clear()`

Clear this `error_code` object. The postconditions of invoking this method are.

- `value() == hpx::success` and `category() == hpx::get_hpx_category()`

`error_code(error_code const &rhs)`

Copy constructor for `error_code`

Note This function maintains the error category of the left hand side if the right hand side is a success code.

`error_code &operator=(error_code const &rhs)`

Assignment operator for `error_code`

Note This function maintains the error category of the left hand side if the right hand side is a success code.

Private Functions

`error_code(int err, hpx::exception const &e)`

`error_code(std::exception_ptr const &e)`

Private Members

`std::exception_ptr exception_`

Friends

```
friend hpx::exception  
error_code make_error_code (std::exception_ptr const &e)
```

hpx/errors/exception.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

TypeDefs

```
using custom_exception_info_handler_type = std::function<hpx::exception_info (std::string  
const&,  
std::string  
const&,  
long,  
std::string  
const&) >  
using pre_exception_handler_type = std::function<void () >
```

Functions

```
void set_custom_exception_info_handler (custom_exception_info_handler_type f)
```

```
void set_pre_exception_handler (pre_exception_handler_type f)
```

```
std::string get_error_what (exception_info const &xi)
```

Return the error message of the thrown exception.

The function `hpx::get_error_what` can be used to extract the diagnostic information element representing the error message as stored in the given exception instance.

Return The error message stored in the exception If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`, `hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()` `hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_config()`, `hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`error get_error(hpx::exception const &e)`

Return the error code value of the exception thrown.

The function `hpx::get_error` can be used to extract the diagnostic information element representing the error value code as stored in the given exception instance.

Return The error value code of the locality where the exception was thrown. If the exception instance does not hold this information, the function will return `hpx::naming::invalid_locality_id`.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`,
`hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`,
`hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `e`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception`, `hpx::error_code`, or `std::exception_ptr`.

Exceptions

- nothing:

`error get_error(hpx::error_code const &e)`

`std::string get_error_function_name(hpx::exception_info const &xi)`

Return the function name from which the exception was thrown.

The function `hpx::get_error_function_name` can be used to extract the diagnostic information element representing the name of the function as stored in the given exception instance.

Return The name of the function from which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_file_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`,
`hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`,
`hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`std::string get_error_file_name(hpx::exception_info const &xi)`

Return the (source code) file name of the function from which the exception was thrown.

The function `hpx::get_error_file_name` can be used to extract the diagnostic information element representing the name of the source file as stored in the given exception instance.

Return The name of the source file of the function from which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`,
`hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`,
`hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

long **get_error_line_number** (`hpx::exception_info const &xi`)

Return the line number in the (source code) file of the function from which the exception was thrown.

The function `hpx::get_error_line_number` can be used to extract the diagnostic information element representing the line number as stored in the given exception instance.

Return The line number of the place where the exception was thrown. If the exception instance does not hold this information, the function will return -1.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_os_thread()`,
`hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`,
`hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- nothing:

class exception : public system_error

#include <exception.hpp> A `hpx::exception` is the main exception type used by HPX to report errors.

The `hpx::exception` type is the main exception type used by HPX to report errors. Any exceptions thrown by functions in the HPX library are either of this type or of a type derived from it. This implies that it is always safe to use this type only in catch statements guarding HPX library calls.

Subclassed by `hpx::exception_list`

Public Functions

exception (*error e = success*)

Construct a *hpx::exception* from a *hpx::error*.

Parameters

- *e*: The parameter *e* holds the *hpx::error* code the new exception should encapsulate.

exception (*std::system_error const &e*)

Construct a *hpx::exception* from a *boost::system_error*.

exception (*std::error_code const &e*)

Construct a *hpx::exception* from a *boost::system::error_code* (this is new for Boost V1.69). This constructor is required to compensate for the changes introduced as a resolution to LWG3162 (<https://cplusplus.github.io/LWG/issue3162>).

exception (*error e, char const *msg, throwmode mode = throwmode::plain*)

Construct a *hpx::exception* from a *hpx::error* and an error message.

Parameters

- *e*: The parameter *e* holds the *hpx::error* code the new exception should encapsulate.
- *msg*: The parameter *msg* holds the error message the new exception should encapsulate.
- *mode*: The parameter *mode* specifies whether the returned *hpx::error_code* belongs to the error category *hpx_category* (if *mode* is *plain*, this is the default) or to the category *hpx_category_rethrow* (if *mode* is *rethrow*).

exception (*error e, std::string const &msg, throwmode mode = throwmode::plain*)

Construct a *hpx::exception* from a *hpx::error* and an error message.

Parameters

- *e*: The parameter *e* holds the *hpx::error* code the new exception should encapsulate.
- *msg*: The parameter *msg* holds the error message the new exception should encapsulate.
- *mode*: The parameter *mode* specifies whether the returned *hpx::error_code* belongs to the error category *hpx_category* (if *mode* is *plain*, this is the default) or to the category *hpx_category_rethrow* (if *mode* is *rethrow*).

~exception()

Destruct a *hpx::exception*

Exceptions

- nothing:

error get_error() const

The function *get_error()* returns the *hpx::error* code stored in the referenced instance of a *hpx::exception*. It returns the *hpx::error* code this exception instance was constructed from.

Exceptions

- nothing:

error_code get_error_code (throwmode mode = throwmode::plain) const

The function *get_error_code()* returns a *hpx::error_code* which represents the same error condition as this *hpx::exception* instance.

Parameters

- mode: The parameter mode specifies whether the returned `hpx::error_code` belongs to the error category `hpx_category` (if mode is `throwmode::plain`, this is the default) or to the category `hpx_category_rethrow` (if mode is `rethrow`).

```
struct thread_interrupted : public exception
```

```
#include <exception.hpp> A hpx::thread_interrupted is the exception type used by HPX to interrupt a running HPX thread.
```

The `hpx::thread_interrupted` type is the exception type used by HPX to interrupt a running thread.

A running thread can be interrupted by invoking the `interrupt()` member function of the corresponding `hpx::thread` object. When the interrupted thread next executes one of the specified interruption points (or if it is currently blocked whilst executing one) with interruption enabled, then a `hpx::thread_interrupted` exception will be thrown in the interrupted thread. If not caught, this will cause the execution of the interrupted thread to terminate. As with any other exception, the stack will be unwound, and destructors for objects of automatic storage duration will be executed.

If a thread wishes to avoid being interrupted, it can create an instance of `hpx::this_thread::disable_interruption`. Objects of this class disable interruption for the thread that created them on construction, and restore the interruption state to whatever it was before on destruction.

```
void f()
{
    // interruption enabled here
    {
        hpx::this_thread::disable_interruption di;
        // interruption disabled
        {
            hpx::this_thread::disable_interruption di2;
            // interruption still disabled
        } // di2 destroyed, interruption state restored
        // interruption still disabled
    } // di destroyed, interruption state restored
    // interruption now enabled
}
```

The effects of an instance of `hpx::this_thread::disable_interruption` can be temporarily reversed by constructing an instance of `hpx::this_thread::restore_interruption`, passing in the `hpx::this_thread::disable_interruption` object in question. This will restore the interruption state to what it was when the `hpx::this_thread::disable_interruption` object was constructed, and then disable interruption again when the `hpx::this_thread::restore_interruption` object is destroyed.

```
void g()
{
    // interruption enabled here
    {
        hpx::this_thread::disable_interruption di;
        // interruption disabled
        {
            hpx::this_thread::restore_interruption ri(di);
            // interruption now enabled
        } // ri destroyed, interruption disable again
    } // di destroyed, interruption state restored
}
```

(continues on next page)

(continued from previous page)

```
// interruption now enabled
}
```

At any point, the interruption state for the current thread can be queried by calling `hpx::this_thread::interruption_enabled()`.

hpx/errors/exception_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_THROWMODE_UNSCOPED_ENUM_DEPRECATED_MSG
namespace hpx
```

Enums

```
enum throwmode
    Encode error category for new error_code.
    Values:
    plain = 0
    rethrow = 1
    lightweight = 0x80
```

Functions

```
constexpr bool operator& (throwmode lhs, throwmode rhs)
```

Variables

```
constexpr throwmode plain = throwmode::plain
constexpr throwmode rethrow = throwmode::rethrow
constexpr throwmode lightweight = throwmode::lightweight
constexpr throwmode lightweight_rethrow = throwmode::lightweight_rethrow
error_code throws
```

Predefined `error_code` object used as “throw on error” tag.

The predefined `hpx::error_code` object `hpx::throws` is supplied for use as a “throw on error” tag.

Functions that specify an argument in the form ‘`error_code& ec=throws`’ (with appropriate namespace qualifiers), have the following error handling semantics:

If `&ec != &throws` and an error occurred: `ec.value()` returns the implementation specific error number for the particular error that occurred and `ec.category()` returns the `error_category` for `ec.value()`.

If `&ec != &throws` and an error did not occur, `ec.clear()`.

If an error occurs and `&ec == &throws`, the function throws an exception of type `hpx::exception` or of a type derived from it. The exception's `get_errorcode()` member function returns a reference to an `hpx::error_code` object with the behavior as specified above.

hpx/errors/exception_list.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`class exception_list : public hpx::exception`

`#include <exception_list.hpp>` The class `exception_list` is a container of `exception_ptr` objects parallel algorithms may use to communicate uncaught exceptions encountered during parallel execution to the caller of the algorithm

The type `exception_list::const_iterator` fulfills the requirements of a forward iterator.

Public Types

`using iterator = exception_list_type::const_iterator`
bidirectional iterator

Public Functions

`std::size_t size() const`

The number of `exception_ptr` objects contained within the `exception_list`.

Note Complexity: Constant time.

`exception_list_type::const_iterator begin() const`

An iterator referring to the first `exception_ptr` object contained within the `exception_list`.

`exception_list_type::const_iterator end() const`

An iterator which is the past-the-end value for the `exception_list`.

hpx/errors/throw_exception.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

`HPX_THROW_EXCEPTION(errcode, f, ...)`

Throw a `hpx::exception` initialized from the given parameters.

The macro `HPX_THROW_EXCEPTION` can be used to throw a `hpx::exception`. The purpose of this macro is to prepend the source file name and line number of the position where the exception is thrown to the error message. Moreover, this associates additional diagnostic information with the exception, such as file name and line number, locality id and thread id, and stack backtrace from the point where the exception was thrown.

The parameter `errcode` holds the `hpx::error` code the new exception should encapsulate. The parameter `f` is expected to hold the name of the function exception is thrown from and the parameter `msg` holds the error message the new exception should encapsulate.

```
void raise_exception()
{
    // Throw a hpx::exception initialized from the given parameters.
    // Additionally associate with this exception some detailed
    // diagnostic information about the throw-site.
    HPX_THROW_EXCEPTION(hpx::no_success, "raise_exception", "simulated error");
}
```

Example:

HPX_THROWS_IF (`ec, errcode, f, ...`)

Either throw a `hpx::exception` or initialize `hpx::error_code` from the given parameters.

The macro `HPX_THROWS_IF` can be used to either throw a `hpx::exception` or to initialize a `hpx::error_code` from the given parameters. If `&ec == &hpx::throws`, the semantics of this macro are equivalent to `HPX_THROW_EXCEPTION`. If `&ec != &hpx::throws`, the `hpx::error_code` instance `ec` is initialized instead.

The parameter `errcode` holds the `hpx::error` code from which the new exception should be initialized. The parameter `f` is expected to hold the name of the function exception is thrown from and the parameter `msg` holds the error message the new exception should encapsulate.

execution

See [Public API](#) for a list of names and headers that are part of the public `HPX` API.

`hpx/execution/executors/auto_chunk_size.hpp`

See [Public API](#) for a list of names and headers that are part of the public `HPX` API.

`namespace hpx`

`namespace execution`

`struct auto_chunk_size`

`#include <auto_chunk_size.hpp>` Loop iterations are divided into pieces and then assigned to threads.

The number of loop iterations combined is determined based on measurements of how long the execution of 1% of the overall number of iterations takes. This executor parameters type makes sure that as many loop iterations are combined as necessary to run for the amount of time specified.

Public Functions

```
constexpr auto_chunk_size(std::uint64_t num_iters_for_timing = 0)  
Construct an auto_chunk_size executor parameters object
```

Note Default constructed *auto_chunk_size* executor parameter types will use 80 microseconds as the minimal time for which any of the scheduled chunks should run.

```
auto_chunk_size(hpx::chrono::steady_duration const &rel_time, std::uint64_t  
                  num_iters_for_timing = 0)  
Construct an auto_chunk_size executor parameters object
```

Parameters

- *rel_time*: [in] The time duration to use as the minimum to decide how many loop iterations should be combined.

[hpx/execution/executors/dynamic_chunk_size.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
struct dynamic_chunk_size  
#include <dynamic_chunk_size.hpp> Loop iterations are divided into pieces of size chunk_size and  
then dynamically scheduled among the threads; when a thread finishes one chunk, it is dynamically  
assigned another If chunk_size is not specified, the default chunk size is 1.
```

Note This executor parameters type is equivalent to OpenMP's DYNAMIC scheduling directive.

Public Functions

```
constexpr dynamic_chunk_size(std::size_t chunk_size = 1)  
Construct a dynamic_chunk_size executor parameters object
```

Parameters

- *chunk_size*: [in] The optional chunk size to use as the number of loop iterations to schedule together. The default chunk size is 1.

hpx/execution/executors/execution.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/execution/executors/execution_information.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace execution
```

Variables

```
hpx::parallel::execution::has_pending_closures_t has_pending_closures
hpx::parallel::execution::get_pu_mask_t get_pu_mask
hpx::parallel::execution::set_scheduler_mode_t set_scheduler_mode

struct get_pu_mask_t : public hpx::functional::detail::tagFallback<get_pu_mask_t>
    #include <execution_information.hpp> Retrieve the bitmask describing the processing units the
    given thread is allowed to run on
    All threads::executors invoke sched.get_pu_mask().
```

Note If the executor does not support this operation, this call will always invoke hpx::threads::get_pu_mask()

Parameters

- exec: [in] The executor object to use for querying the number of pending tasks.
- topo: [in] The topology object to use to extract the requested information.
- thream_num: [in] The sequence number of the thread to retrieve information for.

Private Functions

```
template<typename Executor>
decltype(auto) friend tagFallback_invoke(get_pu_mask_t, Executor&&,  

                                         threads::topology &topo, std::size_t  

                                         thread_num)

template<typename Executor>
decltype(auto) friend tag_invoke(get_pu_mask_t, Executor &&exec, threads::topology  

                                         &topo, std::size_t thread_num)

struct has_pending_closures_t : public hpx::functional::detail::tagFallback<has_pending_closures_t>
    #include <execution_information.hpp> Retrieve whether this executor has operations pending or
    not.
```

Note If the executor does not expose this information, this call will always return *false*

Parameters

- `exec`: [in] The executor object to use to extract the requested information for.

Private Functions

```
template<typename Executor>
decltype(auto) friend tagFallback_invoke(has_pending_closures_t, Executor&&)

template<typename Executor>
decltype(auto) friend tag_invoke(has_pending_closures_t, Executor &&exec)

struct set_scheduler_mode_t : public hpx::functional::detail::tagFallback<set_scheduler_mode_t>
#include <execution_information.hpp> Set various modes of operation on the scheduler underneath the given executor.
```

Note This calls `exec.set_scheduler_mode(mode)` if it exists; otherwise it does nothing.

Parameters

- `exec`: [in] The executor object to use.
- `mode`: [in] The new mode for the scheduler to pick up

Friends

```
template<typename Executor, typename Mode>
void tagFallback_invoke(set_scheduler_mode_t, Executor&&, Mode const&)

template<typename Executor, typename Mode>
void tag_invoke(set_scheduler_mode_t, Executor &&exec, Mode const &mode)
```

hpx/execution/executors/execution_parameters.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace execution
```

Functions

```
template<typename ...Params>
constexpr executor_parameters_join<Params...>::type join_executor_parameters(Params&&... params)

template<typename Param>
constexpr Param &&join_executor_parameters(Param &&param)

template<typename ...Params>
struct executor_parameters_join
```

Public Types

```
template<>
using type = detail::executor_parameters<std::decay_t<Params>...>

template<typename Param>
struct executor_parameters_join<Param>
```

Public Types

```
template<>
using type = Param
```

`hpx/execution/executors/execution_parameters_fwd.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace execution
```

Variables

```
hpx::parallel::execution::get_chunk_size_t get_chunk_size
hpx::parallel::execution::maximal_number_of_chunks_t maximal_number_of_chunks
hpx::parallel::execution::reset_thread_distribution_t reset_thread_distribution
hpx::parallel::execution::processing_units_count_t processing_units_count
hpx::parallel::execution::with_processing_units_count_t with_processing_units_count
hpx::parallel::execution::mark_begin_execution_t mark_begin_execution
hpx::parallel::execution::mark_end_of_scheduling_t mark_end_of_scheduling
hpx::parallel::execution::mark_end_execution_t mark_end_execution

struct get_chunk_size_t : public hpx::functional::detail::tag_fallback<get_chunk_size_t>
    #include <execution_parameters_fwd.hpp> Return the number of invocations of the given function f which should be combined into a single task
```

Note The parameter *f* is expected to be a nullary function returning a `std::size_t` representing the number of iteration the function has already executed (i.e. which don't have to be scheduled anymore).

Parameters

- `params`: [in] The executor parameters object to use for determining the chunk size for the given number of tasks `num_tasks`.
- `exec`: [in] The executor object which will be used for scheduling of the loop iterations.
- `f`: [in] The function which will be optionally scheduled using the given executor.
- `cores`: [in] The number of cores the number of chunks should be determined for.

- num_tasks: [in] The number of tasks the chunk size should be determined for

Private Functions

```
template<typename Parameters, typename Executor, typename F>
decltype(auto) friend tagFallback_invoke(get_chunk_size_t,           Parameters
                                         &&params, Executor &&exec, F
                                         &&f, std::size_t cores, std::size_t
                                         num_tasks)

struct mark_begin_execution_t : public hpx::functional::detail::tag_fallback<mark_begin_execution_t>
#include <execution_parameters_fwd.hpp> Mark the begin of a parallel algorithm execution
```

Note This calls params.mark_begin_execution(exec) if it exists; otherwise it does nothing.

Parameters

- params: [in] The executor parameters object to use as a fallback if the executor does not expose

Private Functions

```
template<typename Parameters, typename Executor>
decltype(auto) friend tagFallback_invoke(mark_begin_execution_t, Parameters
                                         &&params, Executor &&exec)

struct mark_end_execution_t : public hpx::functional::detail::tag_fallback<mark_end_execution_t>
#include <execution_parameters_fwd.hpp> Mark the end of a parallel algorithm execution
```

Note This calls params.mark_end_execution(exec) if it exists; otherwise it does nothing.

Parameters

- params: [in] The executor parameters object to use as a fallback if the executor does not expose

Private Functions

```
template<typename Parameters, typename Executor>
decltype(auto) friend tagFallback_invoke(mark_end_execution_t, Parameters
                                         &&params, Executor &&exec)

struct mark_end_of_scheduling_t : public hpx::functional::detail::tag_fallback<mark_end_of_scheduling_t>
#include <execution_parameters_fwd.hpp> Mark the end of scheduling tasks during parallel algorithm execution
```

Note This calls params.mark_begin_execution(exec) if it exists; otherwise it does nothing.

Parameters

- params: [in] The executor parameters object to use as a fallback if the executor does not expose

Private Functions

```
template<typename Parameters, typename Executor>
decltype(auto) friend tagFallback_invoke(mark_end_of_scheduling_t, Parameters &&params, Executor &&exec)

struct maximal_number_of_chunks_t : public hpx::functional::detail::tagFallback<maximal_number_of_chunks_t>
#include <execution_parameters_fwd.hpp> Return the largest reasonable number of chunks to
create for a single algorithm invocation.
```

Parameters

- params: [in] The executor parameters object to use for determining the number of chunks for the given number of cores.
- exec: [in] The executor object which will be used for scheduling of the loop iterations.
- cores: [in] The number of cores the number of chunks should be determined for.
- num_tasks: [in] The number of tasks the chunk size should be determined for

Private Functions

```
template<typename Parameters, typename Executor>
decltype(auto) friend tagFallback_invoke(maximal_number_of_chunks_t, Parameters &&params, Executor &&exec, std::size_t cores, std::size_t num_tasks)

struct processing_units_count_t : public hpx::functional::detail::tagFallback<processing_units_count_t>
#include <execution_parameters_fwd.hpp> Retrieve the number of (kernel-)threads used by the
associated executor.
```

Note This calls params.processing_units_count(Executor&&) if it exists; otherwise it forwards the request to the executor parameters object.

Parameters

- params: [in] The executor parameters object to use as a fallback if the executor does not expose

Private Functions

```
template<typename Parameters, typename Executor>
decltype(auto) friend tagFallback_invoke(processing_units_count_t, Parameters &&params, Executor &&exec)

struct reset_thread_distribution_t : public hpx::functional::detail::tagFallback<reset_thread_distribution_t>
#include <execution_parameters_fwd.hpp> Reset the internal round robin thread distribution
scheme for the given executor.
```

Note This calls params.reset_thread_distribution(exec) if it exists; otherwise it does nothing.

Parameters

- params: [in] The executor parameters object to use for resetting the thread distribution scheme.
- exec: [in] The executor object to use.

Private Functions

```
template<typename Parameters, typename Executor>
decltype(auto) friend tagFallback_invoke(resetThreadDistribution_t, Parameters &&params, Executor &&exec)

struct with_processing_units_count_t : public hpx::functional::detail::tagFallback<with_processing_units_count_t>
#include <execution_parameters_fwd.hpp> Generate a policy that supports setting the number of cores for execution.
```

hpx/execution/executors/guided_chunk_size.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
struct guided_chunk_size
#include <guided_chunk_size.hpp> Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned. Similar to dynamic\_chunk\_size except that the block size decreases each time a number of loop iterations is given to a thread. The size of the initial block is proportional to  $number\_of\_iterations / number\_of\_cores$ . Subsequent blocks are proportional to  $number\_of\_iterations\_remaining / number\_of\_cores$ . The optional chunk size parameter defines the minimum block size. The default chunk size is 1.
```

Note This executor parameters type is equivalent to OpenMP's GUIDED scheduling directive.

Public Functions

```
constexpr guided_chunk_size (std::size_t min_chunk_size = 1)
Construct a guided\_chunk\_size executor parameters object
```

Parameters

- `min_chunk_size`: [in] The optional minimal chunk size to use as the minimal number of loop iterations to schedule together. The default minimal chunk size is 1.

hpx/execution/executors/num_cores.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
struct num_cores
#include <num_cores.hpp> Control number of cores in executors which need a functionality for setting the number of cores to be used by an algorithm directly
```

Public Functions

```
constexpr num_cores (std::size_t cores = 1)
Construct a num_cores executor parameters object
```

Note make sure the minimal number of cores is 1

hpx/execution/executors/persistent_auto_chunk_size.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
struct persistent_auto_chunk_size
#include <persistent_auto_chunk_size.hpp> Loop iterations are divided into pieces and then assigned to threads. The number of loop iterations combined is determined based on measurements of how long the execution of 1% of the overall number of iterations takes. This executor parameters type makes sure that as many loop iterations are combined as necessary to run for the amount of time specified.
```

Public Functions

```
constexpr persistent_auto_chunk_size (std::uint64_t num_iters_for_timing = 0)
Construct an persistent_auto_chunk_size executor parameters object
```

Note Default constructed *persistent_auto_chunk_size* executor parameter types will use 0 microseconds as the execution time for each chunk and 80 microseconds as the minimal time for which any of the scheduled chunks should run.

```
persistent_auto_chunk_size (hpx::chrono::steady_duration const &time_cs,
std::uint64_t num_iters_for_timing = 0)
Construct an persistent_auto_chunk_size executor parameters object
```

Parameters

- *time_cs*: The execution time for each chunk.

```
persistent_auto_chunk_size (hpx::chrono::steady_duration const &time_cs,
hpx::chrono::steady_duration const &rel_time,
std::uint64_t num_iters_for_timing = 0)
Construct an persistent_auto_chunk_size executor parameters object
```

Parameters

- *rel_time*: [in] The time duration to use as the minimum to decide how many loop iterations should be combined.
- *time_cs*: The execution time for each chunk.

hpx/execution/executors/polymorphic_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

```
        namespace execution
```

```
            template<typename R, typename ...Ts>
            class polymorphic_executor<R (Ts...)> : private
                hpx::parallel::execution::detail::polymorphic_executor_base
```

Public Types

```
template<typename T>
using future_type = hpx::future<R>
```

Public Functions

```
constexpr polymorphic_executor()
polymorphic_executor(polymorphic_executor const &other)
polymorphic_executor(polymorphic_executor &&other)
polymorphic_executor &operator=(polymorphic_executor const &other)
polymorphic_executor &operator=(polymorphic_executor &&other)

template<typename Exec, typename PE = typename std::decay<Exec>::type, typename Enable = typename
polymorphic_executor(Exec &&exec)

template<typename Exec, typename PE = typename std::decay<Exec>::type, typename Enable = typename
polymorphic_executor &operator=(Exec &&exec)

void reset()

template<typename F>
void post (F &&f, Ts... ts) const

template<typename F>
R sync_execute (F &&f, Ts... ts) const

template<typename F>
hpx::future<R> async_execute (F &&f, Ts... ts) const

template<typename F, typename Future>
hpx::future<R> then_execute (F &&f, Future &&predecessor, Ts&&... ts) const

template<typename F, typename Shape>
std::vector<R> bulk_sync_execute (F &&f, Shape const &s, Ts&&... ts) const
```

```

template<typename F, typename Shape>
std::vector<hpx::future<R>> bulk_async_execute (F &&f, Shape const &s, Ts&&...
                                         ts) const

template<typename F, typename Shape>
hpx::future<std::vector<R>> bulk_then_execute (F &&f, Shape const &s,
                                                 hpx::shared_future<void> const
                                                 &predecessor, Ts&&... ts) const

```

Private Types

```

template<>
using base_type = detail::polymorphic_executor_base

template<>
using vtable = detail::polymorphic_executor_vtable<R (Ts...) >

```

Private Functions

```

void assign (std::nullptr_t)

template<typename Exec>
void assign (Exec &&exec)

```

Private Static Functions

```

static constexpr vtable const *get_empty_vtable ()

template<typename T>
static constexpr vtable const *get_vtable ()

```

[hpx/execution/executors/rebind_executor.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

```
        namespace execution
```

Variables

```

constexpr create_rebound_policy_t create_rebound_policy = {}

struct create_rebound_policy_t

```

Public Functions

```
template<typename ExPolicy, typename Executor, typename Parameters>
constexpr decltype(auto) operator()(ExPolicy&&, Executor &&exec, Parameters
&&parameters) const

template<typename ExPolicy, typename Executor, typename Parameters>
struct rebind_executor
#include <rebind_executor.hpp> Rebind the type of executor used by an execution policy. The
execution category of Executor shall not be weaker than that of ExecutionPolicy.
```

Public Types

```
template<>
using type = typename policy_type::template rebind::type
The type of the rebound execution policy.
```

hpx/execution/executors/static_chunk_size.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
struct static_chunk_size
#include <static_chunk_size.hpp> Loop iterations are divided into pieces of size chunk_size and then
assigned to threads. If chunk_size is not specified, the iterations are evenly (if possible) divided
contiguously among the threads.
```

Note This executor parameters type is equivalent to OpenMP's STATIC scheduling directive.

Public Functions

```
constexpr static_chunk_size()
Construct a static_chunk_size executor parameters object
```

Note By default the number of loop iterations is determined from the number of available cores
and the overall number of loop iterations to schedule.

```
constexpr static_chunk_size(std::size_t chunk_size)
Construct a static_chunk_size executor parameters object
```

Parameters

- `chunk_size`: [in] The optional chunk size to use as the number of loop iterations to run
on a single thread.

hpx/execution/traits/is_execution_policy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Variables

```
template<typename T>
constexpr bool is_execution_policy_v = is_execution_policy<T>::value

template<typename T>
constexpr bool is_parallel_execution_policy_v = is_parallel_execution_policy<T>::value

template<typename T>
constexpr bool is_sequenced_execution_policy_v = is_sequenced_execution_policy<T>::value

template<typename T>
constexpr bool is_async_execution_policy_v = is_async_execution_policy<T>::value

template<typename T>
struct is_async_execution_policy : public hpx::detail::is_async_execution_policy<std::decay<T>::type>
    #include <is_execution_policy.hpp> Extension: Detect whether given execution policy makes algorithms
    asynchronous
```

1. The type `is_async_execution_policy` can be used to detect asynchronous execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.
2. If `T` is the type of a standard or implementation-defined execution policy, `is_async_execution_policy<T>` shall be publicly derived from `integral_constant<bool, true>`, otherwise from `integral_constant<bool, false>`.
3. The behavior of a program that adds specializations for `is_async_execution_policy` is undefined.

```
template<typename T>
struct is_execution_policy : public hpx::detail::is_execution_policy<std::decay<T>::type>
    #include <is_execution_policy.hpp>
    1. The type is_execution_policy can be used to detect execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.
    2. If T is the type of a standard or implementation-defined execution policy, is_execution_policy<T> shall be publicly derived from integral_constant<bool, true>, otherwise from integral_constant<bool, false>.
    3. The behavior of a program that adds specializations for is_execution_policy is undefined.
```

```
template<typename T>
struct is_parallel_execution_policy : public hpx::detail::is_parallel_execution_policy<std::decay<T>::type>
    #include <is_execution_policy.hpp> Extension: Detect whether given execution policy enables parallelization
```

1. The type `is_parallel_execution_policy` can be used to detect parallel execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.

2. If T is the type of a standard or implementation-defined execution policy, $\text{is_parallel_execution_policy}\langle T \rangle$ shall be publicly derived from $\text{integral_constant}\langle \text{bool}, \text{true} \rangle$, otherwise from $\text{integral_constant}\langle \text{bool}, \text{false} \rangle$.

3. The behavior of a program that adds specializations for $\text{is_parallel_execution_policy}$ is undefined.

```
template<typename T>
struct is_sequenced_execution_policy : public hpx::detail::is_sequenced_execution_policy<std::decay<T>::type>
    #include <is_execution_policy.hpp> Extension: Detect whether given execution policy does not enable parallelization
```

1. The type $\text{is_sequenced_execution_policy}$ can be used to detect non-parallel execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.

2. If T is the type of a standard or implementation-defined execution policy, $\text{is_sequenced_execution_policy}\langle T \rangle$ shall be publicly derived from $\text{integral_constant}\langle \text{bool}, \text{true} \rangle$, otherwise from $\text{integral_constant}\langle \text{bool}, \text{false} \rangle$.

3. The behavior of a program that adds specializations for $\text{is_sequenced_execution_policy}$ is undefined.

execution_base

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/execution_base/execution.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

Variables

```
hpx::sync_execute_t sync_execute
hpx::async_execute_t async_execute
hpx::then_execute_t then_execute
hpx::post_t post
hpx::bulk_sync_execute_t bulk_sync_execute
hpx::bulk_async_execute_t bulk_async_execute
hpx::bulk_then_execute_t bulk_then_execute

struct async_execute_t : public hpx::functional::detail::tagFallback<async_execute_t>
    #include <execution.hpp> Customization point for asynchronous execution agent creation.
```

This asynchronously creates a single function invocation $f()$ using the associated executor.

Note Executors have to implement only $\text{async_execute}()$. All other functions will be emulated by this or other customization points in terms of this single basic primitive. However, some executors will naturally specialize all operations for maximum efficiency.

Note This is valid for one way executors (calls `make_ready_future(exec.sync_execute(f, ts...))` if it exists) and for two way executors (calls `exec.async_execute(f, ts...)` if it exists).

Return `f(ts...)`'s result through a future

Parameters

- `exec`: [in] The executor object to use for scheduling of the function *f*.
- `f`: [in] The function which will be scheduled using the given executor.
- `ts`: [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tag_fallback_invoke(async_execute_t, Executor &&exec, F &&f,
                                         Ts&&... ts)

struct bulk_async_execute_t : public hpx::functional::detail::tag_fallback<bulk_async_execute_t>
#include <execution.hpp> Bulk form of asynchronous execution agent creation.
```

This asynchronously creates a group of function invocations `f(i)` whose ordering is given by the `execution_category` associated with the executor.

Note This is deliberately different from the `bulk_async_execute` customization points specified in P0443. The `bulk_async_execute` customization point defined here is more generic and is used as the workhorse for implementing the specified APIs.

Here *i* takes on all values in the index space implied by `shape`. All exceptions thrown by invocations of `f(i)` are reported in a manner consistent with parallel algorithm execution through the returned future.

Return The return type of `executor_type::bulk_async_execute` if defined by `executor_type`. Otherwise a vector of futures holding the returned values of each invocation of *f*.

Note This calls `exec.bulk_async_execute(f, shape, ts...)` if it exists; otherwise it executes `async_execute(f, shape, ts...)` as often as needed.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function *f*.
- `f`: [in] The function which will be scheduled using the given executor.
- `shape`: [in] The `shape` objects which defines the iteration boundaries for the arguments to be passed to *f*.
- `ts`: [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename Shape, typename ...Ts>
decltype(auto) friend tagFallback_invoke(bulk_async_execute_t, Executor &&exec, F
                                         &&f, Shape const &shape, Ts&&... ts)

template<typename Executor, typename F, typename Shape, typename ...Ts>
decltype(auto) friend tagFallback_invoke(bulk_async_execute_t, Executor &&exec, F
                                         &&f, Shape const &shape, Ts&&... ts)

struct bulk_sync_execute_t : public hpx::functional::detail::tag_fallback<bulk_sync_execute_t>
#include <execution.hpp> Bulk form of synchronous execution agent creation.
```

This synchronously creates a group of function invocations $f(i)$ whose ordering is given by the `execution_category` associated with the executor. The function synchronizes the execution of all scheduled functions with the caller.

Note This is deliberately different from the `bulk_sync_execute` customization points specified in P0443. The `bulk_sync_execute` customization point defined here is more generic and is used as the workhorse for implementing the specified APIs.

Here i takes on all values in the index space implied by `shape`. All exceptions thrown by invocations of $f(i)$ are reported in a manner consistent with parallel algorithm execution through the returned future.

Return The return type of `executor_type::bulk_sync_execute` if defined by `executor_type`. Otherwise a vector holding the returned values of each invocation of f except when f returns void, which case void is returned.

Note This calls `exec.bulk_sync_execute(f, shape, ts...)` if it exists; otherwise it executes `sync_execute(f, shape, ts...)` as often as needed.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function f .
- `f`: [in] The function which will be scheduled using the given executor.
- `shape`: [in] The shape objects which defines the iteration boundaries for the arguments to be passed to f .
- `ts`: [in] Additional arguments to use to invoke f .

Private Functions

```
template<typename Executor, typename F, typename Shape, typename ...Ts>
decltype(auto) friend tagFallback_invoke(bulk_sync_execute_t, Executor &&exec, F
                                         &&f, Shape const &shape, Ts&&... ts)

template<typename Executor, typename F, typename Shape, typename ...Ts>
decltype(auto) friend tagFallback_invoke(bulk_sync_execute_t, Executor &&exec, F
                                         &&f, Shape const &shape, Ts&&... ts)

struct bulk_then_execute_t : public hpx::functional::detail::tag_fallback<bulk_then_execute_t>
#include <execution.hpp> Bulk form of execution agent creation depending on a given future.
```

This creates a group of function invocations $f(i)$ whose ordering is given by the execution_category associated with the executor.

Note This is deliberately different from the then_sync_execute customization points specified in P0443. The bulk_then_execute customization point defined here is more generic and is used as the workhorse for implementing the specified APIs.

Here i takes on all values in the index space implied by shape. All exceptions thrown by invocations of $f(i)$ are reported in a manner consistent with parallel algorithm execution through the returned future.

Return The return type of `executor_type::bulk_then_execute` if defined by `executor_type`. Otherwise a vector holding the returned values of each invocation of f .

Note This calls `exec.bulk_then_execute(f, shape, pred, ts...)` if it exists; otherwise it executes `sync_execute(f, shape, pred.share(), ts...)` (if this executor is also an OneWayExecutor), or `async_execute(f, shape, pred.share(), ts...)` (if this executor is also a TwoWayExecutor) - as often as needed.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function f .
- `f`: [in] The function which will be scheduled using the given executor.
- `shape`: [in] The shape objects which defines the iteration boundaries for the arguments to be passed to f .
- `predecessor`: [in] The future object the execution of the given function depends on.
- `ts`: [in] Additional arguments to use to invoke f .

Private Functions

```
template<typename Executor, typename F, typename Shape, typename Future, typename ...Ts>
decltype(auto) friend tagFallback_invoke(bulk_then_execute_t, Executor &&exec, F
                                         &&f, Shape const &shape, Future &&pre-
                                         decessor, Ts&&... ts)

template<typename Executor, typename F, typename Shape, typename Future, typename ...Ts>
decltype(auto) friend tagFallback_invoke(bulk_then_execute_t, Executor &&exec, F
                                         &&f, Shape const &shape, Future &&pre-
                                         decessor, Ts&&... ts)
```

```
struct post_t : public hpx::functional::detail::tagFallback<post_t>
#include <execution.hpp> Customization point for asynchronous fire & forget execution agent creation.
```

This asynchronously (fire & forget) creates a single function invocation $f()$ using the associated executor.

Note This is valid for two way executors (calls `exec.post(f, ts...)`, if available, otherwise it calls `exec.async_execute(f, ts...)` while discarding the returned future), and for non-blocking two way executors (calls `exec.post(f, ts...)` if it exists).

Parameters

- `exec`: [in] The executor object to use for scheduling of the function f .
- `f`: [in] The function which will be scheduled using the given executor.
- `ts`: [in] Additional arguments to use to invoke f .

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tagFallback_invoke(post_t, Executor &&exec, F &&f, Ts&&... ts)

struct sync_execute_t : public hpx::functional::detail::tag_fallback<sync_execute_t>
#include <execution.hpp> Customization point for synchronous execution agent creation.
```

This synchronously creates a single function invocation *f()* using the associated executor. The execution of the supplied function synchronizes with the caller

Return *f(ts...)*'s result

Note This is valid for one way executors only, it will call *exec.sync_execute(f, ts...)* if it exists.

Parameters

- *exec*: [in] The executor object to use for scheduling of the function *f*.
- *f*: [in] The function which will be scheduled using the given executor.
- *ts*: [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tagFallback_invoke(sync_execute_t, Executor &&exec, F &&f,
                                         Ts&&... ts)

struct then_execute_t : public hpx::functional::detail::tag_fallback<then_execute_t>
#include <execution.hpp> Customization point for execution agent creation depending on a given future.
```

This creates a single function invocation *f()* using the associated executor after the given future object has become ready.

Return *f(ts...)*'s result through a future

Note This is valid for two way executors (calls *exec.then_execute(f, predecessor, ts...)* if it exists) and for one way executors (calls *predecessor.then(bind(f, ts...))*).

Parameters

- *exec*: [in] The executor object to use for scheduling of the function *f*.
- *f*: [in] The function which will be scheduled using the given executor.
- *predecessor*: [in] The future object the execution of the given function depends on.
- *ts*: [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename Future, typename ...Ts>
decltype(auto) friend tagFallback_invoke(then_execute_t, Executor &&exec, F &&f,
                                         Future &&predecessor, Ts&&... ts)
```

namespace execution

```
struct parallel_execution_tag
    #include <execution.hpp> Function invocations executed by a group of parallel execution agents
execute in unordered fashion. Any such invocations executing in the same thread are indeterminately
sequenced with respect to each other.
```

Note `parallel_execution_tag` is weaker than `sequenced_execution_tag`.

```
struct sequenced_execution_tag
    #include <execution.hpp> Function invocations executed by a group of sequential execution agents
execute in sequential order.
```

```
struct unsequenced_execution_tag
    #include <execution.hpp> Function invocations executed by a group of vector execution agents are
permitted to execute in unordered fashion when executed in different threads, and un-sequenced with
respect to one another when executed in the same thread.
```

Note `unsequenced_execution_tag` is weaker than `parallel_execution_tag`.

hpx/execution_base/receiver.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace execution

namespace experimental

Functions

```
template<typename R, typename ...As>
void set_value(R &&r, As&&... as)
    set_value is a customization point object. The expression
    hpx::execution::set_value(r, as...) is equivalent to:
    • r.set_value(as...), if that expression is valid. If the function selected does not send
        the value(s) as... to the Receiver r's value channel, the program is ill-formed (no diagnostic
        required).
    • Otherwise, `set_value(r, as...), if that expression is valid, with overload resolution performed
        in a context that include the declaration void set_value();
    • Otherwise, the expression is ill-formed.
The customization is implemented in terms of hpx::functional::tag_invoke.
```

```
template<typename R>
void set_stopped(R &&r)
    set_stopped is a customization point object.           The expression
    hpx::execution::set_stopped(r) is equivalent to:
        • r.set_stopped(), if that expression is valid. If the function selected does not signal the
          Receiver r's done channel, the program is ill-formed (no diagnostic required).
        • Otherwise, `set_stopped(r), if that expression is valid, with overload resolution performed in a
          context that include the declaration void set_stopped();
        • Otherwise, the expression is ill-formed.
```

The customization is implemented in terms of `hpx::functional::tag_invoke`.

```
template<typename R, typename E>
void set_error(R &&r, E &&e)
    set_error is a customization point object.           The expression
    hpx::execution::set_error(r, e) is equivalent to:
        • r.set_stopped(e), if that expression is valid. If the function selected does not send the
          error e the Receiver r's error channel, the program is ill-formed (no diagnostic required).
        • Otherwise, `set_error(r, e), if that expression is valid, with overload resolution performed in a
          context that include the declaration void set_error();
        • Otherwise, the expression is ill-formed.
```

The customization is implemented in terms of `hpx::functional::tag_invoke`.

Variables

```
hpx::execution::experimental::set_value_t set_value
hpx::execution::experimental::set_error_t set_error
hpx::execution::experimental::set_stopped_t set_stopped
template<typename T, typename E = std::exception_ptr>
constexpr bool is_receiver_v = is_receiver<T, E>::value
template<typename T, typename ...As>
constexpr bool is_receiver_of_v = is_receiver_of<T, As...>::value
template<typename T, typename ...As>
constexpr bool is_nothrow_receiver_of_v = is_nothrow_receiver_of<T, As...>::value
template<typename T, typename E>
struct is_receiver
    #include <receiver.hpp> Receiving values from asynchronous computations is handled by the
    Receiver concept. A Receiver needs to be able to receive an error or be marked as being
    canceled. As such, the Receiver concept is defined by having the following two customization
    points defined, which form the completion-signal operations:
```

- `hpx::execution::experimental::set_stopped`
- `hpx::execution::experimental::set_error`

Those two functions denote the completion-signal operations. The Receiver contract is as follows:

- None of a Receiver's completion-signal operation shall be invoked before `hpx::execution::experimental::start` has been called on the operation state object that was returned by connecting a Receiver to a sender `hpx::execution::experimental::connect`.
- Once `hpx::execution::start` has been called on the operation state object, exactly one of the Receiver's completion-signal operation shall complete without an exception before the Receiver is destroyed

Once one of the Receiver's completion-signal operation has been completed without throwing an exception, the Receiver contract has been satisfied. In other words: The asynchronous operation has been completed.

See [hpx::execution::experimental::is_receiver_of](#)

```
template<typename T, typename ...As>
struct is_receiver_of
#include <receiver.hpp> The receiver_of concept is a refinement of the Receiver concept
by requiring one additional completion-signal operation:
• hpx::execution::set_value
This completion-signal operation adds the following to the Receiver's contract:
• If hpx::execution::set_value exits with an exception, it is still valid to call
  hpx::execution::set_error or hpx::execution::set_stopped
```

See [hpx::execution::traits::is_receiver](#)

[hpx/execution_base/traits/is_executor_parameters.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
template<typename Executor>
struct extract_executor_parameters<Executor, std::void_t<typename Executor::executor_parameters_type>>
```

Public Types

```
template<>
using type = typename Executor::executor_parameters_type
namespace hpx

namespace parallel

namespace execution
```

Typedefs

```
template<typename Executor>
using extract_executor_parameters_t = typename extract_executor_parameters<Executor>::type
template<typename T>
using is_executor_parameters_t = typename is_executor_parameters<T>::type
```

Variables

```
template<typename Parameters>
constexpr bool extract_has_variable_chunk_size_v = extract_has_variable_chunk_size<Parameters>()

template<typename Parameters>
constexpr bool extract_invokes_testing_function_v = extract_invokes_testing_function<Parameters>()

template<typename T>
constexpr bool is_executor_parameters_v = is_executor_parameters<T>::value

template<typename Executor, typename Enable = void>
struct extract_executor_parameters
```

Public Types

```
template<>
using type = sequential_executor_parameters

template<typename Executor>
struct extract_executor_parameters<Executor, std::void_t<typename Executor::executor_parameters>>
```

Public Types

```
template<>
using type = typename Executor::executor_parameters_type

namespace traits
```

Typedefs

```
template<typename T>
using is_executor_parameters_t = typename is_executor_parameters<T>::type
```

Variables

```
template<typename T>
constexpr bool is_executor_parameters_v = is_executor_parameters<T>::value
```

executors

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/executors/annotating_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace execution
```

```
        namespace experimental
```

Functions

```
template<typename Executor>
constexpr auto tagFallbackInvoke(with_annotation_t, Executor &&exec, char const *annotation)
```

```
template<typename Executor>
auto tagFallbackInvoke(with_annotation_t, Executor &&exec, std::string annotation)
```

```
template<typename BaseExecutor>
struct annotating_executor
#include <annotating_executor.hpp> A annotating_executor wraps any other executor and adds
the capability to add annotations to the launched threads.
```

Public Functions

```
template<typename Executor, typename Enable = std::enable_if_t<hpx::traits::is_executor_any_v<Executor>>
constexpr annotating_executor(Executor &&exec, char const *annotation =
nullptr)
```

```
template<typename Executor, typename Enable = std::enable_if_t<hpx::traits::is_executor_any_v<Executor>>
annotating_executor(Executor &&exec, std::string annotation)
```

hpx/executors/current_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

```
        namespace execution
```

TypeDefs

```
using current_executor = parallel::execution::thread_pool_executor  
namespace this_thread
```

Functions

`parallel::execution::current_executor get_executor(error_code &ec = throws)`
Returns a reference to the executor which was used to create the current thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

```
namespace threads
```

Functions

`parallel::execution::current_executor get_executor(thread_id_type const &id, error_code &ec = throws)`
Returns a reference to the executor which was used to create the given thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

[hpx/executors/exception_list.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

[hpx/executors/datapar/execution_policy.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/executors/execution_policy_annotation.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace execution
```

```
        namespace experimental
```

Functions

```
template<typename ExPolicy>
constexpr decltype(auto) tag_invoke(hpx::execution::experimental::with_annotation_t,
                                    ExPolicy &&policy, char const *annotation)
```

```
template<typename ExPolicy>
decltype(auto) tag_invoke(hpx::execution::experimental::with_annotation_t,           ExPolicy
                           &&policy, std::string annotation)
```

```
template<typename ExPolicy>
constexpr decltype(auto) tag_invoke(hpx::execution::experimental::get_annotation_t,
                                    ExPolicy &&policy)
```

hpx/executors/execution_policy_parameters.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

```
        namespace execution
```

Functions

```
template<typename ExPolicy>
constexpr decltype(auto) tag_invoke(with_processing_units_count_t, ExPolicy &&pol-
                                icy, std::size_t num_cores)
```

```
template<typename ExPolicy, typename Params>
constexpr decltype(auto) tag_invoke(with_processing_units_count_t, ExPolicy &&pol-
                                icy, Params &&params)
```

```
template<typename ParametersProperty, typename ExPolicy, typename Params>
constexpr decltype(auto) tag_fallback_invoke(ParametersProperty,           ExPolicy
                                            &&policy, Params &&params)
```

hpx/executors/fork_join_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace execution

namespace experimental

Functions

```
std::ostream &operator<< (std::ostream &os, fork_join_executor::loop_schedule const &schedule)
```

class fork_join_executor

#include <fork_join_executor.hpp> An executor with fork-join (blocking) semantics.

The *fork_join_executor* creates on construction a set of worker threads that are kept alive for the duration of the executor. Copying the executor has reference semantics, i.e. copies of a *fork_join_executor* hold a reference to the worker threads of the original instance. Scheduling work through the executor concurrently from different threads is undefined behaviour.

The executor keeps a set of worker threads alive for the lifetime of the executor, meaning other work will not be executed while the executor is busy or waiting for work. The executor has a customizable delay after which it will yield to other work. Since starting and resuming the worker threads is a slow operation the executor should be reused whenever possible for multiple adjacent parallel algorithms or invocations of bulk_(a)sync_execute.

Public Types

enum loop_schedule

Type of loop schedule for use with the *fork_join_executor*. `loop_schedule::static_` implies no work-stealing; `loop_schedule::dynamic` allows stealing when a worker has finished its local work.

Values:

static_

dynamic

Public Functions

```
fork_join_executor (threads::thread_priority priority = threads::thread_priority::high,
                    threads::thread_stacksize stacksize = threads::thread_stacksize::small_,
                    loop_schedule schedule = loop_schedule::static_,
                    std::chrono::nanoseconds yield_delay = std::chrono::milliseconds(1))
```

Construct a *fork_join_executor*.

Parameters

- `priority`: The priority of the worker threads.

- `stacksize`: The stacksize of the worker threads. Must not be `nostack`.
- `schedule`: The loop schedule of the parallel regions.
- `yield_delay`: The time after which the executor yields to other work if it hasn't received any new work for bulk execution.

`hpx/executors/parallel_executor.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

Typedefs

```
using parallel_executor = parallel_policy_executor<hpx::launch>

template<typename Policy>
struct parallel_policy_executor
    #include <parallel_executor.hpp> A parallel_executor creates groups of parallel execution agents
    which execute in threads implicitly created by the executor. This executor prefers continuing with the
    creating thread first before executing newly created threads.
```

This executor conforms to the concepts of a `TwoWayExecutor`, and a `BulkTwoWayExecutor`

Public Types

```
template<>
using execution_category = parallel_execution_tag
    Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = static_chunk_size
    Associate the static_chunk_size executor parameters type as a default with this executor.
```

Public Functions

```
constexpr parallel_policy_executor(threads::thread_priority priority,
                                    threads::thread_stacksize stacksize =
                                    threads::thread_stacksize::default_,
                                    threads::thread_schedule_hint sched-
                                    ulehint = {}, Policy l = parallel::execution::detail::get_default_policy<Policy>::call(),
                                    std::size_t hierarchical_threshold = hierar-
                                    chical_threshold_default_)
```

Create a new parallel executor.

```
constexpr parallel_policy_executor(threads::thread_stacksize stacksize,
                                    threads::thread_schedule_hint sched-
                                    ulehint = {}, Policy l = parallel::execution::detail::get_default_policy<Policy>::call())
```

```
constexpr parallel_policy_executor(threads::thread_schedule_hint  
schedulehint, Policy l = parallel::execution::detail::get_default_policy<Policy>::call())  
  
constexpr parallel_policy_executor(Policy l = parallel::execution::detail::get_default_policy<Policy>::call())  
  
constexpr parallel_policy_executor(threads::thread_pool_base *pool,  
threads::thread_priority priority = threads::thread_priority::default_,  
threads::thread_stacksize stacksize = threads::thread_stacksize::default_,  
threads::thread_schedule_hint schedulehint = {}, Policy l = parallel::execution::detail::get_default_policy<Policy>::call(),  
std::size_t hierarchical_threshold = hierarchical_threshold_default_)
```

Friends

```
friend constexpr parallel_policy_executor tag_invoke(hpx::execution::experimental::with_hint_t,  
parallel_policy_executor  
const &exec,  
hpx::threads::thread_schedule_hint  
hint)  
  
friend constexpr hpx::threads::thread_schedule_hint tag_invoke(hpx::execution::experimental::get_hint_t,  
parallel_policy_executor  
const &exec)  
  
friend constexpr parallel_policy_executor tag_invoke(hpx::execution::experimental::with_priority_t,  
parallel_policy_executor  
const &exec,  
hpx::threads::thread_priority  
priority)  
  
friend constexpr hpx::threads::thread_priority tag_invoke(hpx::execution::experimental::get_priority_t,  
parallel_policy_executor  
const &exec)  
  
friend constexpr parallel_policy_executor tag_dispatch(hpx::execution::experimental::with_stacksize_t,  
parallel_policy_executor  
const &exec,  
hpx::threads::thread_stacksize  
stacksize)  
  
friend constexpr hpx::threads::thread_stacksize tag_dispatch(hpx::execution::experimental::get_stacksize_t,  
parallel_policy_executor  
const &exec)  
  
friend constexpr parallel_policy_executor tag_invoke(hpx::execution::experimental::with_annotation_t,  
parallel_policy_executor  
const &exec, char const  
*annotation)
```

```

parallel_policy_executor tag_invoke (hpx::execution::experimental::with_annotation_t, parallel_policy_executor const &exec, std::string annotation)
friend constexpr char const *tag_invoke (hpx::execution::experimental::get_annotation_t, parallel_policy_executor const &exec)
friend constexpr parallel_policy_executor tag_invoke (hpx::parallel::execution::with_processing_units_count_t, parallel_policy_executor const &exec, std::size_t num_cores)
template<typename Parameters>
friend constexpr std::size_t tag_invoke (hpx::parallel::execution::processing_units_count_t, Parameters&&, parallel_policy_executor const &exec)

```

hpx/executors/parallel_executor_aggregated.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
template<>
struct parallel_policy_executor_aggregated<hpx::launch>
```

Public Types

```
template<>
using execution_category = hpx::execution::parallel_execution_tag
Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = hpx::execution::static_chunk_size
Associate the static_chunk_size executor parameters type as a default with this executor.
```

Public Functions

```
constexpr parallel_policy_executor_aggregated(hpx::launch l = hpx::launch::async_policy{},
std::size_t spread = 4, std::size_t tasks = std::size_t(-1))
Create a new parallel executor.

template<typename F, typename S, typename ...Ts>
auto bulk_async_execute(F &&f, S const &shape, Ts&&... ts) const
namespace hpx

namespace parallel

namespace execution
```

TypeDefs

```
using parallel_executor_aggregated = parallel_policy_executor_aggregated<hpx::launch::async_policy>

template<typename Policy = hpx::launch::async_policy>
struct parallel_policy_executor_aggregated
    #include <parallel_executor_aggregated.hpp> A parallel_executor_aggregated creates groups
    of parallel execution agents that execute in threads implicitly created by the executor. This execu-
    tor prefers continuing with the creating thread first before executing newly created threads.

This executor conforms to the concepts of a TwoWayExecutor, and a BulkTwoWayExecutor
```

Public Types

```
template<>
using execution_category = hpx::execution::parallel_execution_tag
    Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = hpx::execution::static_chunk_size
    Associate the static_chunk_size executor parameters type as a default with this executor.
```

Public Functions

```
constexpr parallel_policy_executor_aggregated(std::size_t spread = 4, std::size_t tasks = std::size_t(-1))
```

Create a new parallel executor.

```
template<typename F, typename S, typename ...Ts>
std::vector<hpx::future<void>> bulk_async_execute(F &&f, S const &shape,
    Ts&&... ts) const
```

```
template<>
struct parallel_policy_executor_aggregated<hpx::launch>
```

Public Types

```
template<>
using execution_category = hpx::execution::parallel_execution_tag
    Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = hpx::execution::static_chunk_size
    Associate the static_chunk_size executor parameters type as a default with this executor.
```

Public Functions

```
constexpr parallel_policy_executor_aggregated(hpx::launch l = hpx::launch::async_policy{},  

std::size_t spread = 4, std::size_t tasks = std::size_t(-1))
```

Create a new parallel executor.

```
template<typename F, typename S, typename ...Ts>  
auto bulk_async_execute(F &&f, S const &shape, Ts&&... ts) const
```

`hpx/executors/restricted_thread_pool_executor.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel  
  
namespace execution  
  
class restricted_thread_pool_executor
```

Public Types

```
typedef hpx::execution::parallel_execution_tag execution_category
```

Associate the parallel_execution_tag executor tag type as a default with this executor.

```
typedef hpx::execution::static_chunk_size executor_parameters_type
```

Associate the static_chunk_size executor parameters type as a default with this executor.

Public Functions

```
restricted_thread_pool_executor(std::size_t first_thread = 0, std::size_t  
num_threads = 1, threads::thread_priority  
priority = threads::thread_priority::default_,  
threads::thread_stacksize stacksize =  
threads::thread_stacksize::default_,  
threads::thread_schedule_hint schedule-  
hint = {}, std::size_t hierarchical_threshold =  
hierarchical_threshold_default_)
```

Create a new parallel executor.

```
restricted_thread_pool_executor(restricted_thread_pool_executor &other) const
```

Private Members

```
threads::thread_pool_base *pool_ = nullptr
threads::thread_priority priority_ = threads::thread_priority::default_
threads::thread_stacksize stacksize_ = threads::thread_stacksize::default_
threads::thread_schedule_hint schedulehint_ = {}
std::size_t hierarchical_threshold_ = hierarchical_threshold_default_
std::size_t first_thread_
std::size_t num_threads_
std::atomic<std::size_t> os_thread_
```

Private Static Attributes

```
constexpr std::size_t hierarchical_threshold_default_ = 6
```

hpx/executors/scheduler_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace execution
```

```
namespace experimental
```

Functions

```
template<typename BaseScheduler>
scheduler_executor(BaseScheduler &&sched)

template<typename BaseScheduler>
struct scheduler_executor
```

Public Functions

```
constexpr scheduler_executor()

template<typename Scheduler, typename Enable = std::enable_if_t<hpx::execution::experimental::is_scheduler_v<Scheduler>>
constexpr scheduler_executor(Scheduler &&sched)

constexpr scheduler_executor(scheduler_executor&&)

constexpr scheduler_executor &operator=(scheduler_executor&&)

constexpr scheduler_executor(scheduler_executor const&)

constexpr scheduler_executor &operator=(scheduler_executor const&)
```

hpx/executors/sequenced_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace execution
```

```
        struct sequenced_executor
```

```
            #include <sequenced_executor.hpp> A sequential_executor creates groups of sequential execution agents which execute in the calling thread. The sequential order is given by the lexicographical order of indices in the index space.
```

hpx/executors/service_executors.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/executors/std_execution_policy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/executors/thread_pool_scheduler.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace execution
```

```
        namespace experimental
```

```
            struct thread_pool_scheduler
```

Public Functions

```
            constexpr thread_pool_scheduler()
```

```
            thread_pool_scheduler(hpx::threads::thread_pool_base *pool)
```

filesystem

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/modules/filesystem.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

This file provides a compatibility layer using Boost.Filesystem for the C++17 filesystem library. It is *not* intended to be a complete compatibility layer. It only contains functions required by the HPX codebase. It also provides some functions only available in Boost.Filesystem when using C++17 filesystem.

```
namespace hpx
```

```
namespace filesystem
```

Functions

```
path initial_path()  
  
std::string basename(path const &p)  
  
path canonical(path const &p, path const &base)  
  
path canonical(path const &p, path const &base, std::error_code &ec)
```

functional

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/functional/invoke.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_INVOKE_R(R, F, ...)
```

```
namespace hpx
```

```
namespace util
```

Functions

```
template<typename F, typename ...Ts>
constexpr util::invoke_result<F, Ts...>::type invoke (F &&f, Ts&&... vs)
    Invokes the given callable object f with the content of the argument pack vs
```

Return The result of the callable object when it's called with the given argument types.

Note This function is similar to `std::invoke` (C++17)

Parameters

- `f`: Requires to be a callable object. If `f` is a member function pointer, the first argument in the pack will be treated as the callee (this object).
- `vs`: An arbitrary pack of arguments

Exceptions

- `std::exception`-like objects thrown by call to object `f` with the argument types `vs`.

```
template<typename R, typename F, typename ...Ts>
constexpr R invoke_r (F &&f, Ts&&... vs)
```

Template Parameters

- `R`: The result type of the function when it's called with the content of the given argument types `vs`.

```
namespace functional
```

```
struct invoke
```

Public Functions

```
template<typename F, typename ...Ts>
constexpr util::invoke_result<F, Ts...>::type operator() (F &&f, Ts&&... vs) const
```

```
template<typename R>
struct invoke_r
```

Public Functions

```
template<typename F, typename ...Ts>
constexpr R operator() (F &&f, Ts&&... vs) const
```

hpx/functional/invoke_fused.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

Functions

```
template<typename F, typename Tuple>
constexpr detail::invoke_fused_result<F, Tuple>::type invoke_fused(F &&f, Tuple &&t)
    Invokes the given callable object f with the content of the sequenced type t (tuples, pairs)
```

Return The result of the callable object when it's called with the content of the given sequenced type.

Note This function is similar to `std::apply` (C++17)

Parameters

- `f`: Must be a callable object. If `f` is a member function pointer, the first argument in the sequenced type will be treated as the callee (this object).
- `t`: A type whose contents are accessible through a call to `hpx::get`.

Exceptions

- `std::exception`: like objects thrown by call to object `f` with the arguments contained in the sequenceable type `t`.

```
template<typename R, typename F, typename Tuple>
constexpr R invoke_fused_r(F &&f, Tuple &&t)
```

Template Parameters

- `R`: The result type of the function when it's called with the content of the given sequenced type.

futures

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/futures/future_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace lcos

TypeDefs

```
typedef hpx::shared_future<R> instead
```

io_service

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/io_service/io_service_pool.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

```
class io_service_pool
    #include <io_service_pool.hpp> A pool of io_service objects.
```

Public Functions

```
HPX_NON_COPYABLE (io_service_pool)
```

```
io_service_pool (std::size_t pool_size = 2, threads::policies::callback_notifier const
                    &notifier = threads::policies::callback_notifier(), char const
                    *pool_name = "", char const *name_postfix = "")
```

Construct the io_service pool.

Parameters

- *pool_size*: [in] The number of threads to run to serve incoming requests
- *start_thread*: [in]

```
io_service_pool (threads::policies::callback_notifier const &notifier, char const
                    *pool_name = "", char const *name_postfix = "")
```

Construct the io_service pool.

Parameters

- *start_thread*: [in]

```
~io_service_pool ()
```

```
bool run (bool join_threads = true, barrier *startup = nullptr)
```

Run all io_service objects in the pool. If *join_threads* is true this will also wait for all threads to complete

```
bool run (std::size_t num_threads, bool join_threads = true, barrier *startup = nullptr)
```

Run all io_service objects in the pool. If *join_threads* is true this will also wait for all threads to complete

```
void stop ()
```

Stop all io_service objects in the pool.

```
void join ()
```

Join all io_service threads in the pool.

```
void clear ()
```

Clear all internal data structures.

```
void wait ()
```

Wait for all work to be done.

```
bool stopped ()
```

```
asio::io_context &get_io_service (int index = -1)
    Get an io_service to use.

std::thread &get_os_thread_handle (std::size_t thread_num)
    access underlying thread handle

std::size_t size () const
    Get number of threads associated with this I/O service.

void thread_run (std::size_t index, barrier *startup = nullptr)
    Activate the thread index for this thread pool.

char const *get_name () const
    Return name of this pool.

void init (std::size_t pool_size)
```

Protected Functions

```
bool run_locked (std::size_t num_threads, bool join_threads, barrier *startup)
void stop_locked ()
void join_locked ()
void clear_locked ()
void wait_locked ()
```

Private Types

```
using io_service_ptr = std::unique_ptr<asio::io_context>
using work_type = asio::io_context::work
```

Private Functions

```
work_type initialize_work (asio::io_context &io_service)
```

Private Members

```
std::mutex mtx_
std::vector<io_service_ptr> io_services_
    The pool of io_services.

std::vector<std::thread> threads_

std::vector<work_type> work_
    The work that keeps the io_services running.

std::size_t next_io_service_
    The next io_service to use for a connection.

bool stopped_
    set to true if stopped
```

```
std::size_t pool_size_
    initial number of OS threads to execute in this pool

threads::policies::callback_notifier const &notifier_
    call this for each thread start/stop

char const *pool_name_
char const *pool_name_postfix_
bool waiting_
    Set to true if waiting for work to finish.

std::unique_ptr<barrier> wait_barrier_
std::unique_ptr<barrier> continue_barrier_
```

lcos_local

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/lcos_local/and_gate.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

```
namespace lcos

namespace local

struct and_gate : public hpx::lcos::local::base_and_gate<hpx::no_mutex>
```

Public Functions

```
and_gate (std::size_t count = 0)

and_gate (and_gate &&rhs)

and_gate &operator= (and_gate &&rhs)

template<typename Lock>
hpx::future<void> get_future (Lock &l, std::size_t count = std::size_t(-1), std::size_t
    *generation_value = nullptr, error_code &ec =
    hpx::throws)

template<typename Lock>
hpx::shared_future<void> get_shared_future (Lock &l, std::size_t count = std::size_t(-
    1), std::size_t *generation_value =
    nullptr, error_code &ec = hpx::throws)

template<typename Lock>
bool set (std::size_t which, Lock l, error_code &ec = throws)

template<typename Lock>
```

```
void synchronize (std::size_t generation_value, Lock &l, char const *function_name =  
    "and_gate::synchronize", error_code &ec = throws)
```

Private Types

```
typedef base_and_gate<hpx::no_mutex> base_type  
  
template<typename Mutex = hpx::spinlock>  
struct base_and_gate
```

Public Functions

```
base_and_gate (std::size_t count = 0)
```

This constructor initializes the *base_and_gate* object from the the number of participants to synchronize the control flow with.

```
base_and_gate (base_and_gate &&rhs)
```

```
base_and_gate &operator= (base_and_gate &&rhs)
```

```
hpx::future<void> get_future (std::size_t count = std::size_t(-1), std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)
```

```
hpx::shared_future<void> get_shared_future (std::size_t count = std::size_t(-1),  
                                              std::size_t *generation_value = nullptr,  
                                              error_code &ec = hpx::throws)
```

```
bool set (std::size_t which, error_code &ec = throws)
```

```
void synchronize (std::size_t generation_value, char const *function_name =  
    "base_and_gate<>::synchronize", error_code &ec = throws)
```

Wait for the generational counter to reach the requested stage.

```
std::size_t next_generation ()
```

```
std::size_t generation () const
```

Protected Types

```
typedef Mutex mutex_type
```

Protected Functions

```
bool trigger_conditions (error_code &ec = throws)
```

```
template<typename OuterLock>  
hpx::future<void> get_future (OuterLock &outer_lock, std::size_t count = std::size_t(-1),  
                           std::size_t *generation_value = nullptr, error_code &ec =  
                           hpx::throws)
```

get a future allowing to wait for the gate to fire

```
template<typename OuterLock>
```

```

hpx::shared_future<void> get_shared_future(OuterLock &outer_lock, std::size_t
    count = std::size_t(-1), std::size_t *gen-
    eration_value = nullptr, error_code &ec
    = hpx::throws)
    get a shared future allowing to wait for the gate to fire

template<typename OuterLock>
bool set(std::size_t which, OuterLock outer_lock, error_code &ec = throws)
    Set the data which has to go into the segment which.

bool test_condition(std::size_t generation_value)

template<typename Lock>
void synchronize(std::size_t generation_value, Lock &l, char const *function_name =
    "base_and_gate<>::synchronize", error_code &ec = throws)

template<typename OuterLock, typename Lock>
void init_locked(OuterLock &outer_lock, Lock &l, std::size_t count, error_code &ec =
    throws)
```

Private Types

```
typedef std::list<conditional_trigger*> condition_list_type
```

Private Members

```

mutex_type mtx_
hpx::detail::dynamic_bitset received_segments_
hpx::promise<void> promise_
std::size_t generation_
condition_list_type conditions_
struct manage_condition
```

Public Functions

```

template<>
manage_condition(base_and_gate &gate, conditional_trigger &cond)

template<>
~manage_condition()

template<typename Condition>
hpx::future<void> get_future(Condition &&func, error_code &ec = hpx::throws)
```

Public Members

```
template<>
base_and_gate &this_
template<>
condition_list_type::iterator it_
```

hpx/lcos_local/conditional_trigger.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace lcos
```

```
namespace local
```

```
struct conditional_trigger
```

Public Functions

```
conditional_trigger()
```

```
conditional_trigger(conditional_trigger &&rhs)
```

```
conditional_trigger &operator=(conditional_trigger &&rhs)
```

```
template<typename Condition>
hpx::future<void> get_future(Condition &&func, error_code &ec = hpx::throws)
    get a future allowing to wait for the trigger to fire
```

```
void reset()
```

```
bool set(error_code &ec = throws)
```

Trigger this object.

Private Members

```
hpx::promise<void> promise_
```

```
hpx::function<bool ()> cond_
```

hpx/lcos_local/trigger.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace lcos
```

```
        namespace local
```

```
template<typename Mutex = hpx::spinlock>
struct base_trigger
```

Public Functions

```
base_trigger()
```

```
base_trigger(base_trigger &&rhs)
```

```
base_trigger &operator=(base_trigger &&rhs)
```

```
hpx::future<void> get_future (std::size_t *generation_value = nullptr, error_code &ec =
                               hpx::throws)
```

get a future allowing to wait for the trigger to fire

```
bool set (error_code &ec = throws)
```

Trigger this object.

```
void synchronize (std::size_t generation_value, char const *function_name = "trigger::synchronize", error_code &ec = throws)
```

Wait for the generational counter to reach the requested stage.

```
std::size_t next_generation()
```

```
std::size_t generation () const
```

Protected Types

```
typedef Mutex mutex_type
```

Protected Functions

```
bool trigger_conditions (error_code &ec = throws)
```

```
template<typename Lock>
```

```
void synchronize (std::size_t generation_value, Lock &l, char const *function_name =
                  "trigger::synchronize", error_code &ec = throws)
```

Private Types

```
typedef std::list<conditional_trigger*> condition_list_type
```

Private Functions

```
bool test_condition (std::size_t generation_value)
```

Private Members

```
mutex_type mtx_
hpx::promise<void> promise_
std::size_t generation_
condition_list_type conditions_

struct manage_condition
```

Public Functions

```
template<>
manage_condition (base_trigger &gate, conditional_trigger &cond)

template<>
~manage_condition ()

template<typename Condition>
hpx::future<void> get_future (Condition &&func, error_code &ec = hpx::throws)
```

Public Members

```
template<>
base_trigger &this_

template<>
condition_list_type::iterator it_

struct trigger : public hpx::lcos::local::base_trigger<hpx::no_mutex>
```

Public Functions

```
trigger()

trigger (trigger &&rhs)

trigger &operator= (trigger &&rhs)

template<typename Lock>
void synchronize (std::size_t generation_value, Lock &l, char const *function_name =
"trigger::synchronize", error_code &ec = throws)
```

Private Types

```
typedef base_trigger<hpx::no_mutex> base_type
```

pack_traversal

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/pack_traversal/pack_traversal.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

Functions

```
template<typename Mapper, typename... T><unspecified> hpx::util::map_pack(Mapper &
```

Maps the pack with the given mapper.

This function tries to visit all plain elements which may be wrapped in:

- homogeneous containers (`std::vector`, `std::list`)
 - heterogeneous containers (`hpx::tuple`, `std::pair`, `std::array`) and re-assembles the pack with the result of the mapper. Mapping from one type to a different one is supported.
- Elements that aren't accepted by the mapper are routed through and preserved through the hierarchy.

```
// Maps all integers to floats
map_pack([](int value) {
    return float(value);
},
1, hpx::make_tuple(2, std::vector<int>{3, 4}), 5);
```

Return The mapped element or in case the pack contains multiple elements, the pack is wrapped into a `hpx::tuple`.

Exceptions

- `std::exception`: like objects which are thrown by an invocation to the mapper.

Parameters

- `mapper`: A callable object, which accept an arbitrary type and maps it to another type or the same one.
- `pack`: An arbitrary variadic pack which may contain any type.

hpx/pack_traversal/pack_traversal_async.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

Functions

```
template<typename Visitor, typename ...T>
auto traverse_pack_async (Visitor &&visitor, T&&... pack)
    Traverses the pack with the given visitor in an asynchronous way.
```

This function works in the same way as `traverse_pack`, however, we are able to suspend and continue the traversal at later time. Thus we require a visitor callable object which provides three `operator()` overloads as depicted by the code sample below:

```
struct my_async_visitor
{
    template <typename T>
    bool operator() (async_traverse_visit_tag, T&& element)
    {
        return true;
    }

    template <typename T, typename N>
    void operator() (async_traverse_detach_tag, T&& element, N&& next)
    {
    }

    template <typename T>
    void operator() (async_traverse_complete_tag, T&& pack)
    {
    }
};
```

See `traverse_pack` for a detailed description about the traversal behavior and capabilities.

Return A `hpx::intrusive_ptr` that references an instance of the given visitor object.

Parameters

- `visitor`: A visitor object which provides the three `operator()` overloads that were described above. Additionally the visitor must be compatible for referencing it from a `hpx::intrusive_ptr`. The visitor should must have a virtual destructor!
- `pack`: The arbitrary parameter pack which is traversed asynchronously. Nested objects inside containers and tuple like types are traversed recursively.

```
template<typename Allocator, typename Visitor, typename ...T>
auto traverse_pack_async_allocator (Allocator const &alloc, Visitor &&visitor,
                                    T&&... pack)
```

Traverses the pack with the given visitor in an asynchronous way.

This function works in the same way as `traverse_pack`, however, we are able to suspend and continue the traversal at later time. Thus we require a visitor callable object which provides three `operator()` overloads as depicted by the code sample below:

```
struct my_async_visitor
{
    template <typename T>
    bool operator()(async_traverse_visit_tag, T&& element)
    {
        return true;
    }

    template <typename T, typename N>
    void operator()(async_traverse_detach_tag, T&& element, N&& next)
    {
    }

    template <typename T>
    void operator()(async_traverse_complete_tag, T&& pack)
    {
    }
};
```

See `traverse_pack` for a detailed description about the traversal behavior and capabilities.

Return A `hpx::intrusive_ptr` that references an instance of the given visitor object.

Parameters

- `visitor`: A visitor object which provides the three `operator()` overloads that were described above. Additionally the visitor must be compatible for referencing it from a `hpx::intrusive_ptr`. The visitor should have a virtual destructor!
- `pack`: The arbitrary parameter pack which is traversed asynchronously. Nested objects inside containers and tuple like types are traversed recursively.
- `alloc`: Allocator instance to use to create the traversal frame.

hpx/pack_traversal/unwrap.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

```
template<typename ...Args>
auto unwrap(Args&&... args)
```

A helper function for retrieving the actual result of any `hpx::future` like type which is wrapped in an arbitrary way.

Unwraps the given pack of arguments, so that any `hpx::future` object is replaced by its future result type in the argument pack:

- `hpx::future<int> -> int`
- `hpx::future<std::vector<float>> -> std::vector<float>`
- `std::vector<future<float>> -> std::vector<float>`

The function is capable of unwrapping `hpx::future` like objects that are wrapped inside any container or tuple like type, see `hpx::util::map_pack()` for a detailed description about which surrounding types are supported. Non `hpx::future` like types are permitted as arguments and passed through.

```
// Single arguments
int i1 = hpx::unwrap(hpx::make_ready_future(0));

// Multiple arguments
hpx::tuple<int, int> i2 =
    hpx::unwrap(hpx::make_ready_future(1),
                hpx::make_ready_future(2));
```

Note This function unwraps the given arguments until the first traversed nested hpx::future which corresponds to an unwrapping depth of one. See `hpx::unwrap_n()` for a function which unwraps the given arguments to a particular depth or `hpx::unwrap_all()` that unwraps all future like objects recursively which are contained in the arguments.

Return Depending on the count of arguments this function returns a `hpx::tuple` containing the unwrapped arguments if multiple arguments are given. In case the function is called with a single argument, the argument is unwrapped and returned.

Parameters

- `args`: the arguments that are unwrapped which may contain any arbitrary future or non future type.

Exceptions

- `std::exception`: like objects in case any of the given wrapped hpx::future objects were resolved through an exception. See `hpx::future::get()` for details.

```
template<std::size_t Depth, typename ...Args>
auto unwrap_n(Args&&... args)
```

An alterntive version of `hpx::unwrap()`, which unwraps the given arguments to a certain depth of hpx::future like objects.

See `unwrap` for a detailed description.

Template Parameters

- `Depth`: The count of hpx::future like objects which are unwrapped maximally.

```
template<typename ...Args>
auto unwrap_all(Args&&... args)
```

An alterntive version of `hpx::unwrap()`, which unwraps the given arguments recursively so that all contained hpx::future like objects are replaced by their actual value.

See `hpx::unwrap()` for a detailed description.

```
template<typename T>
auto unwrapping(T &&callable)
```

Returns a callable object which unwraps its arguments upon invocation using the `hpx::unwrap()` function and then passes the result to the given callable object.

```
auto callable = hpx::unwrapping([](int left, int right) {
    return left + right;
});
```

(continues on next page)

(continued from previous page)

```
int il = callable(hpx::make_ready_future(1) ,
                   hpx::make_ready_future(2));
```

See `hpx::unwrap()` for a detailed description.

Parameters

- `callable`: the callable object which is called with the result of the corresponding `unwrap` function.

```
template<std::size_t Depth, typename T>
auto unwrapping_n (T && callable)
```

Returns a callable object which unwraps its arguments upon invocation using the `hpx::unwrap_n()` function and then passes the result to the given callable object.

See `hpx::unwrapping()` for a detailed description.

```
template<typename T>
auto unwrapping_all (T && callable)
```

Returns a callable object which unwraps its arguments upon invocation using the `hpx::unwrap_all()` function and then passes the result to the given callable object.

See `hpx::unwrapping()` for a detailed description.

namespace functional

struct `unwrap`

#include <unwrap.hpp> A helper function object for functionally invoking `hpx::unwrap`. For more information please refer to its documentation.

struct `unwrap_all`

#include <unwrap.hpp> A helper function object for functionally invoking `hpx::unwrap_all`. For more information please refer to its documentation.

```
template<std::size_t Depth>
```

struct `unwrap_n`

#include <unwrap.hpp> A helper function object for functionally invoking `hpx::unwrap_n`. For more information please refer to its documentation.

namespace util

Functions

```
template<typename ...Args>
auto unwrap (Args&&... args)
```

```
template<std::size_t Depth, typename ...Args>
auto unwrap_n (Args&&... args)
```

```
template<typename ...Args>
auto unwrap_all (Args&&... args)
```

```
template<typename T>
```

```
auto unwrapping(T &&callable)

template<std::size_t Depth, typename T>
auto unwrapping_n(T &&callable)

template<typename T>
auto unwrapping_all(T &&callable)

namespace functional
```

Functions

```
struct hpx::util::functional::HPX_DEPRECATED_V(1, 7, "Please use hpx::functional
```

preprocessor

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/preprocessor/cat.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_PP_CAT(*A, B*)

Concatenates the tokens *A* and *B* into a single token. Evaluates to *AB*

Parameters

- *A*: First token
- *B*: Second token

hpx/preprocessor/expand.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_PP_EXPAND(*X*)

The **HPX_PP_EXPAND** macro performs a double macro-expansion on its argument. This macro can be used to produce a delayed preprocessor expansion.

Parameters

- *X*: Token to be expanded twice

Example:

```
#define MACRO(a, b, c) (a) (b) (c)
#define ARGS() (1, 2, 3)

HPX_PP_EXPAND(MACRO ARGS()) // expands to (1) (2) (3)
```

hpx/preprocessor/nargs.hpp

See [Public API](#) for a list of names and headers that are part of the public *HPX* API.

Defines

HPX_PP_NARGS (...)

Expands to the number of arguments passed in

Example Usage:

```
HPX_PP_NARGS (hpx, pp, nargs)
HPX_PP_NARGS (hpx, pp)
HPX_PP_NARGS (hpx)
```

Parameters

- ...: The variadic number of arguments

Expands to:

```
3
2
1
```

hpx/preprocessor/stringize.hpp

See [Public API](#) for a list of names and headers that are part of the public *HPX* API.

Defines

HPX_PP_STRINGIZE (X)

The *HPX_PP_STRINGIZE* macro stringizes its argument after it has been expanded.

The passed argument X will expand to "X". Note that the stringizing operator (#) prevents arguments from expanding. This macro circumvents this shortcoming.

Parameters

- X: The text to be converted to a string literal

hpx/preprocessor/strip_parens.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_PP_STRIP_PARENS (X)

For any symbol X, this macro returns the same symbol from which potential outer parens have been removed. If no outer parens are found, this macros evaluates to X itself without error.

The original implementation of this macro is from Steven Watanbe as shown in <http://boost.2283326.n4.nabble.com/preprocessor-removing-parentheses-td2591973.html#a2591976>

```
HPX_PP_STRIP_PARENS (no_parens)
HPX_PP_STRIP_PARENS ( (with_parens) )
```

Example Usage:

Parameters

- X: Symbol to strip parens from

This produces the following output

```
no_parens
with_parens
```

resiliency

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/resiliency/replay_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace resiliency
```

```
    namespace experimental
```

Functions

```
template<typename BaseExecutor, typename Validate>
replay_executor<BaseExecutor, typename std::decay<Validate>::type> make_replay_executor (BaseExecutor&exec,
n,
Val-
i-
date
&&val-
i-
date)
```

```
template<typename BaseExecutor>
replay_executor<BaseExecutor, detail::replay_validator> make_replay_executor (BaseExecutor&exec,
n)
```

```
template<typename BaseExecutor, typename Validate>
class replay_executor
```

Public Types

```
template<>
using execution_category = typename BaseExecutor::execution_category

template<>
using executor_parameters_type = typename BaseExecutor::executor_parameters_type

template<typename Result>
using future_type = typename hpx::parallel::execution::executor_future<BaseExecutor, Result>::type
```

Public Functions

```
template<typename F>
replay_executor (BaseExecutor&exec, std::size_t n, F &&f)
```

```
bool operator== (replay_executor const &rhs) const
```

```
bool operator!= (replay_executor const &rhs) const
```

```
replay_executor const &context() const
```

```
template<typename F, typename ...Ts>
decltype(auto) async_execute (F &&f, Ts&&... ts) const
```

```
template<typename F, typename S, typename ...Ts>
decltype(auto) bulk_async_execute (F &&f, S const &shape, Ts&&... ts) const
```

Public Static Attributes

```
constexpr int num_spread = 4  
constexpr int num_tasks = 128
```

Private Members

```
BaseExecutor &exec_  
std::size_t replay_count_  
Validate validator_
```

hpx/resiliency/replicate_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace resiliency
```

```
namespace experimental
```

Functions

```
template<typename BaseExecutor, typename Voter, typename Validate>  
replicate_executor<BaseExecutor, typename std::decay<Voter>::type, typename std::decay<Validate>::type> make_
```

```
template<typename BaseExecutor, typename Validate>  
replicate_executor<BaseExecutor, detail::replicate_voter, typename std::decay<Validate>::type> make_replicat
```

```
template<typename BaseExecutor>
```

```
replicate_executor<BaseExecutor, detail::replicate_voter, detail::replicate_validator> make_replicate_executor
```

```
template<typename BaseExecutor, typename Vote, typename Validate>
class replicate_executor
```

Public Types

```
template<>
using execution_category = typename BaseExecutor::execution_category
template<>
using executor_parameters_type = typename BaseExecutor::executor_parameters_type
template<typename Result>
using future_type = typename hpx::parallel::execution::executor_future<BaseExecutor, Result>::type
```

Public Functions

```
template<typename V, typename F>
replicate_executor(BaseExecutor &exec, std::size_t n, V &&v, F &&f)
bool operator==(replicate_executor const &rhs) const
bool operator!=(replicate_executor const &rhs) const
replicate_executor const &context() const
template<typename F, typename ...Ts>
decltype(auto) async_execute(F &&f, Ts&&... ts) const
template<typename F, typename S, typename ...Ts>
decltype(auto) bulk_async_execute(F &&f, S const &shape, Ts&&... ts) const
```

Public Static Attributes

```
constexpr int num_spread = 4
constexpr int num_tasks = 128
```

Private Members

```
BaseExecutor &exec_
std::size_t replicate_count_
Vote voter_
Validate validator_
```

runtime_configuration

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/runtime_configuration/component_commandline_base.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_COMMANDLINE_REGISTRY (*RegistryType, componentname*)

The macro *HPX_REGISTER_COMMANDLINE_REGISTRY* is used to register the given component factory with Hpx.Plugin. This macro has to be used for each of the components.

HPX_REGISTER_COMMANDLINE_REGISTRY_DYNAMIC (*RegistryType, componentname*)

HPX_REGISTER_COMMANDLINE_OPTIONS ()

The macro *HPX_REGISTER_COMMANDLINE_OPTIONS* is used to define the required Hpx.Plugin entry point for the command line option registry. This macro has to be used in not more than one compilation unit of a component module.

HPX_REGISTER_COMMANDLINE_OPTIONS_DYNAMIC ()

namespace hpx

namespace components

struct component_commandline_base

#include <component_commandline_base.hpp> The *component_commandline_base* has to be used as a base class for all component command-line line handling registries.

Public Functions

virtual ~component_commandline_base ()

virtual hpx::program_options::options_description add_commandline_options () = 0

Return any additional command line options valid for this component.

Return The module is expected to fill a *options_description* object with any additional command line options this component will handle.

Note This function will be executed by the runtime system during system startup.

hpx/runtime_configuration/component_registry_base.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_COMPONENT_REGISTRY (*RegistryType, componentname*)

This macro is used to register the given component factory with Hpx.Plugin. This macro has to be used for each of the components.

HPX_REGISTER_COMPONENT_REGISTRY_DYNAMIC (*RegistryType, componentname*)

HPX_REGISTER_REGISTRY_MODULE()

This macro is used to define the required Hpx.Plugin entry points. This macro has to be used in exactly one compilation unit of a component module.

HPX_REGISTER_REGISTRY_MODULE_DYNAMIC()

namespace hpx

namespace components

struct component_registry_base

#include <component_registry_base.hpp> The `component_registry_base` has to be used as a base class for all component registries.

Public Functions

virtual ~component_registry_base()

virtual bool get_component_info (std::vector<std::string> &fillini, std::string const &filepath, bool is_static = false) = 0

Return the ini-information for all contained components.

Return Returns *true* if the parameter *fillini* has been successfully initialized with the registry data of all implemented in this module.

Parameters

- *fillini*: [in, out] The module is expected to fill this vector with the ini-information (one line per vector element) for all components implemented in this module.

virtual void register_component_type () = 0

Return the unique identifier of the component type this factory is responsible for.

Return Returns the unique identifier of the component type this factory instance is responsible for. This function throws on any error.

Parameters

- *locality*: [in] The id of the locality this factory is responsible for.
- *agas_client*: [in] The AGAS client to use for component id registration (if needed).

hpx/runtime_configuration/plugin_registry_base.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_PLUGIN_BASE_REGISTRY (*PluginType, name*)

This macro is used to register the given component factory with Hpx.Plugin. This macro has to be used for each of the components.

HPX_REGISTER_PLUGIN_REGISTRY_MODULE ()

This macro is used to define the required Hpx.Plugin entry points. This macro has to be used in exactly one compilation unit of a component module.

HPX_REGISTER_PLUGIN_REGISTRY_MODULE_DYNAMIC ()

```
namespace hpx
```

```
namespace plugins
```

```
struct plugin_registry_base
```

#include <plugin_registry_base.hpp> The `plugin_registry_base` has to be used as a base class for all plugin registries.

Public Functions

virtual ~plugin_registry_base()

virtual bool get_plugin_info (std::vector<std::string> &fillini) = 0

Return the configuration information for any plugin implemented by this module

Return Returns *true* if the parameter *fillini* has been successfully initialized with the registry data of all implemented in this module.

Parameters

- *fillini*: [in, out] The module is expected to fill this vector with the ini-information (one line per vector element) for all plugins implemented in this module.

virtual void init (int*, char*, util::runtime_configuration&)**

hpx/runtime_configuration/runtime_mode.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Enums

`enum runtime_mode`

A HPX runtime can be executed in two different modes: console mode and worker mode.

Values:

`invalid = -1`

`console = 0`

The runtime is the console locality.

`worker = 1`

The runtime is a worker locality.

`connect = 2`

The runtime is a worker locality connecting late

`local = 3`

The runtime is fully local.

`default_ = 4`

The runtime mode will be determined based on the command line arguments

`last`

Functions

`char const *get_runtime_mode_name (runtime_mode state)`

Get the readable string representing the name of the given runtime_mode constant.

`runtime_mode get_runtime_mode_from_name (std::string const &mode)`

Returns the internal representation (runtime_mode constant) from the readable string representing the name.

This represents the internal representation from the readable string representing the name.

Parameters

- `mode`: this represents the runtime mode

`runtime_local`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`hpx/runtime_local/component_startup_shutdown_base.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_STARTUP_SHUTDOWN_REGISTRY (*RegistryType, componentname*)

This macro is used to register the given component factory with Hpx.Plugin. This macro has to be used for each of the components.

HPX_REGISTER_STARTUP_SHUTDOWN_REGISTRY_DYNAMIC (*RegistryType, componentname*)

HPX_REGISTER_STARTUP_SHUTDOWN_FUNCTIONS ()

This macro is used to define the required Hpx.Plugin entry point for the startup/shutdown registry. This macro has to be used in not more than one compilation unit of a component module.

HPX_REGISTER_STARTUP_SHUTDOWN_FUNCTIONS_DYNAMIC ()

```
namespace hpx
```

```
namespace components
```

```
struct component_startup_shutdown_base
```

#include <component_startup_shutdown_base.hpp> The `component_startup_shutdown_base` has to be used as a base class for all component startup/shutdown registries.

Public Functions

```
virtual ~component_startup_shutdown_base()
```

```
virtual bool get_startup_function (startup_function_type &startup, bool &pre_startup) = 0
```

Return any startup function for this component.

Return Returns *true* if the parameter *startup* has been successfully initialized with the startup function.

Parameters

- *startup*: [in, out] The module is expected to fill this function object with a reference to a startup function. This function will be executed by the runtime system during system startup.

```
virtual bool get_shutdown_function (shutdown_function_type &shutdown, bool &pre_shutdown) = 0
```

Return any shutdown function for this component.

Return Returns *true* if the parameter *shutdown* has been successfully initialized with the shutdown function.

Parameters

- *shutdown*: [in, out] The module is expected to fill this function object with a reference to a shutdown function. This function will be executed by the runtime system during system shutdown.

hpx/runtime_local/custom_exception_info.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`std::string diagnostic_information(exception_info const &xi)`

Extract the diagnostic information embedded in the given exception and return a string holding a formatted message.

The function `hpx::diagnostic_information` can be used to extract all diagnostic information stored in the given exception instance as a formatted string. This simplifies debug output as it composes the diagnostics into one, easy to use function call. This includes the name of the source file and line number, the sequence number of the OS-thread and the HPX-thread id, the locality id and the stack backtrace of the point where the original exception was thrown.

Return The formatted string holding all of the available diagnostic information stored in the given exception instance.

See `hpx::get_error_locality_id()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`, `hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`, `hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`, `hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for all diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if any of the required allocation operations fail)

`std::uint32_t get_error_locality_id(hpx::exception_info const &xi)`

Return the locality id where the exception was thrown.

The function `hpx::get_error_locality_id` can be used to extract the diagnostic information element representing the locality id as stored in the given exception instance.

Return The locality id of the locality where the exception was thrown. If the exception instance does not hold this information, the function will return `hpx::naming::invalid_locality_id`.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`, `hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`, `hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`, `hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- nothing:

`std::string get_error_host_name(hpx::exception_info const &xi)`

Return the hostname of the locality where the exception was thrown.

The function `hpx::get_error_host_name` can be used to extract the diagnostic information element representing the host name as stored in the given exception instance.

Return The hostname of the locality where the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_process_id()`, `hpx::get_error_function_name()`,
`hpx::get_error_file_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`,
`hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`,
`hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`std::int64_t get_error_process_id(hpx::exception_info const &xi)`

Return the (operating system) process id of the locality where the exception was thrown.

The function `hpx::get_error_process_id` can be used to extract the diagnostic information element representing the process id as stored in the given exception instance.

Return The process id of the OS-process which threw the exception. If the exception instance does not hold this information, the function will return 0.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_function_name()`,
`hpx::get_error_file_name()`, `hpx::get_error_line_number()`, `hpx::get_error_os_thread()`,
`hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`, `hpx::get_error()`,
`hpx::get_error_backtrace()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- nothing:

`std::string get_error_env(hpx::exception_info const &xi)`

Return the environment of the OS-process at the point the exception was thrown.

The function `hpx::get_error_env` can be used to extract the diagnostic information element representing the environment of the OS-process collected at the point the exception was thrown.

Return The environment from the point the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`,
`hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`,
`hpx::get_error()`, `hpx::get_error_backtrace()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`std::string get_error_backtrace (hpx::exception_info const &xi)`

Return the stack backtrace from the point the exception was thrown.

The function `hpx::get_error_backtrace` can be used to extract the diagnostic information element representing the stack backtrace collected at the point the exception was thrown.

Return The stack back trace from the point the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`,
`hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_thread_description()`,
`hpx::get_error()`, `hpx::get_error_env()`, `hpx::get_error_what()`, `hpx::get_error_config()`,
`hpx::get_error_state()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`std::size_t get_error_os_thread (hpx::exception_info const &xi)`

Return the sequence number of the OS-thread used to execute HPX-threads from which the exception was thrown.

The function `hpx::get_error_os_thread` can be used to extract the diagnostic information element representing the sequence number of the OS-thread as stored in the given exception instance.

Return The sequence number of the OS-thread used to execute the HPX-thread from which the exception was thrown. If the exception instance does not hold this information, the function will return `std::size(-1)`.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`,

hpx::get_error_thread_id(), *hpx::get_error_thread_description()*, *hpx::get_error()*,
hpx::get_error_backtrace(), *hpx::get_error_env()*, *hpx::get_error_what()*, *hpx::get_error_config()*,
hpx::get_error_state()

Parameters

- *xi*: The parameter *e* will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: *hpx::exception_info*, *hpx::error_code*, *std::exception*, or *std::exception_ptr*.

Exceptions

- nothing:

std::size_t get_error_thread_id(hpx::exception_info const &xi)

Return the unique thread id of the HPX-thread from which the exception was thrown.

The function *hpx::get_error_thread_id* can be used to extract the diagnostic information element representing the HPX-thread id as stored in the given exception instance.

Return The unique thread id of the HPX-thread from which the exception was thrown. If the exception instance does not hold this information, the function will return *std::size_t(0)*.

See *hpx::diagnostic_information()*, *hpx::get_error_host_name()*, *hpx::get_error_process_id()*,
hpx::get_error_function_name(), *hpx::get_error_file_name()*, *hpx::get_error_line_number()*,
hpx::get_error_os_thread(), *hpx::get_error_thread_description()*, *hpx::get_error()*,
hpx::get_error_backtrace(), *hpx::get_error_env()*, *hpx::get_error_what()*, *hpx::get_error_config()*,
hpx::get_error_state()

Parameters

- *xi*: The parameter *e* will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: *hpx::exception_info*, *hpx::error_code*, *std::exception*, or *std::exception_ptr*.

Exceptions

- nothing:

std::string get_error_thread_description(hpx::exception_info const &xi)

Return any additionally available thread description of the HPX-thread from which the exception was thrown.

The function *hpx::get_error_thread_description* can be used to extract the diagnostic information element representing the additional thread description as stored in the given exception instance.

Return Any additionally available thread description of the HPX-thread from which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See *hpx::diagnostic_information()*, *hpx::get_error_host_name()*, *hpx::get_error_process_id()*,
hpx::get_error_function_name(), *hpx::get_error_file_name()*, *hpx::get_error_line_number()*,
hpx::get_error_os_thread(), *hpx::get_error_thread_id()*, *hpx::get_error_backtrace()*,
hpx::get_error_env(), *hpx::get_error()*, *hpx::get_error_state()*, *hpx::get_error_what()*,
hpx::get_error_config()

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`std::string get_error_config(hpx::exception_info const &xi)`

Return the HPX configuration information point from which the exception was thrown.

The function `hpx::get_error_config` can be used to extract the HPX configuration information element representing the full HPX configuration information as stored in the given exception instance.

Return Any additionally available HPX configuration information the point from which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`,
`hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_backtrace()`,
`hpx::get_error_env()`, `hpx::get_error()`, `hpx::get_error_state()` `hpx::get_error_what()`,
`hpx::get_error_thread_description()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

`std::string get_error_state(hpx::exception_info const &xi)`

Return the HPX runtime state information at which the exception was thrown.

The function `hpx::get_error_state` can be used to extract the HPX runtime state information element representing the state the runtime system is currently in as stored in the given exception instance.

Return The point runtime state at the point at which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See `hpx::diagnostic_information()`, `hpx::get_error_host_name()`, `hpx::get_error_process_id()`,
`hpx::get_error_function_name()`, `hpx::get_error_file_name()`, `hpx::get_error_line_number()`,
`hpx::get_error_os_thread()`, `hpx::get_error_thread_id()`, `hpx::get_error_backtrace()`,
`hpx::get_error_env()`, `hpx::get_error()`, `hpx::get_error_what()`, `hpx::get_error_thread_description()`

Parameters

- `xi`: The parameter `e` will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: `hpx::exception_info`, `hpx::error_code`, `std::exception`, or `std::exception_ptr`.

Exceptions

- `std::bad_alloc`: (if one of the required allocations fails)

hpx/runtime_local/get_locality_id.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`std::uint32_t get_locality_id(error_code &ec = throws)`

Return the number of the locality this function is being called from.

This function returns the id of the current locality.

Note The returned value is zero based and its maximum value is smaller than the overall number of localities the current application is running on (as returned by `get_num_localities()`).

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

hpx/runtime_local/get_locality_name.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`std::string get_locality_name()`

Return the name of the locality this function is called on.

This function returns the name for the locality on which this function is called.

Return This function returns the name for the locality on which the function is called. The name is retrieved from the underlying networking layer and may be different for different parcelports.

See `future<std::string> get_locality_name(hpx::id_type const& id)`

`hpx/runtime_local/get_num_all_localities.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

`std::uint32_t get_initial_num_localities()`

Return the number of localities which were registered at startup for the running application.

The function `get_initial_num_localities` returns the number of localities which were connected to the console at application startup.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

See `hpx::find_all_localities, hpx::get_num_localities`

`hpx::future<std::uint32_t> get_num_localities()`

Asynchronously return the number of localities which are currently registered for the running application.

The function `get_num_localities` asynchronously returns the number of localities currently connected to the console. The returned future represents the actual result.

Note This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

See `hpx::find_all_localities, hpx::get_num_localities`

`std::uint32_t get_num_localities(launch::sync_policy, error_code &ec = throws)`

Return the number of localities which are currently registered for the running application.

The function `get_num_localities` returns the number of localities currently connected to the console.

Note This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

See `hpx::find_all_localities, hpx::get_num_localities`

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

hpx/runtime_local/get_os_thread_count.hpp

See [Public API](#) for a list of names and headers that are part of the public *HPX* API.

namespace hpx

Functions

`std::size_t get_os_thread_count()`

Return the number of OS-threads running in the runtime instance the current HPX-thread is associated with.

`std::size_t get_os_thread_count (threads::executor const &exec)`

Return the number of worker OS- threads used by the given executor to execute HPX threads.

This function returns the number of cores used to execute HPX threads for the given executor. If the function is called while no HPX runtime system is active, it will return zero. If the executor is not valid, this function will fall back to retrieving the number of OS threads used by HPX.

Parameters

- `exec`: [in] The executor to be used.

hpx/runtime_local/get_thread_name.hpp

See [Public API](#) for a list of names and headers that are part of the public *HPX* API.

namespace hpx

Functions

`std::string get_thread_name()`

Return the name of the calling thread.

This function returns the name of the calling thread. This name uniquely identifies the thread in the context of HPX. If the function is called while no HPX runtime system is active, the result will be “<unknown>”.

hpx/runtime_local/get_worker_thread_num.hpp

See [Public API](#) for a list of names and headers that are part of the public *HPX* API.

hpx/runtime_local/report_error.hpp

See [Public API](#) for a list of names and headers that are part of the public *HPX* API.

namespace hpx

Functions

```
void report_error (std::size_t num_thread, std::exception_ptr const &e)
    The function report_error reports the given exception to the console.
```

```
void report_error (std::exception_ptr const &e)
    The function report_error reports the given exception to the console.
```

hpx/runtime_local/runtime_local.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

```
void set_error_handlers ()
class runtime
```

Public Types

```
using notification_policy_type = threads::policies::callback_notifier
    Generate a new notification policy instance for the given thread name prefix

using hpx_main_function_type = int ()
    The hpx_main_function_type is the default function type usable as the main HPX thread function.

using hpx_errorssink_function_type = void (std::uint32_t, std::string const&)
```

Public Functions

```
virtual notification_policy_type get_notification_policy (char const *prefix, runtime_local::os_thread_type type)

state get_state () const

void set_state (state s)

runtime (hpx::util::runtime_configuration &rtcfg, bool initialize)
    Construct a new HPX runtime instance.

virtual ~runtime ()
    The destructor makes sure all HPX runtime services are properly shut down before exiting.

void on_exit (hpx::function<void> > const &f)
    Manage list of functions to call on exit.

void starting ()
    Manage runtime ‘stopped’ state.

void stopping ()
    Call all registered on_exit functions.
```

`bool stopped() const`

This accessor returns whether the runtime instance has been stopped.

`hpx::util::runtime_configuration &get_config()`

access configuration information

`hpx::util::runtime_configuration const &get_config() const`

`std::size_t get_instance_number() const`

`util::thread_mapper &get_thread_mapper()`

Return a reference to the internal PAPI thread manager.

`threads::topology const &get_topology() const`

`virtual int run(hpx::function<hpx_main_function_type> const &func)`

Run the HPX runtime system, use the given function for the main *thread* and block waiting for all threads to finish.

Note The parameter *func* is optional. If no function is supplied, the runtime system will simply wait for the shutdown action without explicitly executing any main thread.

Return This function will return the value as returned as the result of the invocation of the function object given by the parameter *func*.

Parameters

- *func*: [in] This is the main function of an HPX application. It will be scheduled for execution by the thread manager as soon as the runtime has been initialized. This function is expected to expose an interface as defined by the typedef *hpx_main_function_type*. This parameter is optional and defaults to none main thread function, in which case all threads have to be scheduled explicitly.

`virtual int run()`

Run the HPX runtime system, initially use the given number of (OS) threads in the thread-manager and block waiting for all threads to finish.

Return This function will always return 0 (zero).

`virtual void rethrow_exception()`

Rethrow any stored exception (to be called after *stop()*)

`virtual int start(hpx::function<hpx_main_function_type> const &func, bool blocking = false)`

Start the runtime system.

Return If a *blocking* is a true, this function will return the value as returned as the result of the invocation of the function object given by the parameter *func*. Otherwise it will return zero.

Parameters

- *func*: [in] This is the main function of an HPX application. It will be scheduled for execution by the thread manager as soon as the runtime has been initialized. This function is expected to expose an interface as defined by the typedef *hpx_main_function_type*.
- *blocking*: [in] This allows to control whether this call blocks until the runtime system has been stopped. If this parameter is *true* the function `runtime::start` will call `runtime::wait` internally.

`virtual int start(bool blocking = false)`

Start the runtime system.

Return If a blocking is a true, this function will return the value as returned as the result of the invocation of the function object given by the parameter `func`. Otherwise it will return zero.

Parameters

- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been stopped. If this parameter is *true* the function `runtime::start` will call `runtime::wait` internally .

virtual int wait()

Wait for the shutdown action to be executed.

Return This function will return the value as returned as the result of the invocation of the function object given by the parameter `func`.

virtual void stop (bool blocking = true)

Initiate termination of the runtime system.

Parameters

- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been fully stopped. If this parameter is *false* then this call will initiate the stop action but will return immediately. Use a second call to stop with this parameter set to *true* to wait for all internal work to be completed.

virtual int suspend()

Suspend the runtime system.

virtual int resume()

Resume the runtime system.

virtual int finalize (double)

virtual bool is_networking_enabled()

Return true if networking is enabled.

virtual hpx::threads::threadmanager &get_thread_manager()

Allow access to the thread manager instance used by the HPX runtime.

virtual std::string here () const

Returns a string of the locality endpoints (usable in debug output)

virtual bool report_error (std::size_t num_thread, std::exception_ptr const &e, bool terminate_all = true)

Report a non-recoverable error to the runtime system.

Parameters

- `num_thread`: [in] The number of the operating system thread the error has been detected in.
- `e`: [in] This is an instance encapsulating an exception which lead to this function call.

virtual bool report_error (std::exception_ptr const &e, bool terminate_all = true)

Report a non-recoverable error to the runtime system.

Note This function will retrieve the number of the current shepherd thread and forward to the `report_error` function above.

Parameters

- `e`: [in] This is an instance encapsulating an exception which lead to this function call.

```
virtual void add_pre_startup_function(startup_function_type f)
```

Add a function to be executed inside a HPX thread before hpx_main but guaranteed to be executed before any startup function registered with *add_startup_function*.

Note The difference to a startup function is that all pre-startup functions will be (system-wide) executed before any startup function.

Parameters

- f: The function ‘f’ will be called from inside a HPX thread before hpx_main is executed. This is very useful to setup the runtime environment of the application (install performance counters, etc.)

```
virtual void add_startup_function(startup_function_type f)
```

Add a function to be executed inside a HPX thread before hpx_main

Parameters

- f: The function ‘f’ will be called from inside a HPX thread before hpx_main is executed. This is very useful to setup the runtime environment of the application (install performance counters, etc.)

```
virtual void add_pre_shutdown_function(shutdown_function_type f)
```

Add a function to be executed inside a HPX thread during hpx::finalize, but guaranteed before any of the shutdown functions is executed.

Note The difference to a shutdown function is that all pre-shutdown functions will be (system-wide) executed before any shutdown function.

Parameters

- f: The function ‘f’ will be called from inside a HPX thread while hpx::finalize is executed. This is very useful to tear down the runtime environment of the application (uninstall performance counters, etc.)

```
virtual void add_shutdown_function(shutdown_function_type f)
```

Add a function to be executed inside a HPX thread during hpx::finalize

Parameters

- f: The function ‘f’ will be called from inside a HPX thread while hpx::finalize is executed. This is very useful to tear down the runtime environment of the application (uninstall performance counters, etc.)

```
virtual hpx::util::io_service_pool* get_thread_pool(char const *name)
```

Access one of the internal thread pools (io_service instances) HPX is using to perform specific tasks. The three possible values for the argument name are “main_pool”, “io_pool”, “parcel_pool”, and “timer_pool”. For any other argument value the function will return zero.

```
virtual bool register_thread(char const *name, std::size_t num = 0, bool service_thread  
                           = true, error_code &ec = throws)
```

Register an external OS-thread with HPX.

This function should be called from any OS-thread which is external to HPX (not created by HPX), but which needs to access HPX functionality, such as setting a value on a promise or similar.

‘main’, ‘io’, ‘timer’, ‘parcel’, ‘worker’

Note The function will compose a thread name of the form ‘<name>-thread#<num>’ which is used to register the thread. It is the user’s responsibility to ensure that each (composed) thread name is unique. HPX internally uses the following names for the threads it creates, do not reuse those:

Parameters

- name: [in] The name to use for thread registration.
- num: [in] The sequence number to use for thread registration. The default for this parameter is zero.
- service_thread: [in] The thread should be registered as a service thread. The default for this parameter is ‘true’. Any service threads will be pinned to cores not currently used by any of the HPX worker threads.

Note This function should be called for each thread exactly once. It will fail if it is called more than once.

Return This function will return whether the requested operation succeeded or not.

virtual bool unregister_thread()

Unregister an external OS-thread with HPX.

This function will unregister any external OS-thread from HPX.

Note This function should be called for each thread exactly once. It will fail if it is called more than once. It will fail as well if the thread has not been registered before (see *register_thread*).

Return This function will return whether the requested operation succeeded or not.

virtual runtime_local::os_thread_data get_os_thread_data (std::string const &label)

Access data for a given OS thread that was previously registered by *register_thread*.

virtual bool enumerate_os_threads (hpx::function<bool> runtime_local::os_thread_data const > const &f const

Enumerate all OS threads that have registered with the runtime.

notification_policy_type::on_startstop_type on_start_func () const

notification_policy_type::on_startstop_type on_stop_func () const

notification_policy_type::on_error_type on_error_func () const

notification_policy_type::on_startstop_type on_start_func (notification_policy_type::on_startstop_type&&)

notification_policy_type::on_startstop_type on_stop_func (notification_policy_type::on_startstop_type&&)

notification_policy_type::on_error_type on_error_func (notification_policy_type::on_error_type&&)

virtual std::uint32_t get_locality_id(error_code &ec) const

virtual std::size_t get_num_worker_threads () const

virtual std::uint32_t get_num_localities (hpx::launch::sync_policy, error_code &ec) const

virtual std::uint32_t get_initial_num_localities () const

virtual hpx::future<std::uint32_t> get_num_localities () const

virtual std::string get_locality_name () const

virtual std::uint32_t assign_cores (std::string const&, std::uint32_t)

virtual std::uint32_t assign_cores ()

Public Static Functions

```
static std::uint64_t get_system_uptime()  
    Return the system uptime measure on the thread executing this call.
```

Protected Types

```
using on_exit_type = std::vector<hpx::function<void ()>>
```

Protected Functions

```
runtime (hpx::util::runtime_configuration &rtcfg)  
  
void set_notification_policies (notification_policy_type  
                                &&notifier,  
                                threads::detail::network_background_callback_type  
                                network_background_callback)  
  
void init ()  
    Common initialization for different constructors.  
  
void init_global_data ()  
  
void deinit_global_data ()  
  
threads::thread_result_type run_helper (hpx::function<runtime::hpx_main_function_type>  
                                         const &func, int &result, bool call_startup_functions)  
  
void wait_helper (std::mutex &mtx, std::condition_variable &cond, bool &running)
```

Protected Attributes

```
on_exit_type on_exit_functions_  
std::mutex mtx_  
hpx::util::runtime_configuration rtcfg_  
long instance_number_  
std::unique_ptr<util::thread_mapper> thread_support_  
threads::topology &topology_  
std::atomic<state> state_  
notification_policy_type::on_startstop_type on_start_func_  
notification_policy_type::on_startstop_type on_stop_func_  
notification_policy_type::on_error_type on_error_func_  
int result_  
std::exception_ptr exception_  
notification_policy_type main_pool_notifier_  
util::io_service_pool main_pool_  
notification_policy_type notifier_
```

```
std::unique_ptr<hpx::threads::threadmanager> thread_manager_
```

Protected Static Attributes

```
std::atomic<int> instance_number_counter_
```

Private Functions

```
void stop_helper(bool blocking, std::condition_variable &cond, std::mutex &mtx)
    Helper function to stop the runtime.
```

Parameters

- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been fully stopped. If this parameter is *false* then this call will initiate the stop action but will return immediately. Use a second call to stop with this parameter set to *true* to wait for all internal work to be completed.

```
void deinit_tss_helper(char const *context, std::size_t num)
```

```
void init_tss_ex(char const *context, runtime_local::os_thread_type type, std::size_t local_thread_num, std::size_t global_thread_num, char const *pool_name, char const *postfix, bool service_thread, error_code &ec)
```

```
void init_tss_helper(char const *context, runtime_local::os_thread_type type, std::size_t local_thread_num, std::size_t global_thread_num, char const *pool_name, char const *postfix, bool service_thread)
```

```
void notify_finalize()
```

```
void wait_finalize()
```

```
void call_startup_functions(bool pre_startup)
```

Private Members

```
std::list<startup_function_type> pre_startup_functions_
std::list<startup_function_type> startup_functions_
std::list<shutdown_function_type> pre_shutdown_functions_
std::list<shutdown_function_type> shutdown_functions_
bool stop_called_
bool stop_done_
std::condition_variable wait_condition_
namespace threads
```

Functions

```
char const *get_stack_size_name (std::ptrdiff_t size)
    Returns the stack size name.

    Get the readable string representing the given stack size constant.
```

Parameters

- `size`: this represents the stack size

```
std::ptrdiff_t get_default_stack_size ()
    Returns the default stack size.

    Get the default stack size in bytes.

std::ptrdiff_t get_stack_size (thread_stacksize)
    Returns the stack size corresponding to the given stack size enumeration.

    Get the stack size corresponding to the given stack size enumeration.
```

Parameters

- `size`: this represents the stack size

```
namespace util
```

Functions

```
bool retrieve_commandline_arguments (hpx::program_options::options_description
                                     const &app_options,
                                     hpx::program_options::variables_map &vm)

bool retrieve_commandline_arguments (std::string const &appname,
                                     hpx::program_options::variables_map &vm)
```

hpx/runtime_local/runtime_local_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
bool register_thread (runtime *rt, char const *name, error_code &ec = throws)
    Register the current kernel thread with HPX, this should be done once for each external OS-thread intended
    to invoke HPX functionality. Calling this function more than once will return false.

void unregister_thread (runtime *rt)
    Unregister the thread from HPX, this should be done once in the end before the external thread exists.

runtime_local::os_thread_data get_os_thread_data (std::string const &label)
    Access data for a given OS thread that was previously registered by register_thread. This function must
    be called from a thread that was previously registered with the runtime.
```

```
bool enumerate_os_threads (hpx::function<bool> os_thread_data const&
    > const &/& Enumerate all OS threads that have registered with the runtime.
```

std::size_t **get_runtime_instance_number** ()
Return the runtime instance number associated with the runtime instance the current thread is running in.

```
bool register_on_exit (hpx::function<void>)
    > const&/& Register a function to be called during system shutdown.
```

```
bool is_starting ()
Test whether the runtime system is currently being started.  
This function returns whether the runtime system is currently being started or not, e.g. whether the current state of the runtime system is hpx::state::startup
```

Note This function needs to be executed on a HPX-thread. It will return false otherwise.

```
bool tolerate_node_faults ()
Test if HPX runs in fault-tolerant mode.  
This function returns whether the runtime system is running in fault-tolerant mode
```

```
bool is_running ()
Test whether the runtime system is currently running.  
This function returns whether the runtime system is currently running or not, e.g. whether the current state of the runtime system is hpx::state::running
```

Note This function needs to be executed on a HPX-thread. It will return false otherwise.

```
bool is_stopped ()
Test whether the runtime system is currently stopped.  
This function returns whether the runtime system is currently stopped or not, e.g. whether the current state of the runtime system is hpx::state::stopped
```

Note This function needs to be executed on a HPX-thread. It will return false otherwise.

```
bool is_stopped_or_shutting_down ()
Test whether the runtime system is currently being shut down.  
This function returns whether the runtime system is currently being shut down or not, e.g. whether the current state of the runtime system is hpx::state::stopped or hpx::state::shutdown
```

Note This function needs to be executed on a HPX-thread. It will return false otherwise.

```
std::size_t get_num_worker_threads ()
Return the number of worker OS- threads used to execute HPX threads.  
This function returns the number of OS-threads used to execute HPX threads. If the function is called while no HPX runtime system is active, it will return zero.
```

```
std::uint64_t get_system_uptime ()
Return the system uptime measure on the thread executing this call.
```

This function returns the system uptime measured in nanoseconds for the thread executing this call. If the function is called while no HPX runtime system is active, it will return zero.

hpx/runtime_local/service_executors.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

```
        namespace execution
```

Enums

```
enum service_executor_type
```

Values:

```
    io_thread_pool
```

Selects creating a service executor using the I/O pool of threads

```
    parcel_thread_pool
```

Selects creating a service executor using the parcel pool of threads

```
    timer_thread_pool
```

Selects creating a service executor using the timer pool of threads

```
    main_thread
```

Selects creating a service executor using the main thread

```
struct io_pool_executor : public service_executor
```

Public Functions

```
io_pool_executor()
```

```
struct main_pool_executor : public service_executor
```

Public Functions

```
main_pool_executor()
```

```
struct parcel_pool_executor : public service_executor
```

Public Functions

```
parcel_pool_executor(char const *name_suffix = "-tcp")
struct service_executor: public service_executor
```

Public Functions

```
service_executor(service_executor_type t, char const *name_suffix = "")
struct timer_pool_executor: public service_executor
```

Public Functions

```
timer_pool_executor()
```

hpx/runtime_local/shutdown_function.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

Typedefs

typedef *hpx*::move_only_function<void ()> **shutdown_function_type**

The type of a function which is registered to be executed as a shutdown or pre-shutdown function.

Functions

void register_pre_shutdown_function(*shutdown_function_type* f)

Add a function to be executed by a HPX thread during *hpx::finalize()* but guaranteed before any shutdown function is executed (system-wide)

Any of the functions registered with *register_pre_shutdown_function* are guaranteed to be executed by an HPX thread during the execution of *hpx::finalize()* before any of the registered shutdown functions are executed (see: *hpx::register_shutdown_function()*).

Note If this function is called while the pre-shutdown functions are being executed, or after that point, it will raise a invalid_status exception.

See *hpx::register_shutdown_function()*

Parameters

- *f*: [in] The function to be registered to run by an HPX thread as a pre-shutdown function.

void register_shutdown_function(*shutdown_function_type* f)

Add a function to be executed by a HPX thread during *hpx::finalize()* but guaranteed after any pre-shutdown function is executed (system-wide)

Any of the functions registered with *register_shutdown_function* are guaranteed to be executed by an HPX thread during the execution of *hpx::finalize()* after any of the registered pre-shutdown functions are executed (see: *hpx::register_pre_shutdown_function()*).

Note If this function is called while the shutdown functions are being executed, or after that point, it will raise a `invalid_status` exception.

See *hpx::register_pre_shutdown_function()*

Parameters

- `f`: [in] The function to be registered to run by an HPX thread as a shutdown function.

[hpx/runtime_local/startup_function.hpp](#)

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

Typedefs

typedef *hpx::move_only_function<void ()>* **startup_function_type**

The type of a function which is registered to be executed as a startup or pre-startup function.

Functions

void register_pre_startup_function(*startup_function_type f*)

Add a function to be executed by a HPX thread before `hpx_main` but guaranteed before any startup function is executed (system-wide).

Any of the functions registered with *register_pre_startup_function* are guaranteed to be executed by an HPX thread before any of the registered startup functions are executed (see *hpx::register_startup_function()*).

This function is one of the few API functions which can be called before the runtime system has been fully initialized. It will automatically stage the provided startup function to the runtime system during its initialization (if necessary).

Note If this function is called while the pre-startup functions are being executed or after that point, it will raise a `invalid_status` exception.

Parameters

- `f`: [in] The function to be registered to run by an HPX thread as a pre-startup function.

See *hpx::register_startup_function()*

void register_startup_function(*startup_function_type f*)

Add a function to be executed by a HPX thread before `hpx_main` but guaranteed after any pre-startup function is executed (system-wide).

Any of the functions registered with `register_startup_function` are guaranteed to be executed by an HPX thread after any of the registered pre-startup functions are executed (see: `hpx::register_pre_startup_function()`), but before `hpx_main` is being called.

This function is one of the few API functions which can be called before the runtime system has been fully initialized. It will automatically stage the provided startup function to the runtime system during its initialization (if necessary).

Note If this function is called while the startup functions are being executed or after that point, it will raise a `invalid_status` exception.

Parameters

- `f`: [in] The function to be registered to run by an HPX thread as a startup function.

See `hpx::register_pre_startup_function()`

`hpx/runtime_local/thread_hooks.hpp`

See *Public API* for a list of names and headers that are part of the public *HPX API*.

`namespace hpx`

Functions

`threads::policies::callback_notifier::on_startstop_type get_thread_on_start_func()`

Retrieve the currently installed start handler function. This is a function that will be called by HPX for each newly created thread that is made known to the runtime. HPX stores exactly one such function reference, thus the caller needs to make sure any newly registered start function chains into the previous one (see `register_thread_on_start_func`).

Return The currently installed error handler function.

Note This function can be called before the HPX runtime is initialized.

`threads::policies::callback_notifier::on_startstop_type get_thread_on_stop_func()`

Retrieve the currently installed stop handler function. This is a function that will be called by HPX for each newly created thread that is made known to the runtime. HPX stores exactly one such function reference, thus the caller needs to make sure any newly registered stop function chains into the previous one (see `register_thread_on_stop_func`).

Return The currently installed error handler function.

Note This function can be called before the HPX runtime is initialized.

`threads::policies::callback_notifier::on_error_type get_thread_on_error_func()`

Retrieve the currently installed error handler function. This is a function that will be called by HPX for each newly created thread that is made known to the runtime. HPX stores exactly one such function reference, thus the caller needs to make sure any newly registered error function chains into the previous one (see `register_thread_on_error_func`).

Return The currently installed error handler function.

Note This function can be called before the HPX runtime is initialized.

`threads::policies::callback_notifier::on_startstop_type register_thread_on_start_func (threads::policies::callback_notifier::on_startstop_type&&f)`

Set the currently installed start handler function. This is a function that will be called by HPX for each newly created thread that is made known to the runtime. HPX stores exactly one such function reference, thus the caller needs to make sure any newly registered start function chains into the previous one (see `get_thread_on_start_func`).

Return The previously registered function of this category. It is the user's responsibility to call that function if the callback is invoked by HPX.

Note This function can be called before the HPX runtime is initialized.

Parameters

- `f`: The function to install as the new start handler.

`threads::policies::callback_notifier::on_startstop_type register_thread_on_stop_func (threads::policies::callback_notifier::on_startstop_type&&f)`

Set the currently installed stop handler function. This is a function that will be called by HPX for each newly created thread that is made known to the runtime. HPX stores exactly one such function reference, thus the caller needs to make sure any newly registered stop function chains into the previous one (see `get_thread_on_stop_func`).

Return The previously registered function of this category. It is the user's responsibility to call that function if the callback is invoked by HPX.

Note This function can be called before the HPX runtime is initialized.

Parameters

- `f`: The function to install as the new stop handler.

`threads::policies::callback_notifier::on_error_type register_thread_on_error_func (threads::policies::callback_notifier::on_error_type&&f)`

Set the currently installed error handler function. This is a function that will be called by HPX for each newly created thread that is made known to the runtime. HPX stores exactly one such function reference, thus the caller needs to make sure any newly registered error function chains into the previous one (see `get_thread_on_error_func`).

Return The previously registered function of this category. It is the user's responsibility to call that function if the callback is invoked by HPX.

Note This function can be called before the HPX runtime is initialized.

Parameters

- `f`: The function to install as the new error handler.

`hpx/runtime_local/thread_pool_helpers.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace resource`

Functions

`std::size_t get_num_thread_pools()`

Return the number of thread pools currently managed by the *resource_partitioner*

`std::size_t get_num_threads()`

Return the number of threads in all thread pools currently managed by the *resource_partitioner*

`std::size_t get_num_threads(std::string const &pool_name)`

Return the number of threads in the given thread pool currently managed by the *resource_partitioner*

`std::size_t get_num_threads(std::size_t pool_index)`

Return the number of threads in the given thread pool currently managed by the *resource_partitioner*

`std::size_t get_pool_index(std::string const &pool_name)`

Return the internal index of the pool given its name.

`std::string const &get_pool_name(std::size_t pool_index)`

Return the name of the pool given its internal index.

`threads::thread_pool_base &get_thread_pool(std::string const &pool_name)`

Return the name of the pool given its name.

`threads::thread_pool_base &get_thread_pool(std::size_t pool_index)`

Return the thread pool given its internal index.

`bool pool_exists(std::string const &pool_name)`

Return true if the pool with the given name exists.

`bool pool_exists(std::size_t pool_index)`

Return true if the pool with the given index exists.

`namespace threads`

Functions

`std::int64_t get_thread_count(thread_schedule_state state = thread_schedule_state::unknown)`

The function *get_thread_count* returns the number of currently known threads.

Note If state == unknown this function will not only return the number of currently existing threads, but will add the number of registered task descriptions (which have not been converted into threads yet).

Parameters

- `state`: [in] This specifies the thread-state for which the number of threads should be retrieved.

```
std::int64_t get_thread_count (thread_priority priority, thread_schedule_state state =  
                           thread_schedule_state::unknown)
```

The function *get_thread_count* returns the number of currently known threads.

Note If *state* == *unknown* this function will not only return the number of currently existing threads, but will add the number of registered task descriptions (which have not been converted into threads yet).

Parameters

- *priority*: [in] This specifies the thread-priority for which the number of threads should be retrieved.
- *state*: [in] This specifies the thread-state for which the number of threads should be retrieved.

```
std::int64_t get_idle_core_count ()
```

The function *get_idle_core_count* returns the number of currently idling threads (cores).

```
mask_type get_idle_core_mask ()
```

The function *get_idle_core_mask* returns a bit-mask representing the currently idling threads (cores).

```
bool enumerate_threads (hpx::function<bool> thread_id_type
```

> **const** &*f*, *thread_schedule_state state* = *thread_schedule_state*::*unknown*)

The function *enumerate_threads* will invoke the given function *f* for each thread with a matching thread state.

Parameters

- *f*: [in] The function which should be called for each matching thread. Returning ‘false’ from this function will stop the enumeration process.
- *state*: [in] This specifies the thread-state for which the threads should be enumerated.

serialization

See *Public API* for a list of names and headers that are part of the public *HPX API*.

hpx/serialization/base_object.hpp

See *Public API* for a list of names and headers that are part of the public *HPX API*.

```
template<typename Derived, typename Base>  
struct base_object_type<Derived, Base, std::true_type>
```

Public Functions

```
constexpr base_object_type (Derived &d)
```

```
template<typename Archive>  
void save (Archive &ar, unsigned) const
```

```
template<typename Archive>  
void load (Archive &ar, unsigned)
```

```
HPX_SERIALIZATION_SPLIT_MEMBER ()
```

Public Members

```
Derived &d_
namespace hpx

namespace serialization
```

Functions

```
template<typename Base, typename Derived>
constexpr base_object_type<Derived, Base> base_object (Derived &d)

template<typename D, typename B>
output_archive &operator<< (output_archive &ar, base_object_type<D, B> t)

template<typename D, typename B>
input_archive &operator>> (input_archive &ar, base_object_type<D, B> t)

template<typename D, typename B>
output_archive &operator& (output_archive &ar, base_object_type<D, B> t)

template<typename D, typename B>
input_archive &operator& (input_archive &ar, base_object_type<D, B> t)

template<typename Derived, typename Base, typename Enable = typename hpx::traits::is_intrusive_polymorphic<Derived, Base>::value>
struct base_object_type
```

Public Functions

```
constexpr base_object_type (Derived &d)

template<typename Archive>
void serialize (Archive &ar, unsigned)
```

Public Members

Derived &d_

```
template<typename Derived, typename Base>
struct base_object_type<Derived, Base, std::true_type>
```

Public Functions

```
constexpr base_object_type (Derived &d)

template<typename Archive>
void save (Archive &ar, unsigned) const

template<typename Archive>
void load (Archive &ar, unsigned)

HPX_SERIALIZATION_SPLIT_MEMBER ()
```

Public Members

Derived & `d_`

synchronization

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`hpx/synchronization/barrier.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

```
template<typename OnCompletion = detail::empty_oncompletion>
class barrier
    #include <barrier.hpp> A barrier is a thread coordination mechanism whose lifetime consists of a sequence of barrier phases, where each phase allows at most an expected number of threads to block until the expected number of threads arrive at the barrier. [ Note: A barrier is useful for managing repeated tasks that are handled by multiple threads. - end note ] Each barrier phase consists of the following steps:
```

- The expected count is decremented by each call to `arrive` or `arrive_and_drop`.
- When the expected count reaches zero, the phase completion step is run. For the specialization with the default value of the `CompletionFunction` template parameter, the completion step is run as part of the call to `arrive` or `arrive_and_drop` that caused the expected count to reach zero. For other specializations, the completion step is run on one of the threads that arrived at the barrier during the phase.
- When the completion step finishes, the expected count is reset to what was specified by the `expected` argument to the constructor, possibly adjusted by calls to `arrive_and_drop`, and the next phase starts.

Each phase defines a phase synchronization point. Threads that arrive at the barrier during the phase can block on the phase synchronization point by calling `wait`, and will remain blocked until the phase completion step is run. The phase completion step that is executed at the end of each phase has the following effects:

- Invokes the completion function, equivalent to `completion()`.
- Unblocks all threads that are blocked on the phase synchronization point.

The end of the completion step strongly happens before the returns from all calls that were unblocked by the completion step. For specializations that do not have the default value of the `CompletionFunction` template parameter, the behavior is undefined if any of the barrier object's member functions other than `wait` are called while the completion step is in progress.

Concurrent invocations of the member functions of `barrier`, other than its destructor, do not introduce data races. The member functions `arrive` and `arrive_and_drop` execute atomically.

`CompletionFunction` shall meet the Cpp17MoveConstructible (Table 28) and Cpp17Destructible (Table 32) requirements. `std::is_nothrow_invocable_v<CompletionFunction&>` shall be true.

The default value of the CompletionFunction template parameter is an unspecified type, such that, in addition to satisfying the requirements of CompletionFunction, it meets the Cpp17DefaultConstructible requirements (Table 27) and completion() has no effects.

barrier::arrival_token is an unspecified type, such that it meets the Cpp17MoveConstructible (Table 28), Cpp17MoveAssignable (Table 30), and Cpp17Destructible (Table 32) requirements.

Public Types

```
template<>
using arrival_token = bool
```

Public Functions

constexpr barrier (*std*::ptrdiff_t *expected*, OnCompletion *completion* = OnCompletion())

Preconditions: *expected* ≥ 0 is true and *expected* $\leq \text{max}()$ is true.

Effects: Sets both the initial expected count for each barrier phase and the current expected count for the first phase to *expected*. Initializes completion with *std::move(f)*. Starts the first phase. [Note: If *expected* is 0 this object can only be destroyed.- end note]

Exceptions

-

arrival_token **arrive** (*std*::ptrdiff_t *update* = 1)

Preconditions: *update* > 0 is true, and *update* is less than or equal to the expected count for the current barrier phase.

Effects: Constructs an object of type arrival_token that is associated with the phase synchronization point for the current phase. Then, decrements the expected count by *update*.

Synchronization: The call to arrive strongly happens before the start of the phase completion step for the current phase.

Return : The constructed arrival_token object.

Exceptions

-

void **wait** (arrival_token &&*old_phase*) **const**

Preconditions: arrival is associated with the phase synchronization point for the current phase or the immediately preceding phase of the same barrier object.

Effects: Blocks at the synchronization point associated with HPX_MOVE(arrival) until the phase completion step of the synchronization point's phase is run. [Note: If arrival is associated with the synchronization point for a previous phase, the call returns immediately. - end note]

Exceptions

-

void **arrive_and_wait** ()

Effects: Equivalent to: wait(arrive()).

```
void arrive_and_drop()
```

Preconditions: The expected count for the current barrier phase is greater than zero.

Effects: Decrements the initial expected count for all subsequent phases by one. Then decrements the expected count for the current phase by one.

Synchronization: The call to `arrive_and_drop` strongly happens before the start of the phase completion step for the current phase.

Exceptions

-

Public Static Functions

```
static constexpr std::ptrdiff_t() hpx::barrier::max()
```

Private Types

```
template<>
using mutex_type = hpx::spinlock
```

Private Members

```
mutex_type mtx_
hpx::lcos::local::detail::condition_variable cond_
std::ptrdiff_t expected_
std::ptrdiff_t arrived_
OnCompletion completion_
bool phase_
```

hpx/synchronization/counting_semaphore.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

TypeDefs

```
template<std::ptrdiff_t Least.MaxValue = PTRDIFF_MAX>
using counting_semaphore = detail::counting_semaphore<Least.MaxValue>
using binary_semaphore = detail::binary_semaphore<>

template<typename Mutex = hpx::spinlock, int N = 0>
class counting_semaphore_var : private hpx::detail::counting_semaphore<PTRDIFF_MAX, hpx::spinlock>
```

Public Functions

```

counting_semaphore_var (std::ptrdiff_t value = N)

counting_semaphore_var (counting_semaphore_var const&)

counting_semaphore_var &operator= (counting_semaphore_var const&)

void wait (std::ptrdiff_t count = 1)

bool try_wait (std::ptrdiff_t count = 1)

void signal (std::ptrdiff_t count = 1)
    Signal the semaphore.

std::ptrdiff_t signal_all ()

```

Private Types

```

template<>
using mutex_type = Mutex

namespace lcos

namespace local

```

Typedefs

```
typedef hpx::sliding_semaphore instead
```

hpx/synchronization/event.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```

namespace lcos

namespace local

```

class event

#include <event.hpp> Event semaphores can be used for synchronizing multiple threads that need to wait for an event to occur. When the event occurs, all threads waiting for the event are woken up.

Public Functions

```
event ()  
    Construct a new event semaphore.  
  
bool occurred ()  
    Check if the event has occurred.  
  
void wait ()  
    Wait for the event to occur.  
  
void set ()  
    Release all threads waiting on this semaphore.  
  
void reset ()  
    Reset the event.
```

Private Types

```
typedef hpx::spinlock mutex_type
```

Private Functions

```
void wait_locked (std::unique_lock<mutex_type> &l)  
void set_locked (std::unique_lock<mutex_type> l)
```

Private Members

```
mutex_type mtx_  
    This mutex protects the queue.  
local::detail::condition_variable cond_  
std::atomic<bool> event_
```

hpx/synchronization/latch.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
class latch  
#include <latch.hpp> Latches are a thread coordination mechanism that allow one or more threads to  
block until an operation is completed. An individual latch is a singleuse object; once the operation has  
been completed, the latch cannot be reused.  
Subclassed by hpx::lcos::local::latch
```

Public Functions

HPX_NON_COPYABLE (*latch*)

latch (*std*::ptrdiff_t *count*)

Initialize the latch

Requires: count ≥ 0 . Synchronization: None Postconditions: counter_ == count.

~latch()

Requires: No threads are blocked at the synchronization point.

Note May be called even if some threads have not yet returned from *wait()* or *count_down_and_wait()*, provided that counter_ is 0.

Note The destructor might not return until all threads have exited *wait()* or *count_down_and_wait()*.

Note It is the caller's responsibility to ensure that no other thread enters *wait()* after one thread has called the destructor. This may require additional coordination.

void count_down (*std*::ptrdiff_t *update*)

Decrements counter_ by n. Does not block.

Requires: counter_ $\geq n$ and $n \geq 0$.

Synchronization: Synchronizes with all calls that block on this latch and with all try_wait calls on this latch that return true .

Exceptions

- Nothing.:

bool try_wait() const

Returns: With very low probability false. Otherwise counter == 0.

void wait() const

If counter_ is 0, returns immediately. Otherwise, blocks the calling thread at the synchronization point until counter_ reaches 0.

Exceptions

- Nothing.:

void arrive_and_wait (*std*::ptrdiff_t *update* = 1)

Effects: Equivalent to: count_down(update); *wait()*;

Public Static Functions

static constexpr std::ptrdiff_t() hpx::latch::max()

Returns: The maximum value of counter that the implementation supports.

Protected Types

```
using mutex_type = hpx::spinlock
```

Protected Attributes

```
util::cache_line_data<mutex_type> mtx_
util::cache_line_data<hpx::lcos::local::detail::condition_variable> cond_
std::atomic<std::ptrdiff_t> counter_
bool notified_

namespace lcos

namespace local

class latch : public hpx::latch
#include <latch.hpp> A latch maintains an internal counter_ that is initialized when the latch is created. Threads may block at a synchronization point waiting for counter_ to be decremented to 0. When counter_ reaches 0, all such blocked threads are released.

Calls to countdown_and_wait(), count_down(), wait(), is_ready(), count_up(), and reset() behave as atomic operations.
```

Note A `hpx::latch` is not an LCO in the sense that it has no global id and it can't be triggered using the action (parcel) mechanism. Use `hpx::distributed::latch` instead if this is required. It is just a low level synchronization primitive allowing to synchronize a given number of *threads*.

Public Functions

HPX_NON_COPYABLE (`latch`)

latch (`std::ptrdiff_t count`)

Initialize the latch

Requires: `count >= 0`. Synchronization: None Postconditions: `counter_ == count`.

~latch()

Requires: No threads are blocked at the synchronization point.

Note May be called even if some threads have not yet returned from `wait()` or `count_down_and_wait()`, provided that `counter_` is 0.

Note The destructor might not return until all threads have exited `wait()` or `count_down_and_wait()`.

Note It is the caller's responsibility to ensure that no other thread enters `wait()` after one thread has called the destructor. This may require additional coordination.

void count_down_and_wait ()

Decrements `counter_` by 1 . Blocks at the synchronization point until `counter_` reaches 0.

Requires: `counter_ > 0`.

Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.

Exceptions

- Nothing.:

`bool is_ready() const`

Returns: counter_ == 0. Does not block.

Exceptions

- Nothing.:

`void abort_all()`

`void count_up(std::ptrdiff_t n)`

Increments counter_ by n. Does not block.

Requires: n >= 0.

Exceptions

- Nothing.:

`void reset(std::ptrdiff_t n)`

Reset counter_ to n. Does not block.

Requires: n >= 0.

Exceptions

- Nothing.:

`bool reset_if_needed_and_count_up(std::ptrdiff_t n, std::ptrdiff_t count)`

Effects: Equivalent to: if (`is_ready()`) reset(count); count_up(n); Returns: true if the latch was reset

hpx/synchronization/recursive_mutex.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Typedefs

`using recursive_mutex = detail::recursive_mutex_impl<>`

hpx/synchronization/sliding_semaphore.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Typedefs

```
using sliding_semaphore = sliding_semaphore_var<>
```

```
template<typename Mutex = hpx::spinlock>
class sliding_semaphore_var
```

#include <sliding_semaphore.hpp> A semaphore is a protected variable (an entity storing a value) or abstract data type (an entity grouping several variables that may or may not be numerical) which constitutes the classic method for restricting access to shared resources, such as shared memory, in a multiprogramming environment. Semaphores exist in many variants, though usually the term refers to a counting semaphore, since a binary semaphore is better known as a mutex. A counting semaphore is a counter for a set of available resources, rather than a locked/unlocked flag of a single resource. It was invented by Edsger Dijkstra. Semaphores are the classic solution to preventing race conditions in the dining philosophers problem, although they do not prevent resource deadlocks.

Sliding semaphores can be used for synchronizing multiple threads as well: one thread waiting for several other threads to touch (signal) the semaphore, or several threads waiting for one other thread to touch this semaphore. The difference to a counting semaphore is that a sliding semaphore will not limit the number of threads which are allowed to proceed, but will make sure that the difference between the (arbitrary) number passed to set and wait does not exceed a given threshold.

Public Functions

```
sliding_semaphore_var (std::int64_t max_difference, std::int64_t lower_limit = 0)
```

Construct a new sliding semaphore.

Parameters

- *max_difference*: [in] The max difference between the upper limit (as set by [wait\(\)](#)) and the lower limit (as set by [signal\(\)](#)) which is allowed without suspending any thread calling [wait\(\)](#).
- *lower_limit*: [in] The initial lower limit.

```
void set_max_difference (std::int64_t max_difference, std::int64_t lower_limit = 0)
```

Set/Change the difference that will cause the semaphore to trigger.

Parameters

- *max_difference*: [in] The max difference between the upper limit (as set by [wait\(\)](#)) and the lower limit (as set by [signal\(\)](#)) which is allowed without suspending any thread calling [wait\(\)](#).
- *lower_limit*: [in] The initial lower limit.

```
void wait (std::int64_t upper_limit)
```

Wait for the semaphore to be signaled.

Parameters

- `upper_limit`: [in] The new upper limit. The calling thread will be suspended if the difference between this value and the largest `lower_limit` which was set by `signal()` is larger than the `max_difference`.

```
bool try_wait (std::int64_t upper_limit = 1)
Try to wait for the semaphore to be signaled.
```

Return The function returns true if the calling thread would not block if it was calling `wait()`.

Parameters

- `upper_limit`: [in] The new upper limit. The calling thread will be suspended if the difference between this value and the largest `lower_limit` which was set by `signal()` is larger than the `max_difference`.

```
void signal (std::int64_t lower_limit)
Signal the semaphore.
```

Parameters

- `lower_limit`: [in] The new lower limit. This will update the current lower limit of this semaphore. It will also re-schedule all suspended threads for which their associated upper limit is not larger than the lower limit plus the `max_difference`.

```
std::int64_t signal_all ()
```

Private Types

```
template<>
using mutex_type = Mutex
```

Private Members

```
mutex_type mtx_
lcos::local::detail::sliding_semaphore sem_
```

thread_pool_util

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/thread_pool_util/thread_pool_suspension_helpers.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace threads
```

Functions

`hpx::future<void> resume_processing_unit (thread_pool_base &pool, std::size_t virt_core)`
Resumes the given processing unit. When the processing unit has been resumed the returned future will be ready.

Note Can only be called from an HPX thread. Use `resume_processing_unit_cb` or to resume the processing unit from outside HPX. Requires that the pool has `threads::policies::enable_elasticity` set.

Return A `future<void>` which is ready when the given processing unit has been resumed.

Parameters

- `virt_core`: [in] The processing unit on the pool to be resumed. The processing units are indexed starting from 0.

`void resume_processing_unit_cb (thread_pool_base &pool, hpx::function<void> void > callback, std::size_t virt_core, error_code &ec = throws)`Resumes the given processing unit. Takes a callback as a parameter which will be called when the processing unit has been resumed.

Note Requires that the pool has `threads::policies::enable_elasticity` set.

Parameters

- `callback`: [in] Callback which is called when the processing unit has been suspended.
- `virt_core`: [in] The processing unit to resume.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`hpx::future<void> suspend_processing_unit (thread_pool_base &pool, std::size_t virt_core)`

Suspends the given processing unit. When the processing unit has been suspended the returned future will be ready.

Note Can only be called from an HPX thread. Use `suspend_processing_unit_cb` or to suspend the processing unit from outside HPX. Requires that the pool has `threads::policies::enable_elasticity` set.

Return A `future<void>` which is ready when the given processing unit has been suspended.

Parameters

- `virt_core`: [in] The processing unit on the pool to be suspended. The processing units are indexed starting from 0.

Exceptions

- `hpx::exception`: if called from outside the HPX runtime.

`void suspend_processing_unit_cb (hpx::function<void> void > callback, thread_pool_base &pool, std::size_t virt_core, error_code &ec = throws)`

Suspends the given processing unit. Takes a callback as a parameter which will be called when the processing unit has been suspended.

Note Requires that the pool has `threads::policies::enable_elasticity` set.

Parameters

- `callback`: [in] Callback which is called when the processing unit has been suspended.
- `virt_core`: [in] The processing unit to suspend.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`hpx::future<void> resume_pool (thread_pool_base &pool)`

Resumes the thread pool. When the all OS threads on the thread pool have been resumed the returned future will be ready.

Note Can only be called from an HPX thread. Use resume_cb or resume_direct to suspend the pool from outside HPX.

Return A `future<void>` which is ready when the thread pool has been resumed.

Exceptions

- `hpx::exception`: if called from outside the HPX runtime.

`void resume_pool_cb (thread_pool_base &pool, hpx::function<void> void`

`> callback, error_code &ec = throws`Resumes the thread pool. Takes a callback as a parameter which will be called when all OS threads on the thread pool have been resumed.

Parameters

- `callback`: [in] called when the thread pool has been resumed.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`hpx::future<void> suspend_pool (thread_pool_base &pool)`

Suspends the thread pool. When the all OS threads on the thread pool have been suspended the returned future will be ready.

Note Can only be called from an HPX thread. Use suspend_cb or suspend_direct to suspend the pool from outside HPX. A thread pool cannot be suspended from an HPX thread running on the pool itself.

Return A `future<void>` which is ready when the thread pool has been suspended.

Exceptions

- `hpx::exception`: if called from outside the HPX runtime.

`void suspend_pool_cb (thread_pool_base &pool, hpx::function<void> void`

`> callback, error_code &ec = throws`Suspends the thread pool. Takes a callback as a parameter which will be called when all OS threads on the thread pool have been suspended.

Note A thread pool cannot be suspended from an HPX thread running on the pool itself.

Parameters

- `callback`: [in] called when the thread pool has been suspended.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

Exceptions

- `hpx::exception`: if called from an HPX thread which is running on the pool itself.

threading_base

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/threading_base/annotated_function.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

```
template<typename F>
constexpr F &&annotated_function(F &&f, char const* = nullptr)
    Returns a function annotated with the given annotation.
```

Annotating includes setting the thread description per thread id.

Parameters

- function:

```
template<typename F>
constexpr F &&annotated_function(F &&f, std::string const&)

namespace util
```

Functions

```
template<typename F>
constexpr decltype(auto) annotated_function(F &&f, char const *name = nullptr)

template<typename F>
constexpr decltype(auto) annotated_function(F &&f, std::string const &name)
```

hpx/threading_base/print.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/threading_base/register_thread.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace threads`

Functions

```
template<typename F>
thread_function_type make_thread_function(F &&f)

template<typename F>
thread_function_type make_thread_function_nullary(F &&f)

threads::thread_id_ref_type register_thread(threads::thread_init_data &data,
                                             threads::thread_pool_base *pool, error_code
                                             &ec = throws)
```

Create a new *thread* using the given data.

Return This function will return the internal id of the newly created HPX-thread.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *data*: [in] The data to use for creating the thread.
- *pool*: [in] The thread pool to use for launching the work.
- *ec*: [in,out] This represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Exceptions

- *invalid_status*: if the runtime system has not been started yet.

```
threads::thread_id_ref_type register_thread(threads::thread_init_data &data, error_code
                                            &ec = throws)
```

Create a new *thread* using the given data on the same thread pool as the calling thread, or on the default thread pool if not on an HPX thread.

Return This function will return the internal id of the newly created HPX-thread.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *data*: [in] The data to use for creating the thread.
- *ec*: [in,out] This represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Exceptions

- *invalid_status*: if the runtime system has not been started yet.

```
thread_id_ref_type register_work(threads::thread_init_data &data, threads::thread_pool_base
                                 *pool, error_code &ec = throws)
```

Create a new work item using the given data.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *data*: [in] The data to use for creating the thread.
- *pool*: [in] The thread pool to use for launching the work.
- *ec*: [in,out] This represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Exceptions

- *invalid_status*: if the runtime system has not been started yet.

`thread_id_ref_type register_work (threads::thread_init_data &data, error_code &ec = throws)`

Create a new work item using the given data on the same thread pool as the calling thread, or on the default thread pool if not on an HPX thread.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `data`: [in] The data to use for creating the thread.
- `ec`: [in,out] This represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

Exceptions

- `invalid_status`: if the runtime system has not been started yet.

`hpx/threading_base/thread_data.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace threads`

Functions

`thread_data *get_self_id_data()`

The function `get_self_id_data` returns the data of the HPX thread id associated with the current thread (or nullptr if the current thread is not a HPX thread).

`thread_data *get_thread_id_data (thread_id_ref_type const &tid)`

`thread_data *get_thread_id_data (thread_id_type const &tid)`

`thread_self &get_self ()`

The function `get_self` returns a reference to the (OS thread specific) self reference to the current HPX thread.

`thread_self *get_self_ptr()`

The function `get_self_ptr` returns a pointer to the (OS thread specific) self reference to the current HPX thread.

`thread_self_impl_type *get_ctx_ptr()`

The function `get_ctx_ptr` returns a pointer to the internal data associated with each coroutine.

`thread_self *get_self_ptr_checked (error_code &ec = throws)`

The function `get_self_ptr_checked` returns a pointer to the (OS thread specific) self reference to the current HPX thread.

`thread_id_type get_self_id ()`

The function `get_self_id` returns the HPX thread id of the current thread (or zero if the current thread is not a HPX thread).

`thread_id_type get_parent_id ()`

The function `get_parent_id` returns the HPX thread id of the current thread's parent (or zero if the current thread is not a HPX thread).

Note This function will return a meaningful value only if the code was compiled with `HPX_HAVE_THREAD_PARENT_REFERENCE` being defined.

`std::size_t get_parent_phase()`

The function `get_parent_phase` returns the HPX phase of the current thread's parent (or zero if the current thread is not a HPX thread).

Note This function will return a meaningful value only if the code was compiled with `HPX_HAVE_THREAD_PARENT_REFERENCE` being defined.

`std::ptrdiff_t get_self_stacksize()`

The function `get_self_stacksize` returns the stack size of the current thread (or zero if the current thread is not a HPX thread).

`thread_stacksize get_self_stacksize_enum()`

The function `get_self_stacksize_enum` returns the stack size of the /.

`std::uint32_t get_parent_locality_id()`

The function `get_parent_locality_id` returns the id of the locality of the current thread's parent (or zero if the current thread is not a HPX thread).

Note This function will return a meaningful value only if the code was compiled with `HPX_HAVE_THREAD_PARENT_REFERENCE` being defined.

`std::uint64_t get_self_component_id()`

The function `get_self_component_id` returns the lva of the component the current thread is acting on

Note This function will return a meaningful value only if the code was compiled with `HPX_HAVE_THREAD_TARGET_ADDRESS` being defined.

`class thread_data : public thread_data_reference_counting`

`#include <thread_data.hpp>` A `thread` is the representation of a ParalleX thread. It's a first class object in ParalleX. In our implementation this is a user level thread running on top of one of the OS threads spawned by the `thread-manager`.

A `thread` encapsulates:

- A thread status word (see the functions `thread::get_state` and `thread::set_state`)
- A function to execute (the `thread` function)
- A frame (in this implementation this is a block of memory used as the threads stack)
- A block of registers (not implemented yet)

Generally, `threads` are not created or executed directly. All functionality related to the management of `threads` is implemented by the `thread-manager`.

Public Types

`using spinlock_pool = util::spinlock_pool<thread_data>`

Public Functions

```
thread_data (thread_data const&)
thread_data (thread_data&&)
thread_data &operator= (thread_data const&)
thread_data &operator= (thread_data&&)

thread_state get_state (std::memory_order order = std::memory_order_acquire) const
The get_state function queries the state of this thread instance.
```

Return This function returns the current state of this thread. It will return one of the values as defined by the *thread_state* enumeration.

Note This function will be seldom used directly. Most of the time the state of a thread will be retrieved by using the function *threadmanager::get_state*.

```
thread_state set_state (thread_schedule_state state, thread_restart_state state_ex =
                      thread_restart_state::unknown, std::memory_order load_order =
                      std::memory_order_acquire, std::memory_order exchange_order =
                      std::memory_order_seq_cst)
```

The *set_state* function changes the state of this thread instance.

Note This function will be seldom used directly. Most of the time the state of a thread will have to be changed using the *threadmanager*. Moreover, changing the thread state using this function does not change its scheduling status. It only sets the thread's status word. To change the thread's scheduling status *threadmanager::set_state* should be used.

Parameters

- *newstate*: [in] The new state to be set for the thread.

```
bool set_state_tagged (thread_schedule_state newstate, thread_state &prev_state,
                      thread_state &new_tagged_state, std::memory_order exchange_order =
                      std::memory_order_seq_cst)
```

```
bool restore_state (thread_state new_state, thread_state old_state, std::memory_order load_order =
                    std::memory_order_relaxed, std::memory_order load_exchange =
                    std::memory_order_seq_cst)
```

The *restore_state* function changes the state of this thread instance depending on its current state. It will change the state atomically only if the current state is still the same as passed as the second parameter. Otherwise it won't touch the thread state of this instance.

Note This function will be seldom used directly. Most of the time the state of a thread will have to be changed using the *threadmanager*. Moreover, changing the thread state using this function does not change its scheduling status. It only sets the thread's status word. To change the thread's scheduling status *threadmanager::set_state* should be used.

Return This function returns *true* if the state has been changed successfully

Parameters

- *newstate*: [in] The new state to be set for the thread.
- *oldstate*: [in] The old state of the thread which still has to be the current state.

```
bool restore_state (thread_schedule_state new_state, thread_restart_state state_ex,
                   thread_state old_state, std::memory_order load_exchange =
                   std::memory_order_seq_cst)
```

```

constexpr std::uint64_t get_component_id() const
    Return the id of the component this thread is running in.

util::thread_description get_description() const
util::thread_description set_description(util::thread_description)
util::thread_description get_lco_description() const
util::thread_description set_lco_description(util::thread_description)

constexpr std::uint32_t get_parent_locality_id() const
    Return the locality of the parent thread.

constexpr thread_id_type get_parent_thread_id() const
    Return the thread id of the parent thread.

constexpr std::size_t get_parent_thread_phase() const
    Return the phase of the parent thread.

constexpr util::backtrace const *get_backtrace() const
util::backtrace const *set_backtrace(util::backtrace const*)

constexpr thread_priority get_priority() const
void set_priority(thread_priority priority)

bool interruption_requested() const
bool interruption_enabled() const
bool set_interruption_enabled(bool enable)
void interrupt(bool flag = true)
bool interruption_point(bool throw_on_interrupt = true)

bool add_thread_exit_callback(function<void>
    > const &f

void run_thread_exit_callbacks()
void free_thread_exit_callbacks()

bool is_stackless() const
void destroy_thread()

policies::scheduler_base *get_scheduler_base() const
std::size_t get_last_worker_thread_num() const
void set_last_worker_thread_num(std::size_t last_worker_thread_num)

std::ptrdiff_t get_stack_size() const
thread_stacksize get_stack_size_enum() const

template<typename ThreadQueue>
ThreadQueue &get_queue()

```

```
coroutine_type::result_type operator() (hpx::execution_base::this_thread::detail::agent_storage  
*agent_storage)  
Execute the thread function.
```

Return This function returns the thread state the thread should be scheduled from this point on.
The thread manager will use the returned value to set the thread's scheduling status.

```
virtual thread_id_type get_thread_id() const  
virtual std::size_t get_thread_phase() const  
virtual std::size_t get_thread_data() const = 0  
virtual std::size_t set_thread_data(std::size_t data) = 0  
virtual void init() = 0  
virtual void rebind(thread_init_data &init_data) = 0  
thread_data(thread_init_data &init_data, void *queue, std::ptrdiff_t stacksize, bool  
is_stackless = false, thread_id_addr addr = thread_id_addr::yes)  
virtual ~thread_data()  
virtual void destroy() = 0
```

Protected Functions

```
thread_restart_state set_state_ex(thread_restart_state new_state)  
The set_state function changes the extended state of this thread instance.
```

Note This function will be seldom used directly. Most of the time the state of a thread will have to be changed using the threadmanager.

Parameters

- newstate: [in] The new extended state to be set for the thread.

```
void rebind_base(thread_init_data &init_data)
```

Private Members

```
std::atomic<thread_state> current_state_  
thread_priority priority_  
bool requested_interrupt_  
bool enabled_interrupt_  
bool ran_exit_funcs_  
const bool is_stackless_  
std::forward_list<hpx::function<void ()>> exit_funcs_  
policies::scheduler_base *scheduler_base_  
std::size_t last_worker_thread_num_  
std::ptrdiff_t stacksize_
```

```
thread_stacksize stacksize_enum_
void *queue_
```

hpx/threading_base/thread_description.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace threads
```

Functions

```
util::thread_description get_thread_description(thread_id_type const &id, error_code
&ec = throws)
```

The function `get_thread_description` is part of the thread related API allows to query the description of one of the threads known to the thread-manager.

Return This function returns the description of the thread referenced by the `id` parameter. If the thread is not known to the thread-manager the return value will be the string “<unknown>”.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `id`: [in] The thread id of the thread being queried.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
util::thread_description set_thread_description(thread_id_type const &id,
util::thread_description const &desc
= util::thread_description(), error_code
&ec = throws)
```

```
util::thread_description get_thread_lco_description(thread_id_type const &id, error_code &ec = throws)
```

```
util::thread_description set_thread_lco_description(thread_id_type const &id,
util::thread_description const &desc = util::thread_description(),
error_code &ec = throws)
```

```
namespace util
```

Functions

```
std::ostream &operator<<(std::ostream&, thread_description const&)
```

```
std::string as_string(thread_description const &desc)
```

```
struct thread_description
```

Public Types

```
enum data_type
    Values:
        data_type_description = 0
        data_type_address = 1
```

Public Functions

```
thread_description()
constexpr thread_description(char const*)
template<typename F, typename = typename std::enable_if<!std::is_same<F, thread_description>::value && !traits>
constexpr thread_description(F const&, char const* = nullptr)
template<typename Action, typename = typename std::enable_if<traits::is_action<Action>::value>::type>
constexpr thread_description(Action, char const* = nullptr)
constexpr data_type kind() const
constexpr char const *get_description() const
constexpr std::size_t get_address() const
constexpr operator bool() const
constexpr bool valid() const
```

Private Functions

```
void init_from_alternative_name(char const *altname)
```

hpx/threading_base/thread_helpers.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace this_thread
```

Functions

```
threads::thread_restart_state suspend(threads::thread_schedule_state
                                      state,           threads::thread_id_type      id,
                                      util::thread_description const &description = util::thread_description("this_thread::suspend"),
                                      error_code &ec = throws)
```

The function *suspend* will return control to the thread manager (suspends the current thread). It sets the new state of this thread to the thread state passed as the parameter.

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

```
threads::thread_restart_state suspend(threads::thread_schedule_state state = threads::thread_schedule_state::pending,
                                      util::thread_description const &description = util::thread_description("this_thread::suspend"),
                                      error_code &ec = throws)
```

The function `suspend` will return control to the thread manager (suspends the current thread). It sets the new state of this thread to the thread state passed as the parameter.

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

```
threads::thread_restart_state suspend(hpx::chrono::steady_time_point const &abs_time, threads::thread_id_type id, util::thread_description const &description = util::thread_description("this_thread::suspend"), error_code &ec = throws)
```

The function `suspend` will return control to the thread manager (suspends the current thread). It sets the new state of this thread to `suspended` and schedules a wakeup for this threads at the given time.

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

```
threads::thread_restart_state suspend(hpx::chrono::steady_time_point const &abs_time, util::thread_description const &description = util::thread_description("this_thread::suspend"), error_code &ec = throws)
```

The function `suspend` will return control to the thread manager (suspends the current thread). It sets the new state of this thread to `suspended` and schedules a wakeup for this threads at the given time.

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of

hpx::yield_aborted if it is signaled with *wait_aborted*. If called outside of a HPX-thread, this function will throw an *hpx::exception* with an error code of *hpx::null_thread_id*. If this function is called while the thread-manager is not running, it will throw an *hpx::exception* with an error code of *hpx::invalid_status*.

```
threads::thread_restart_state suspend(hpx::chrono::steady_duration const &rel_time,
                                      util::thread_description const &description =
                                      util::thread_description("this_thread::suspend"),
                                      error_code &ec = throws)
```

The function *suspend* will return control to the thread manager (suspends the current thread). It sets the new state of this thread to *suspended* and schedules a wakeup for this threads after the given duration.

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set *ec* to an appropriate value when an error occurs. Otherwise, this function will throw an *hpx::exception* with an error code of *hpx::yield_aborted* if it is signaled with *wait_aborted*. If called outside of a HPX-thread, this function will throw an *hpx::exception* with an error code of *hpx::null_thread_id*. If this function is called while the thread-manager is not running, it will throw an *hpx::exception* with an error code of *hpx::invalid_status*.

```
threads::thread_restart_state suspend(hpx::chrono::steady_duration const &rel_time,
                                      threads::thread_id_type const &id,
                                      util::thread_description const &description =
                                      util::thread_description("this_thread::suspend"),
                                      error_code &ec = throws)
```

The function *suspend* will return control to the thread manager (suspends the current thread). It sets the new state of this thread to *suspended* and schedules a wakeup for this threads after the given duration.

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set *ec* to an appropriate value when an error occurs. Otherwise, this function will throw an *hpx::exception* with an error code of *hpx::yield_aborted* if it is signaled with *wait_aborted*. If called outside of a HPX-thread, this function will throw an *hpx::exception* with an error code of *hpx::null_thread_id*. If this function is called while the thread-manager is not running, it will throw an *hpx::exception* with an error code of *hpx::invalid_status*.

```
threads::thread_restart_state suspend(std::uint64_t ms, util::thread_description const &description =
                                         util::thread_description("this_thread::suspend"),
                                         error_code &ec = throws)
```

The function *suspend* will return control to the thread manager (suspends the current thread). It sets the new state of this thread to *suspended* and schedules a wakeup for this threads after the given time (specified in milliseconds).

Note Must be called from within a HPX-thread.

Exceptions

- If: `&ec != &throws`, never throws, but will set *ec* to an appropriate value when an error occurs. Otherwise, this function will throw an *hpx::exception* with an error code of *hpx::yield_aborted* if it is signaled with *wait_aborted*. If called outside of a HPX-thread, this function will throw an *hpx::exception* with an error code of *hpx::null_thread_id*. If this function is called while the thread-manager is not running, it will throw an *hpx::exception* with an

error code of `hpx::invalid_status`.

`threads::thread_pool_base *get_pool (error_code &ec = throws)`
 Returns a pointer to the pool that was used to run the current thread

Exceptions

- If: `&ec != &throws`, never throws, but will set `ec` to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

namespace threads

Functions

`thread_state set_thread_state (thread_id_type const &id, thread_schedule_state state = thread_schedule_state::pending, thread_restart_state stateex = thread_restart_state::signaled, thread_priority priority = thread_priority::normal, bool retry_on_active = true, hpx::error_code &ec = throws)`

Set the thread state of the *thread* referenced by the thread_id *id*.

Note If the thread referenced by the parameter *id* is in `thread_state`::*active* state this function schedules a new thread which will set the state of the thread as soon as its not active anymore. The function returns `thread_state`::*active* in this case.

Return This function returns the previous state of the thread referenced by the *id* parameter. It will return one of the values as defined by the `thread_state` enumeration. If the thread is not known to the thread-manager the return value will be `thread_state`::*unknown*.

Note As long as *ec* is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of `hpx::exception`.

Parameters

- *id*: [in] The thread id of the thread the state should be modified for.
- *state*: [in] The new state to be set for the thread referenced by the *id* parameter.
- *stateex*: [in] The new extended state to be set for the thread referenced by the *id* parameter.
- *priority*:
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`thread_id_ref_type set_thread_state (thread_id_type const &id, hpx::chrono::steady_time_point &abs_time, std::atomic<bool> *started, thread_schedule_state state = thread_schedule_state::pending, thread_restart_state stateex = thread_restart_state::timeout, thread_priority priority = thread_priority::normal, bool retry_on_active = true, error_code &ec = throws)`

Set the thread state of the *thread* referenced by the thread_id *id*.

Set a timer to set the state of the given *thread* to the given new value after it expired (at the given time)

Return

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread the state should be modified for.
- *abs_time*: [in] Absolute point in time for the new thread to be run
- *started*: [in,out] A helper variable allowing to track the state of the timer helper thread
- *state*: [in] The new state to be set for the thread referenced by the *id* parameter.
- *stateex*: [in] The new extended state to be set for the thread referenced by the *id* parameter.
- *priority*:
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
thread_id_ref_type set_thread_state (thread_id_type const &id,  
                                     hpx::chrono::steady_time_point const  
                                     &abs_time, thread_schedule_state state =  
                                     thread_schedule_state::pending, thread_restart_state  
                                     stateex = thread_restart_state::timeout, thread_priority  
                                     priority = thread_priority::normal, bool retry_on_active  
                                     = true, error_code& = throws)
```

```
thread_id_ref_type set_thread_state (thread_id_type const &id,  
                                     hpx::chrono::steady_duration const  
                                     &rel_time, thread_schedule_state state =  
                                     thread_schedule_state::pending, thread_restart_state  
                                     stateex = thread_restart_state::timeout, thread_priority  
                                     priority = thread_priority::normal, bool retry_on_active  
                                     = true, error_code &ec = throws)
```

Set the thread state of the *thread* referenced by the *thread_id* *id*.

Set a timer to set the state of the given *thread* to the given new value after it expired (after the given duration)

Return

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread the state should be modified for.
- *rel_time*: [in] Time duration after which the new thread should be run
- *state*: [in] The new state to be set for the thread referenced by the *id* parameter.
- *stateex*: [in] The new extended state to be set for the thread referenced by the *id* parameter.
- *priority*:
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
thread_state get_thread_state (thread_id_type const &id, error_code &ec = throws)
```

The function *get_thread_backtrace* is part of the thread related API allows to query the currently stored thread back trace (which is captured during thread suspension).

Return This function returns the currently captured stack back trace of the thread referenced by the *id* parameter. If the thread is not known to the thread-manager the return value will be the zero.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*. The function *get_thread_state* is part of the thread related API. It queries the state of one of the threads known to the thread-manager.

Return This function returns the thread state of the thread referenced by the *id* parameter. If the thread is not known to the thread-manager the return value will be *terminated*.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread being queried.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Parameters

- *id*: [in] The thread id of the thread the state should be modified for.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

`std::size_t get_thread_phase(thread_id_type const &id, error_code &ec = throws)`

The function *get_thread_phase* is part of the thread related API. It queries the phase of one of the threads known to the thread-manager.

Return This function returns the thread phase of the thread referenced by the *id* parameter. If the thread is not known to the thread-manager the return value will be ~0.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread the phase should be modified for.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

`bool get_thread_interruption_enabled(thread_id_type const &id, error_code &ec = throws)`

Returns whether the given thread can be interrupted at this point.

Return This function returns *true* if the given thread can be interrupted at this point in time. It will return *false* otherwise.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread which should be queried.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

`bool set_thread_interruption_enabled(thread_id_type const &id, bool enable, error_code &ec = throws)`

Set whether the given thread can be interrupted at this point.

Return This function returns the previous value of whether the given thread could have been interrupted.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread which should receive the new value.
- *enable*: [in] This value will determine the new interruption enabled status for the given thread.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool get_thread_interruption_requested(thread_id_type const &id, error_code &ec  
= throws)
```

Returns whether the given thread has been flagged for interruption.

Return This function returns *true* if the given thread was flagged for interruption. It will return *false* otherwise.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread which should be queried.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
void interrupt_thread(thread_id_type const &id, bool flag, error_code &ec = throws)
```

Flag the given thread for interruption.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread which should be interrupted.
- *flag*: [in] The flag encodes whether the thread should be interrupted (if it is *true*, or ‘uninterrupted’ if it is *false*).
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
void interrupt_thread(thread_id_type const &id, error_code &ec = throws)
```

```
void interruption_point(thread_id_type const &id, error_code &ec = throws)
```

Interrupt the current thread at this point if it was canceled. This will throw a *thread_interrupted* exception, which will cancel the thread.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread which should be interrupted.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
threads::thread_priority get_thread_priority(thread_id_type const &id, error_code &ec =  
throws)
```

Return priority of the given thread

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread whose priority is queried.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
std::ptrdiff_t get_stack_size(thread_id_type const &id, error_code &ec = throws)
```

Return stack size of the given thread

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The thread id of the thread whose priority is queried.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

`threads::thread_pool_base *get_pool (thread_id_type const &id, error_code &ec = throws)`

Returns a pointer to the pool that was used to run the current thread

Exceptions

- If: *&ec != &throws*, never throws, but will set *ec* to an appropriate value when an error occurs. Otherwise, this function will throw an *hpx::exception* with an error code of *hpx::yield_aborted* if it is signaled with *wait_aborted*. If called outside of a HPX-thread, this function will throw an *hpx::exception* with an error code of *hpx::null_thread_id*. If this function is called while the thread-manager is not running, it will throw an *hpx::exception* with an error code of *hpx::invalid_status*.

hpx/threading_base/thread_num_tss.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

`std::size_t get_worker_thread_num()`

Return the number of the current OS-thread running in the runtime instance the current HPX-thread is executed with.

This function returns the zero based index of the OS-thread which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the overall number of OS-threads executed (as returned by *get_os_thread_count()*).

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

`std::size_t get_worker_thread_num(error_code &ec)`

Return the number of the current OS-thread running in the runtime instance the current HPX-thread is executed with.

This function returns the zero based index of the OS-thread which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the overall number of OS-threads executed (as returned by *get_os_thread_count()*). It will return -1 if the current thread is not a known thread or if the runtime is not in running state.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

Parameters

- *ec*: [in,out] this represents the error status on exit.

`std::size_t get_local_worker_thread_num()`

Return the number of the current OS-thread running in the current thread pool the current HPX-thread is executed with.

This function returns the zero based index of the OS-thread on the current thread pool which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the number of OS-threads executed on the current thread pool. It will return -1 if the current thread is not a known thread or if the runtime is not in running state.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

`std::size_t get_local_worker_thread_num(error_code &ec)`

Return the number of the current OS-thread running in the current thread pool the current HPX-thread is executed with.

This function returns the zero based index of the OS-thread on the current thread pool which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the number of OS-threads executed on the current thread pool. It will return -1 if the current thread is not a known thread or if the runtime is not in running state.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

Parameters

- `ec`: [in,out] this represents the error status on exit.

`std::size_t get_thread_pool_num()`

Return the number of the current thread pool the current HPX-thread is executed with.

This function returns the zero based index of the thread pool which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the number of thread pools started by the runtime. It will return -1 if the current thread pool is not a known thread pool or if the runtime is not in running state.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

`std::size_t get_thread_pool_num(error_code &ec)`

Return the number of the current thread pool the current HPX-thread is executed with.

This function returns the zero based index of the thread pool which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the number of thread pools started by the runtime. It will return -1 if the current thread pool is not a known thread pool or if the runtime is not in running state.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

Parameters

- `ec`: [in,out] this represents the error status on exit.

hpx/threading_base/thread_pool_base.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace threads

Functions

`std::ostream &operator<< (std::ostream &os, thread_pool_base const &thread_pool)`

class thread_pool_base

`#include <thread_pool_base.hpp>` The base class used to manage a pool of OS threads.

Public Functions

virtual void suspend_processing_unit_direct (std::size_t virt_core, error_code &ec = throws) = 0

Suspends the given processing unit. Blocks until the processing unit has been suspended.

Parameters

- `virt_core`: [in] The processing unit on the the pool to be suspended. The processing units are indexed starting from 0.

virtual void resume_processing_unit_direct (std::size_t virt_core, error_code &ec = throws) = 0

Resumes the given processing unit. Blocks until the processing unit has been resumed.

Parameters

- `virt_core`: [in] The processing unit on the the pool to be resumed. The processing units are indexed starting from 0.

virtual void resume_direct (error_code &ec = throws) = 0

Resumes the thread pool. Blocks until all OS threads on the thread pool have been resumed.

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

virtual void suspend_direct (error_code &ec = throws) = 0

Suspends the thread pool. Blocks until all OS threads on the thread pool have been suspended.

Note A thread pool cannot be suspended from an HPX thread running on the pool itself.

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

Exceptions

- `hpx::exception`: if called from an HPX thread which is running on the pool itself.

struct thread_pool_init_parameters

Public Functions

```
thread_pool_init_parameters(std::string const &name, std::size_t index,  
    policies::scheduler_mode mode, std::size_t  
    num_threads, std::size_t thread_offset,  
    hpx::threads::policies::callback_notifier &notifier,  
    hpx::threads::policies::detail::affinity_data  
const &affinity_data,  
hpx::threads::detail::network_background_callback_type  
const &network_background_callback =  
hpx::threads::detail::network_background_callback_type(),  
std::size_t max_background_threads =  
std::size_t(-1), std::size_t max_idle_loop_count  
= HPX_IDLE_LOOP_COUNT_MAX,  
std::size_t max_busy_loop_count =  
HPX_BUSY_LOOP_COUNT_MAX, std::size_t  
shutdown_check_count = 10)
```

Public Members

```
std::string const &name_  
std::size_t index_  
policies::scheduler_mode mode_  
std::size_t num_threads_  
std::size_t thread_offset_  
hpx::threads::policies::callback_notifier &notifier_  
hpx::threads::policies::detail::affinity_data const &affinity_data_  
hpx::threads::detail::network_background_callback_type const &network_background_callback_  
std::size_t max_background_threads_  
std::size_t max_idle_loop_count_  
std::size_t max_busy_loop_count_  
std::size_t shutdown_check_count_
```

hpx/threading_base/threading_base_fwd.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

threadmanager

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/modules/threadmanager.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace threads
```

```
class threadmanager
```

```
#include <threadmanager.hpp> The thread-manager class is the central instance of management for all (non-depleted) threads
```

Public Types

```
typedef threads::policies::callback_notifier notification_policy_type
typedef std::unique_ptr<thread_pool_base> pool_type
typedef threads::policies::scheduler_base scheduler_type
typedef std::vector<pool_type> pool_vector
```

Public Functions

```
threadmanager(hpx::util::runtime_configuration &rtecfg_, notification_policy_type
              &notifier, detail::network_background_callback_type net-
              work_background_callback = detail::network_background_callback_type())
~threadmanager()

void init()

void create_pools()

void print_pools(std::ostream&)
    FIXME move to private and add hpx:printpools cmd line option.

thread_pool_base &default_pool() const
scheduler_type &default_scheduler() const
thread_pool_base &get_pool(std::string const &pool_name) const
thread_pool_base &get_pool(pool_id_type const &pool_id) const
thread_pool_base &get_pool(size_t thread_index) const
bool pool_exists(std::string const &pool_name) const
bool pool_exists(size_t pool_index) const
```

```
thread_id_ref_type register_work (thread_init_data &data, error_code &ec = throws)
```

The function *register_work* adds a new work item to the thread manager. It doesn't immediately create a new *thread*, it just adds the task parameters (function, initial state and description) to the internal management data structures. The thread itself will be created when the number of existing threads drops below the number of threads specified by the constructors `max_count` parameter.

Parameters

- `func`: [in] The function or function object to execute as the thread's function. This must have a signature as defined by `thread_function_type`.
- `description`: [in] The value of this parameter allows to specify a description of the thread to create. This information is used for logging purposes mainly, but might be useful for debugging as well. This parameter is optional and defaults to an empty string.

```
void register_thread (thread_init_data &data, thread_id_ref_type &id, error_code &ec = throws)
```

The function *register_thread* adds a new work item to the thread manager. It creates a new *thread*, adds it to the internal management data structures, and schedules the new thread, if appropriate.

Parameters

- `func`: [in] The function or function object to execute as the thread's function. This must have a signature as defined by `thread_function_type`.
- `id`: [out] This parameter will hold the id of the created thread. This id is guaranteed to be validly initialized before the thread function is executed.
- `description`: [in] The value of this parameter allows to specify a description of the thread to create. This information is used for logging purposes mainly, but might be useful for debugging as well. This parameter is optional and defaults to an empty string.

```
bool run ()
```

Run the thread manager's work queue. This function instantiates the specified number of OS threads in each pool. All OS threads are started to execute the function *tfunc*.

Return The function returns *true* if the thread manager has been started successfully, otherwise it returns *false*.

```
void stop (bool blocking = true)
```

Forcefully stop the thread-manager.

Parameters

- `blocking`:

```
bool is_busy ()
```

```
bool is_idle ()
```

```
void wait ()
```

```
void suspend ()
```

```
void resume ()
```

```
state status () const
```

Return whether the thread manager is still running. This returns the "minimal state", i.e. the state of the least advanced thread pool.

```
std::int64_t get_thread_count (thread_schedule_state state = thread_schedule_state::unknown, thread_priority priority = thread_priority::default_, std::size_t num_thread = std::size_t(-1), bool reset = false)
    return the number of HPX-threads with the given state
```

Note This function lock the internal OS lock in the thread manager

```
std::int64_t get_idle_core_count ()

mask_type get_idle_core_mask ()

std::int64_t get_background_thread_count ()

bool enumerate_threads (hpx::function<bool> thread_id_type > const &f, thread_schedule_state state = thread_schedule_state::unknown) const

void abort_all_suspended_threads ()

bool cleanup_terminated (bool delete_all)

std::size_t get_os_thread_count () const
    Return the number of OS threads running in this thread-manager.

    This function will return correct results only if the thread-manager is running.

std::thread &get_os_thread_handle (std::size_t num_thread) const

void report_error (std::size_t num_thread, std::exception_ptr const &e)
    API functions forwarding to notification policy.

    This notifies the thread manager that the passed exception has been raised. The exception will be
    routed through the notifier and the scheduler (which will result in it being passed to the runtime
    object, which in turn will report it to the console, etc.).

mask_type get_used_processing_units () const
    Returns the mask identifying all processing units used by this thread manager.

hwloc_bitmap_ptr get_pool numa_bitmap (const std::string &pool_name) const

void set_scheduler_mode (threads::policies::scheduler_mode mode)

void add_scheduler_mode (threads::policies::scheduler_mode mode)

void add_remove_scheduler_mode (threads::policies::scheduler_mode to_add_mode, threads::policies::scheduler_mode to_remove_mode)

void remove_scheduler_mode (threads::policies::scheduler_mode mode)

void reset_thread_distribution ()

void init_tss (std::size_t global_thread_num)

void deinit_tss ()

std::int64_t get_queue_length (bool reset)

std::int64_t get_cumulative_duration (bool reset)
```

```
std::int64_t get_thread_count_unknown(bool reset)
std::int64_t get_thread_count_active(bool reset)
std::int64_t get_thread_count_pending(bool reset)
std::int64_t get_thread_count_suspended(bool reset)
std::int64_t get_thread_count_terminated(bool reset)
std::int64_t get_thread_count_staged(bool reset)
```

Private Types

```
typedef std::mutex mutex_type
```

Private Members

```
mutex_type mtx_
hpx::util::runtime_configuration &rtcfg_
std::vector<pool_id_type> threads_lookup_
pool_vector pools_
notification_policy_type &notifier_
detail::network_background_callback_type network_background_callback_
```

timed_execution

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/timed_execution/timed_execution.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/timed_execution/timed_execution_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

```
        namespace execution
```

Variables

```
hpx::parallel::execution::post_at_t post_at
hpx::parallel::execution::post_after_t post_after
hpx::parallel::execution::async_execute_at_t async_execute_at
hpx::parallel::execution::async_execute_after_t async_execute_after
hpx::parallel::execution::sync_execute_at_t sync_execute_at
hpx::parallel::execution::sync_execute_after_t sync_execute_after

struct async_execute_after_t : public hpx::functional::detail::tagFallback<async_execute_after_t>
    #include <timed_execution_fwd.hpp> Customization point of asynchronous execution agent creation supporting timed execution.
```

This asynchronously creates a single function invocation `f()` using the associated executor at the given point in time.

Return `f(ts...)`'s result through a future

Note This calls `exec.async_execute_after(rel_time, f, ts...)`, if available, otherwise it emulates timed scheduling by delaying calling `execution::async_execute()` on the underlying non-time-scheduled execution agent.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function `f`.
- `rel_time`: [in] The duration of time after which the given function should be scheduled to run.
- `f`: [in] The function which will be scheduled using the given executor.
- `ts...`: [in] Additional arguments to use to invoke `f`.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tagFallback_invoke(async_execute_after_t, Executor
                                         &&exec, hpx::chrono::steady_duration
                                         const &rel_time, F &&f, Ts&&... ts)

struct async_execute_at_t : public hpx::functional::detail::tagFallback<async_execute_at_t>
    #include <timed_execution_fwd.hpp> Customization point of asynchronous execution agent creation supporting timed execution.
```

This asynchronously creates a single function invocation `f()` using the associated executor at the given point in time.

Return `f(ts...)`'s result through a future

Note This calls `exec.async_execute_at(abs_time, f, ts...)`, if available, otherwise it emulates timed scheduling by delaying calling `execution::async_execute()` on the underlying non-time-scheduled execution agent.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function `f`.
- `abs_time`: [in] The point in time the given function should be scheduled at to run.
- `f`: [in] The function which will be scheduled using the given executor.
- `ts...`: [in] Additional arguments to use to invoke `f`.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tag_fallback_invoke(async_execute_at_t, Executor &&exec,
                                         hpx::chrono::steady_time_point
                                         const &abs_time, F &&f, Ts&&...
                                         ts)

struct post_after_t : public hpx::functional::detail::tag_fallback<post_after_t>
#include <timed_execution_fwd.hpp> Customization point of asynchronous fire & forget execution agent creation supporting timed execution.
```

This asynchronously (fire & forget) creates a single function invocation f() using the associated executor at the given point in time.

Note This calls exec.post_after(rel_time, f, ts...), if available, otherwise it emulates timed scheduling by delaying calling execution::post() on the underlying non-time-scheduled execution agent.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function *f*.
- `rel_time`: [in] The duration of time after which the given function should be scheduled to run.
- `f`: [in] The function which will be scheduled using the given executor.
- `ts...`: [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tag_fallback_invoke(post_after_t, Executor &&exec,
                                         hpx::chrono::steady_duration const
                                         &rel_time, F &&f, Ts&&... ts)

struct post_at_t : public hpx::functional::detail::tag_fallback<post_at_t>
#include <timed_execution_fwd.hpp> Customization point of asynchronous fire & forget execution agent creation supporting timed execution.
```

This asynchronously (fire & forget) creates a single function invocation f() using the associated executor at the given point in time.

Note This calls exec.post_at(abs_time, f, ts...), if available, otherwise it emulates timed scheduling by delaying calling execution::post() on the underlying non-time-scheduled execution agent.

Parameters

- `exec`: [in] The executor object to use for scheduling of the function *f*.
- `abs_time`: [in] The point in time the given function should be scheduled at to run.
- `f`: [in] The function which will be scheduled using the given executor.
- `ts...`: [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tag_fallback_invoke(post_at_t, Executor &&exec,
                                         hpx::chrono::steady_time_point
                                         const &abs_time, F &&f, Ts&&...
                                         ts)

struct sync_execute_after_t : public hpx::functional::detail::tag_fallback<sync_execute_after_t>
#include <timed_execution_fwd.hpp> Customization point of synchronous execution agent creation supporting timed execution.
```

This synchronously creates a single function invocation f() using the associated executor at the given point in time.

Return f(ts...)'s result

Note This calls exec.sync_execute_after(rel_time, f, ts...), if available, otherwise it emulates timed scheduling by delaying calling execution::sync_execute() on the underlying non-time-scheduled execution agent.

Parameters

- **exec:** [in] The executor object to use for scheduling of the function *f*.
- **rel_time:** [in] The duration of time after which the given function should be scheduled to run.
- **f:** [in] The function which will be scheduled using the given executor.
- **ts...:** [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tag_fallback_invoke(sync_execute_after_t, Executor
                                         &&exec, hpx::chrono::steady_duration
                                         const &rel_time, F &&f, Ts&&... ts)

struct sync_execute_at_t : public hpx::functional::detail::tag_fallback<sync_execute_at_t>
#include <timed_execution_fwd.hpp> Customization point of synchronous execution agent creation supporting timed execution.
```

This synchronously creates a single function invocation f() using the associated executor at the given point in time.

Return f(ts...)'s result

Note This calls exec.sync_execute_at(abs_time, f, ts...), if available, otherwise it emulates timed scheduling by delaying calling execution::sync_execute() on the underlying non-time-scheduled execution agent.

Parameters

- **exec:** [in] The executor object to use for scheduling of the function *f*.
- **abs_time:** [in] The point in time the given function should be scheduled at to run.
- **f:** [in] The function which will be scheduled using the given executor.
- **ts...:** [in] Additional arguments to use to invoke *f*.

Private Functions

```
template<typename Executor, typename F, typename ...Ts>
decltype(auto) friend tagFallback_invoke(sync_execute_at_t, Executor &&exec,
                                         hpx::chrono::steady_time_point
                                         const &abs_time, F &&f, Ts&&...
                                         ts)
```

hpx/timed_execution/timed_executors.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace execution
```

TypeDefs

```
using sequenced_timed_executor = timed_executor<hpx::execution::sequenced_executor>;
using parallel_timed_executor = timed_executor<hpx::execution::parallel_executor>;
template<typename BaseExecutor>
struct timed_executor
```

Public Types

```
typedef std::decay<BaseExecutor>::type base_executor_type
typedef hpx::traits::executor_execution_category<base_executor_type>::type execution_category
typedef hpx::traits::executor_parameters_type<base_executor_type>::type parameters_type
```

Public Functions

```
timed_executor(hpx::chrono::steady_time_point const &abs_time)
timed_executor(hpx::chrono::steady_duration const &rel_time)
template<typename Executor>
timed_executor(Executor &&exec, hpx::chrono::steady_time_point const &abs_time)
template<typename Executor>
timed_executor(Executor &&exec, hpx::chrono::steady_duration const &rel_time)
template<typename F, typename ...Ts>
hpx::util::detail::invoke_deferred_result<F, Ts...>::type sync_execute(F &&f, Ts&&...
                                         ts)
template<typename F, typename ...Ts>
```

```
hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> async_execute(F  
    &&f,  
    Ts&&...  
    ts)
```

```
template<typename F, typename ...Ts>  
void post(F &&f, Ts&&... ts)
```

Public Members

```
BaseExecutor exec_  
std::chrono::steady_clock::time_point execute_at_
```

hpx/timed_execution/traits/is_timed_executor.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace execution
```

TypeDefs

```
template<typename T>  
using is_timed_executor_t = typename is_timed_executor<T>::type
```

Variables

```
template<typename T>  
constexpr bool is_timed_executor_v = is_timed_executor<T>::value
```

topology

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/topology/cpu_mask.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/topology/topology.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace threads

Typedefs

```
using hwloc_bitmap_ptr = std::shared_ptr<hpx_hwloc_bitmap_wrapper>
```

Enums

enum hpx_hwloc_membind_policy

Please see hwloc documentation for the corresponding enums HWLOC_MEMBIND_XXX.

Values:

```
membind_default = HWLOC_MEMBIND_DEFAULT  
membind_firsttouch = HWLOC_MEMBIND_FIRSTTOUCH  
membind_bind = HWLOC_MEMBIND_BIND  
membind_interleave = HWLOC_MEMBIND_INTERLEAVE  
membind_replicate = HWLOC_MEMBIND_REPLICATE  
membind_nexttouch = HWLOC_MEMBIND_NEXTTOUCH  
membind_mixed = HWLOC_MEMBIND_MIXED  
membind_user = HWLOC_MEMBIND_MIXED + 256
```

Functions

topology &**create_topology**()

unsigned int **hardware_concurrency**()

std::size_t **get_memory_page_size**()

struct hpx_hwloc_bitmap_wrapper

Public Functions

HPX_NON_COPYABLE (*hpx_hwloc_bitmap_wrapper*)

hpx_hwloc_bitmap_wrapper()

hpx_hwloc_bitmap_wrapper (void **bmp*)

~hpx_hwloc_bitmap_wrapper()

void **reset** (hwloc_bitmap_t *bmp*)

```
operator bool() const
hwloc_bitmap_t get_bmp() const
```

Private Members

hwloc_bitmap_t **bmp_**

Friends

```
std::ostream &operator<<(std::ostream &os, hpx_hwloc_bitmap_wrapper const *bmp)
struct topology
```

Public Functions

topology()

~topology()

```
std::size_t get_socket_number(std::size_t num_thread, error_code& = throws) const
Return the Socket number of the processing unit the given thread is running on.
```

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
std::size_t get numa_node_number(std::size_t num_thread, error_code& = throws)
const
```

Return the NUMA node number of the processing unit the given thread is running on.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

mask_cref_type **get_machine_affinity_mask**(*error_code* &*ec* = *throws*) **const**

Return a bit mask where each set bit corresponds to a processing unit available to the application.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
mask_type get_service_affinity_mask(mask_cref_type used_processing_units, error_code& = throws)
const
```

Return a bit mask where each set bit corresponds to a processing unit available to the service threads in the application.

Parameters

- *used_processing_units*: [in] This is the mask of processing units which are not available for service threads.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
mask_cref_type get_socket_affinity_mask (std::size_t num_thread, error_code &ec =  
throws) const
```

Return a bit mask where each set bit corresponds to a processing unit available to the given thread inside the socket it is running on.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
mask_cref_type get numa_node_affinity_mask (std::size_t num_thread, error_code &ec =  
throws) const
```

Return a bit mask where each set bit corresponds to a processing unit available to the given thread inside the NUMA domain it is running on.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
mask_type get numa_node_affinity_mask_from_numa_node (std::size_t num_node) const
```

Return a bit mask where each set bit corresponds to a processing unit associated with the given NUMA node.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
mask_cref_type get_core_affinity_mask (std::size_t num_thread, error_code &ec =  
throws) const
```

Return a bit mask where each set bit corresponds to a processing unit available to the given thread inside the core it is running on.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
mask_cref_type get_thread_affinity_mask (std::size_t num_thread, error_code &ec =  
throws) const
```

Return a bit mask where each set bit corresponds to a processing unit available to the given thread.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
void set_thread_affinity_mask (mask_cref_type mask, error_code &ec = throws)  
const
```

Use the given bit mask to set the affinity of the given thread. Each set bit corresponds to a processing unit the thread will be allowed to run on.

Note Use this function on systems where the affinity must be set from inside the thread itself.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.


```
mask_type get_cpubind_mask (std::thread &handle, error_code &ec = throws) const
hwloc_bitmap_ptr cpuset_to_nodeset (mask_cref_type cpuset) const
    convert a cpu mask into a numa node mask in hwloc bitmap form

void write_to_log () const

void *allocate (std::size_t len) const
    This is equivalent to malloc(), except that it tries to allocate page-aligned memory from the OS.

void *allocate_membind (std::size_t len, hwloc_bitmap_ptr bitmap,
                      hpx_hwloc_membind_policy policy, int flags) const
    allocate memory with binding to a numa node set as specified by the policy and flags (see hwloc
    docs)

threads::mask_type get_area_membind_nodeset (const void *addr, std::size_t len)
const

bool set_area_membind_nodeset (const void *addr, std::size_t len, void *nodeset)
const

int get_numa_domain (const void *addr) const

void deallocate (void *addr, std::size_t len) const
    Free memory that was previously allocated by allocate.

void print_vector (std::ostream &os, std::vector<std::size_t> const &v) const

void print_mask_vector (std::ostream &os, std::vector<mask_type> const &v) const

void print_hwloc (std::ostream&) const

mask_type init_socket_affinity_mask_from_socket (std::size_t num_socket)
const

mask_type init_numa_node_affinity_mask_from_numa_node (std::size_t num_numa_node)
const

mask_type init_core_affinity_mask_from_core (std::size_t num_core,
                                              mask_cref_type default_mask,
                                              = empty_mask) const

mask_type init_thread_affinity_mask (std::size_t num_thread) const

mask_type init_thread_affinity_mask (std::size_t num_core, std::size_t num_pu)
const

hwloc_bitmap_t mask_to_bitmap (mask_cref_type mask, hwloc_obj_type_t htype) const

mask_type bitmap_to_mask (hwloc_bitmap_t bitmap, hwloc_obj_type_t htype) const
```

Private Types

```
using mutex_type = hpx::util::spinlock
```

Private Functions

```
std::size_t init_node_number (std::size_t num_thread, hwloc_obj_type_t type)
std::size_t init_socket_number (std::size_t num_thread)
std::size_t init numa_node_number (std::size_t num_thread)
std::size_t init core_number (std::size_t num_thread)

void extract_node_mask (hwloc_obj_t parent, mask_type &mask) const
std::size_t extract_node_count (hwloc_obj_t parent, hwloc_obj_type_t type, std::size_t count) const

mask_type init_machine_affinity_mask () const
mask_type init_socket_affinity_mask (std::size_t num_thread) const
mask_type init numa_node_affinity_mask (std::size_t num_thread) const
mask_type init core_affinity_mask (std::size_t num_thread) const

void init_num_of_pus ()
```

Private Members

```
hwloc_topology_t topo
std::size_t num_of_pus_
bool use_pus_as_cores_
mutex_type topo_mtx
std::vector<std::size_t> socket_numbers_
std::vector<std::size_t> numa_node_numbers_
std::vector<std::size_t> core_numbers_
mask_type machine_affinity_mask_
std::vector<mask_type> socket_affinity_masks_
std::vector<mask_type> numa_node_affinity_masks_
std::vector<mask_type> core_affinity_masks_
std::vector<mask_type> thread_affinity_masks_
```

Private Static Attributes

```
mask_type empty_mask
std::size_t memory_page_size_
constexpr std::size_t pu_offset = 0
constexpr std::size_t core_offset = 0
```

Friends

```
std::size_t get_memory_page_size()
```

util

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/util/insert_checked.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

Functions

```
template<typename Iterator>
bool insert_checked(std::pair<Iterator, bool> const &r)
```

Helper function for writing predicates that test whether an std::map insertion succeeded. This inline template function negates the need to explicitly write the sometimes lengthy std::pair<Iterator, bool> type.

Return This function returns **r.second**.

Parameters

- **r**: [in] The return value of a std::map insert operation.

```
template<typename Iterator>
bool insert_checked(std::pair<Iterator, bool> const &r, Iterator &it)
```

Helper function for writing predicates that test whether an std::map insertion succeeded. This inline template function negates the need to explicitly write the sometimes lengthy std::pair<Iterator, bool> type.

Return This function returns **r.second**.

Parameters

- **r**: [in] The return value of a std::map insert operation.
- **r**: [out] A reference to an Iterator, which is set to **r.first**.

hpx/util/sed_transform.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

Functions

```
bool parse_sed_expression(std::string const &input, std::string &search, std::string &replace)
```

Parse a sed command.

Return *true* if the parsing was successful, false otherwise.

Note Currently, only supports search and replace syntax (s/search/replace/)

Parameters

- *input*: [in] The content to parse.
- *search*: [out] If the parsing is successful, this string is set to the search expression.
- *search*: [out] If the parsing is successful, this string is set to the replace expression.

```
struct sed_transform
```

#include <sed_transform.hpp> An unary function object which applies a sed command to its subject and returns the resulting string.

Note Currently, only supports search and replace syntax (s/search/replace/)

Public Functions

```
sed_transform(std::string const &search, std::string const &replace)
```

```
sed_transform(std::string const &expression)
```

```
std::string operator() (std::string const &input) const
```

```
operator bool() const
```

```
bool operator!() const
```

Private Members

```
std::shared_ptr<command> command_
```

actions

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions/action_support.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions/actions_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions/base_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions/transfer_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions/transfer_base_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

actions_base

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions_base/actions_base_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions_base/actions_base_support.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions_base/basic_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_ACTION_DECLARATION(...)

Declare the necessary component action boilerplate code.

The macro *HPX_REGISTER_ACTION_DECLARATION* can be used to declare all the boilerplate code which is required for proper functioning of component actions in the context of HPX.

The parameter *action* is the type of the action to declare the boilerplate for.

This macro can be invoked with an optional second parameter. This parameter specifies a unique name of the action to be used for serialization purposes. The second parameter has to be specified if the first parameter is not usable as a plain (non-qualified) C++ identifier, i.e. the first parameter contains special characters which cannot be part of a C++ identifier, such as '<', '>', or ':'.

```
namespace app
{
    // Define a simple component exposing one action 'print_greeting'
    class HPX_COMPONENT_EXPORT server
        : public hpx::components::component_base<server>
    {
        void print_greeting ()
        {
            hpx::cout << "Hey, how are you?\n" << std::flush;
        }

        // Component actions need to be declared, this also defines the
        // type 'print_greeting_action' representing the action.
        HPX_DEFINE_COMPONENT_ACTION(server,
            print_greeting, print_greeting_action);
    };
}

// Declare boilerplate code required for each of the component actions.
HPX_REGISTER_ACTION_DECLARATION(app::server::print_greeting_action)
```

Example:

Note This macro has to be used once for each of the component actions defined using one of the *HPX_DEFINE_COMPONENT_ACTION* macros. It has to be visible in all translation units using the action, thus it is recommended to place it into the header file defining the component.

HPX_REGISTER_ACTION_DECLARATION(...)

HPX_REGISTER_ACTION_DECLARATION_1(*action*)

HPX_REGISTER_ACTION(...)

Define the necessary component action boilerplate code.

The macro *HPX_REGISTER_ACTION* can be used to define all the boilerplate code which is required for proper functioning of component actions in the context of HPX.

The parameter *action* is the type of the action to define the boilerplate for.

This macro can be invoked with an optional second parameter. This parameter specifies a unique name of the action to be used for serialization purposes. The second parameter has to be specified if the first parameter is not

usable as a plain (non-qualified) C++ identifier, i.e. the first parameter contains special characters which cannot be part of a C++ identifier, such as '<', '>', or ':'.

Note This macro has to be used once for each of the component actions defined using one of the *HPX_DEFINE_COMPONENT_ACTION* or *HPX_DEFINE_PLAIN_ACTION* macros. It has to occur exactly once for each of the actions, thus it is recommended to place it into the source file defining the component.

Note Only one of the forms of this macro *HPX_REGISTER_ACTION* or *HPX_REGISTER_ACTION_ID* should be used for a particular action, never both.

HPX_REGISTER_ACTION_ID (*action*, *actionname*, *actionid*)

Define the necessary component action boilerplate code and assign a predefined unique id to the action.

The macro *HPX_REGISTER_ACTION* can be used to define all the boilerplate code which is required for proper functioning of component actions in the context of HPX.

The parameter *action* is the type of the action to define the boilerplate for.

The parameter *actionname* specifies an unique name of the action to be used for serialization purposes. The second parameter has to be usable as a plain (non-qualified) C++ identifier, it should not contain special characters which cannot be part of a C++ identifier, such as '<', '>', or ':'.

The parameter *actionid* specifies an unique integer value which will be used to represent the action during serialization.

Note This macro has to be used once for each of the component actions defined using one of the *HPX_DEFINE_COMPONENT_ACTION* or global actions *HPX_DEFINE_PLAIN_ACTION* macros. It has to occur exactly once for each of the actions, thus it is recommended to place it into the source file defining the component.

Note Only one of the forms of this macro *HPX_REGISTER_ACTION* or *HPX_REGISTER_ACTION_ID* should be used for a particular action, never both.

hpx/actions_base/basic_action_fwd.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace actions

```
template<typename Component, typename Signature, typename Derived>
struct basic_action
```

```
    #include <basic_action_fwd.hpp>
```

Template Parameters

- **Component**: component type
- **Signature**: return type and arguments
- **Derived**: derived action class

hpx/actions_base/component_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

`HPX_DEFINE_COMPONENT_ACTION(...)`

Registers a member function of a component as an action type with HPX.

The macro `HPX_DEFINE_COMPONENT_ACTION` can be used to register a member function of a component as an action type named `action_type`.

The parameter `component` is the type of the component exposing the member function `func` which should be associated with the newly defined action type. The parameter `action_type` is the name of the action type to register with HPX.

```
namespace app
{
    // Define a simple component exposing one action 'print_greeting'
    class HPX_COMPONENT_EXPORT server
        : public hpx::components::component_base<server>
    {
        void print_greeting() const
        {
            hpx::cout << "Hey, how are you?\n" << std::flush;
        }

        // Component actions need to be declared, this also defines the
        // type 'print_greeting_action' representing the action.
        HPX_DEFINE_COMPONENT_ACTION(server, print_greeting,
            print_greeting_action);
    };
}
```

Example:

The first argument must provide the type name of the component the action is defined for.

The second argument must provide the member function name the action should wrap.

The default value for the third argument (the typename of the defined action) is derived from the name of the function (as passed as the second argument) by appending ‘_action’. The third argument can be omitted only if the second argument with an appended suffix ‘_action’ resolves to a valid, unqualified C++ type name.

Note The macro `HPX_DEFINE_COMPONENT_ACTION` can be used with 2 or 3 arguments. The third argument is optional.

hpx/actions_base/lambda_to_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/actions_base/plain_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_DEFINE_PLAIN_ACTION(...)

Defines a plain action type.

```
namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }

    // This will define the action type 'app::some_global_action' which
    // represents the function 'app::some_global_function'.
    HPX_DEFINE_PLAIN_ACTION(some_global_function, some_global_action);
}
```

Example:

Note Usually this macro will not be used in user code unless the intent is to avoid defining the action_type in global namespace. Normally, the use of the macro `HPX_PLAIN_ACTION` is recommended.

Note The macro `HPX_DEFINE_PLAIN_ACTION` can be used with 1 or 2 arguments. The second argument is optional. The default value for the second argument (the typename of the defined action) is derived from the name of the function (as passed as the first argument) by appending '_action'. The second argument can be omitted only if the first argument with an appended suffix '_action' resolves to a valid, unqualified C++ type name.

HPX_DECLARE_PLAIN_ACTION(...)

Declares a plain action type.

HPX_PLAIN_ACTION(...)

Defines a plain action type based on the given function *func* and registers it with HPX.

The macro `HPX_PLAIN_ACTION` can be used to define a plain action (e.g. an action encapsulating a global or free function) based on the given function *func*. It defines the action type *name* representing the given function. This macro additionally registers the newly define action type with HPX.

The parameter *func* is a global or free (non-member) function which should be encapsulated into a plain action. The parameter *name* is the name of the action type defined by this macro.

```

namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }
}

// This will define the action type 'some_global_action' which represents
// the function 'app::some_global_function'.
HPX_PLAIN_ACTION(app::some_global_function, some_global_action)

```

Example:

Note The macro *HPX_PLAIN_ACTION* has to be used at global namespace even if the wrapped function is located in some other namespace. The newly defined action type is placed into the global namespace as well.

Note The macro *HPX_PLAIN_ACTION_ID* can be used with 1, 2, or 3 arguments. The second and third arguments are optional. The default value for the second argument (the typename of the defined action) is derived from the name of the function (as passed as the first argument) by appending ‘_action’. The second argument can be omitted only if the first argument with an appended suffix ‘_action’ resolves to a valid, unqualified C++ type name. The default value for the third argument is *hpx::components::factory_check*.

Note Only one of the forms of this macro *HPX_PLAIN_ACTION* or *HPX_PLAIN_ACTION_ID* should be used for a particular action, never both.

HPX_PLAIN_ACTION_ID (*func, name, id*)

Defines a plain action type based on the given function *func* and registers it with HPX.

The macro *HPX_PLAIN_ACTION_ID* can be used to define a plain action (e.g. an action encapsulating a global or free function) based on the given function *func*. It defines the action type *actionname* representing the given function. The parameter *actionid*

The parameter *actionid* specifies an unique integer value which will be used to represent the action during serialization.

The parameter *func* is a global or free (non-member) function which should be encapsulated into a plain action. The parameter *name* is the name of the action type defined by this macro.

The second parameter has to be usable as a plain (non-qualified) C++ identifier, it should not contain special characters which cannot be part of a C++ identifier, such as ‘<’, ‘>’, or ‘.’.

```

namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }
}

// This will define the action type 'some_global_action' which represents
// the function 'app::some_global_function'.
HPX_PLAIN_ACTION_ID(app::some_global_function, some_global_action,
                    some_unique_id);

```

Example:

Note The macro `HPX_PLAIN_ACTION_ID` has to be used at global namespace even if the wrapped function is located in some other namespace. The newly defined action type is placed into the global namespace as well.

Note Only one of the forms of this macro `HPX_PLAIN_ACTION` or `HPX_PLAIN_ACTION_ID` should be used for a particular action, never both.

`hpx/actions_base/preassigned_action_id.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`hpx/actions_base/traits/action_remote_result.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace traits`

TypeDefs

```
template<typename Result>
using action_remote_result_t = typename action_remote_result<Result>::type
```

`agas`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`hpx/agas/addressing_service.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace agas`

```
struct addressing_service
```

Public Types

```

using component_id_type = components::component_type
using iterate_names_return_type = std::map<std::string, hpx::id_type>
using iterate_types_function_type = hpx::function<void (std::string
const&, compo-
nents::component_type),  

true>
using mutex_type = hpx::spinlock
using gva_cache_type = hpx::util::cache::lru_cache<gva_cache_key, gva, hpx::util::cache::statistics::local_full_
using migrated_objects_table_type = std::set<naming::gid_type>
using refcnt_requests_type = std::map<naming::gid_type, std::int64_t>
using resolved_localities_type = std::map<naming::gid_type, parcelset::endpoints_type>

```

Public Functions

```

HPX_NON_COPYABLE (addressing_service)
addressing_service (util::runtime_configuration const &ini_)
~addressing_service ()

void bootstrap (parcelset::endpoints_type const &endpoints, util::runtime_configuration  

&rtcfg)
void initialize (std::uint64_t rts_lva)
void adjust_local_cache_size (std::size_t)
    Adjust the size of the local AGAS Address resolution cache.

state get_status () const
void set_status (state new_state)
naming::gid_type const &get_local_locality (error_code& = throws) const
void set_local_locality (naming::gid_type const &g)
void register_console (parcelset::endpoints_type const &eps)
bool is_bootstrap () const
bool is_console () const
    Returns whether this addressing_service represents the console locality.

bool is_connecting () const
    Returns whether this addressing_service is connecting to a running application.

bool resolve_locally_known_addresses (naming::gid_type const &id, naming::address &addr)
void register_server_instances ()
void garbage_collect_non_blocking (error_code &ec = throws)

```

```
void garbage_collect (error_code &ec = throws)  
  
    std::int64_t synchronize_with_async_incref (hpx::future<std::int64_t> fut,  
                                                hpx::id_type const &id, std::int64_t compensated_credit)  
  
    server::primary_namespace &get_local_primary_namespace_service()  
  
    naming::address::address_type get_primary_ns_lva() const  
  
    naming::address::address_type get_symbol_ns_lva() const  
  
    server::component_namespace *get_local_component_namespace_service()  
  
    server::locality_namespace *get_local_locality_namespace_service()  
  
    server::symbol_namespace &get_local_symbol_namespace_service()  
  
    naming::address::address_type get_runtime_support_lva() const  
  
    std::uint64_t get_cache_entries (bool)  
  
    std::uint64_t get_cache_hits (bool)  
  
    std::uint64_t get_cache_misses (bool)  
  
    std::uint64_t get_cache_evictions (bool)  
  
    std::uint64_t get_cache_insertions (bool)  
  
    std::uint64_t get_cache_get_entry_count (bool reset)  
  
    std::uint64_t get_cache_insertion_entry_count (bool reset)  
  
    std::uint64_t get_cache_update_entry_count (bool reset)  
  
    std::uint64_t get_cache_erase_entry_count (bool reset)  
  
    std::uint64_t get_cache_get_entry_time (bool reset)  
  
    std::uint64_t get_cache_insertion_entry_time (bool reset)  
  
    std::uint64_t get_cache_update_entry_time (bool reset)  
  
    std::uint64_t get_cache_erase_entry_time (bool reset)  
  
    bool register_locality (parcelset::endpoints_type const &endpoints, naming::gid_type &prefix, std::uint32_t num_threads, error_code &ec = throws)  
        Add a locality to the runtime.  
  
parcelset::endpoints_type const &resolve_locality (naming::gid_type const &gid, error_code &ec = throws)  
        Resolve a locality to its prefix.
```

Return Returns an empty vector if the locality is not registered.

```
bool has_resolved_locality (naming::gid_type const &gid)
```

```
bool unregister_locality (naming::gid_type const &gid, error_code &ec = throws)
```

Remove a locality from the runtime.

```
void remove_resolved_locality(naming::gid_type const &gid)
    remove given locality from locality cache

bool get_console_locality(naming::gid_type &locality, error_code &ec = throws)
    Get locality locality_id of the console locality.
```

Return This function returns *true* if a console locality_id exists and returns *false* otherwise.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *locality_id*: [out] The locality_id value uniquely identifying the console locality. This is valid only, if the return value of this function is true.
- *try_cache*: [in] If this is set to true the console is first tried to be found in the local cache. Otherwise this function will always query AGAS, even if the console locality_id is already known locally.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool get_localities(std::vector<naming::gid_type> &locality_ids, components::component_type type, error_code &ec = throws)
```

Query for the locality_ids of all known localities.

This function returns the locality_ids of all localities known to the AGAS server or all localities having a registered factory for a given component type.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *locality_ids*: [out] The vector will contain the prefixes of all localities registered with the AGAS server. The returned vector holds the prefixes representing the runtime_support components of these localities.
- *type*: [in] The component type will be used to determine the set of prefixes having a registered factory for this component. The default value for this parameter is *components::component_invalid*, which will return prefixes of all localities.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool get_localities(std::vector<naming::gid_type> &locality_ids, error_code &ec = throws)
```

```
hpx::future<std::uint32_t> get_num_localities_async(components::component_type
    type = components::component_invalid)
const
```

Query for the number of all known localities.

This function returns the number of localities known to the AGAS server or the number of localities having a registered factory for a given component type.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *type*: [in] The component type will be used to determine the set of prefixes having a registered factory for this component. The default value for this parameter is *components::component_invalid*, which will return prefixes of all localities.

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
std::uint32_t get_num_localities (components::component_type type, error_code &ec =  
    throws) const  
  
std::uint32_t get_num_localities (error_code &ec = throws) const  
  
hpx::future<std::uint32_t> get_num_overall_threads_async () const  
  
std::uint32_t get_num_overall_threads (error_code &ec = throws) const  
  
hpx::future<std::vector<std::uint32_t>> get_num_threads_async () const  
  
std::vector<std::uint32_t> get_num_threads (error_code &ec = throws) const  
  
components::component_type get_component_id (std::string const &name, error_code  
    &ec = throws)
```

Return a unique id usable as a component type.

This function returns the component type id associated with the given component name. If this is the first request for this component name a new unique id will be created.

Return The function returns the currently associated component type. Any error results in an exception thrown from this function.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `name`: [in] The component name (string) to get the component type for.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
void iterate_types (iterate_types_function_type const &f, error_code &ec = throws)  
  
std::string get_component_type_name (components::component_type id, error_code &ec  
    = throws)  
  
components::component_type register_factory (naming::gid_type const &locality_id,  
    std::string const &name, error_code  
    &ec = throws)
```

Register a factory for a specific component type.

This function allows to register a component factory for a given locality and component type.

Return The function returns the currently associated component type. Any error results in an exception thrown from this function. The returned component type is the same as if the function `get_component_id` was called using the same component name.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `locality_id`: [in] The locality value uniquely identifying the given locality the factory needs to be registered for.
- `name`: [in] The component name (string) to register a factory for the given component type for.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
components::component_type register_factory(std::uint32_t locality_id, std::string
                                         const &name, error_code &ec =
                                         throws)
```

```
bool get_id_range(std::uint64_t count, naming::gid_type &lower_bound, naming::gid_type
                  &upper_bound, error_code &ec = throws)
```

Get unique range of freely assignable global ids.

Every locality needs to be able to assign global ids to different components without having to consult the AGAS server for every id to generate. This function can be called to preallocate a range of ids usable for this purpose.

Return This function returns *true* if a new range has been generated (it has been called for the first time for the given locality) and returns *false* if this locality already got a range assigned in an earlier call. Any error results in an exception thrown from this function.

Note This function assigns a range of global ids usable by the given locality for newly created components. Any of the returned global ids still has to be bound to a local address, either by calling *bind* or *bind_range*.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *l*: [in] The locality the locality id needs to be generated for. Repeating calls using the same locality results in identical locality_id values.
- *count*: [in] The number of global ids to be generated.
- *lower_bound*: [out] The lower bound of the assigned id range. The returned value can be used as the first id to assign. This is valid only, if the return value of this function is true.
- *upper_bound*: [out] The upper bound of the assigned id range. The returned value can be used as the last id to assign. This is valid only, if the return value of this function is true.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool bind_local(naming::gid_type const &id, naming::address const &addr, error_code
                &ec = throws)
```

Bind a global address to a local address.

Every element in the HPX namespace has a unique global address (global id). This global address has to be associated with a concrete local address to be able to address an instance of a component using its global address.

Return This function returns *true*, if this global id got associated with an local address. It returns *false* otherwise.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note Binding a gid to a local address sets its global reference count to one.

Parameters

- *id*: [in] The global address which has to be bound to the local address.
- *addr*: [in] The local address to be bound to the global address.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
hpx::future<bool> bind_async(naming::gid_type const &id, naming::address const
                                &addr, std::uint32_t locality_id)
```

```
hpx::future<bool> bind_async(naming::gid_type const &id, naming::address const
                                &addr, naming::gid_type const &locality)
```

```
bool bind_range_local (naming::gid_type const &lower_id, std::uint64_t count, naming::address const &baseaddr, std::uint64_t offset, error_code &ec = throws)
```

Bind unique range of global ids to given base address.

Every locality needs to be able to bind global ids to different components without having to consult the AGAS server for every id to bind. This function can be called to bind a range of consecutive global ids to a range of consecutive local addresses (separated by a given *offset*).

Return This function returns *true*, if the given range was successfully bound. It returns *false* otherwise.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note Binding a gid to a local address sets its global reference count to one.

Parameters

- *lower_id*: [in] The lower bound of the assigned id range. The value can be used as the first id to assign.
- *count*: [in] The number of consecutive global ids to bind starting at *lower_id*.
- *baseaddr*: [in] The local address to bind to the global id given by *lower_id*. This is the base address for all additional local addresses to bind to the remaining global ids.
- *offset*: [in] The offset to use to calculate the local addresses to be bound to the range of global ids.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
hpx::future<bool> bind_range_async (naming::gid_type const &lower_id, std::uint64_t count, naming::address const &baseaddr, std::uint64_t offset, naming::gid_type const &locality)
```

```
hpx::future<bool> bind_range_async (naming::gid_type const &lower_id, std::uint64_t count, naming::address const &baseaddr, std::uint64_t offset, std::uint32_t locality_id)
```

```
bool unbind_local (naming::gid_type const &id, error_code &ec = throws)
```

Unbind a global address.

Remove the association of the given global address with any local address, which was bound to this global address. Additionally it returns the local address which was bound at the time of this call.

Return The function returns *true* if the association has been removed, and it returns *false* if no association existed. Any error results in an exception thrown from this function.

Note You can unbind only global ids bound using the function *bind*. Do not use this function to unbind any of the global ids bound using *bind_range*.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note This function will raise an error if the global reference count of the given gid is not zero!
TODO: confirm that this happens.

Parameters

- *id*: [in] The global address (id) for which the association has to be removed.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool unbind_local (naming::gid_type const &id, naming::address &addr, error_code &ec = throws)
```

Unbind a global address.

Remove the association of the given global address with any local address, which was bound to this global address. Additionally it returns the local address which was bound at the time of this call.

Return The function returns *true* if the association has been removed, and it returns *false* if no association existed. Any error results in an exception thrown from this function.

Note You can unbind only global ids bound using the function *bind*. Do not use this function to unbind any of the global ids bound using *bind_range*.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note This function will raise an error if the global reference count of the given gid is not zero!

TODO: confirm that this happens.

Parameters

- *id*: [in] The global address (id) for which the association has to be removed.
- *addr*: [out] The local address which was associated with the given global address (id). This is valid only if the return value of this function is true.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool unbind_range_local(naming::gid_type const &lower_id, std::uint64_t count, error_code &ec = throws)
```

Unbind the given range of global ids.

Return This function returns *true* if a new range has been generated (it has been called for the first time for the given locality) and returns *false* if this locality already got a range assigned in an earlier call. Any error results in an exception thrown from this function.

Note You can unbind only global ids bound using the function *bind_range*. Do not use this function to unbind any of the global ids bound using *bind*.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note This function will raise an error if the global reference count of the given gid is not zero!

TODO: confirm that this happens.

Parameters

- *lower_id*: [in] The lower bound of the assigned id range. The value must be the first id of the range as specified to the corresponding call to *bind_range*.
- *count*: [in] The number of consecutive global ids to unbind starting at *lower_id*. This number must be identical to the number of global ids bound by the corresponding call to *bind_range*.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
bool unbind_range_local(naming::gid_type const &lower_id, std::uint64_t count, naming::address &addr, error_code &ec = throws)
```

Unbind the given range of global ids.

Return This function returns *true* if a new range has been generated (it has been called for the first time for the given locality) and returns *false* if this locality already got a range assigned in an earlier call.

Note You can unbind only global ids bound using the function *bind_range*. Do not use this function to unbind any of the global ids bound using *bind*.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note This function will raise an error if the global reference count of the given gid is not zero!

Parameters

- `lower_id`: [in] The lower bound of the assigned id range. The value must be the first id of the range as specified to the corresponding call to `bind_range`.
- `count`: [in] The number of consecutive global ids to unbind starting at `lower_id`. This number must be identical to the number of global ids bound by the corresponding call to `bind_range`
- `addr`: [out] The local address which was associated with the given global address (`id`). This is valid only if the return value of this function is true.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
hpx::future<naming::address> unbind_range_async (naming::gid_type const
&lower_id, std::uint64_t count =
1)
```

```
bool is_local_address_cached (naming::gid_type const &id, error_code &ec =
throws)
```

Test whether the given address refers to a local object.

This function will test whether the given address refers to an object living on the locality of the caller.

Return This function returns *true* if the passed address refers to an object which lives on the locality of the caller.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `addr`: [in] The address to test.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
bool is_local_address_cached (naming::gid_type const &id, naming::address &addr,
error_code &ec = throws)
```

```
bool is_local_lva_encoded_address (std::uint64_t msb)
```

```
bool resolve_local (naming::gid_type const &id, naming::address &addr, error_code
&ec = throws)
```

Resolve a given global address (`id`) to its associated local address.

This function returns the local address which is currently associated with the given global address (`id`).

Return This function returns *true* if the global address has been resolved successfully (there exists an association to a local address) and the associated local address has been returned. The function returns *false* if no association exists for the given global address. Any error results in an exception thrown from this function.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `id`: [in] The global address (`id`) for which the associated local address should be returned.
- `addr`: [out] The local address which currently is associated with the given global address (`id`), this is valid only if the return value of this function is true.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```

bool resolve_local (hpx::id_type const &id, naming::address &addr, error_code &ec =
throws)
naming::address resolve_local (naming::gid_type const &id, error_code &ec = throws)
naming::address resolve_local (hpx::id_type const &id, error_code &ec = throws)
hpx::future<naming::address> resolve_async (naming::gid_type const &id)
hpx::future<naming::address> resolve_async (hpx::id_type const &id)
hpx::future<hpx::id_type> get_colocation_id_async (hpx::id_type const &id)

bool resolve_full_local (naming::gid_type const &id, naming::address &addr, error_code &ec =
throws)
bool resolve_full_local (hpx::id_type const &id, naming::address &addr, error_code &ec =
throws)
naming::address resolve_full_local (naming::gid_type const &id, error_code &ec =
throws)
naming::address resolve_full_local (hpx::id_type const &id, error_code &ec =
throws)
hpx::future<naming::address> resolve_full_async (naming::gid_type const &id)
hpx::future<naming::address> resolve_full_async (hpx::id_type const &id)

bool resolve_cached (naming::gid_type const &id, naming::address &addr, error_code &ec =
throws)
bool resolve_cached (hpx::id_type const &id, naming::address &addr, error_code &ec =
throws)
bool resolve_local (naming::gid_type const *gids, naming::address *addrs, std::size_t
size, hpx::detail::dynamic_bitset<> &locals, error_code &ec =
throws)
bool resolve_full_local (naming::gid_type const *gids, naming::address *addrs,
std::size_t size, hpx::detail::dynamic_bitset<> &locals, error_code &ec =
throws)
bool resolve_cached (naming::gid_type const *gids, naming::address *addrs, std::size_t
size, hpx::detail::dynamic_bitset<> &locals, error_code &ec =
throws)
hpx::future<std::int64_t> incref_async (naming::gid_type const &gid, std::int64_t credits = 1, hpx::id_type const &keep_alive = hpx::invalid_id)

```

Increment the global reference count for the given id.

Return Whether the operation was successful.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The global address (id) for which the global reference count has to be incremented.
- *credits*: [in] The number of reference counts to add for the given id.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
std::int64_t incref (naming::gid_type const &gid, std::int64_t credits = 1, error_code &ec =  
throws)
```

```
void decref (naming::gid_type const &id, std::int64_t credits = 1, error_code &ec = throws)
```

Decrement the global reference count for the given id.

Return The global reference count after the decrement.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *id*: [in] The global address (id) for which the global reference count has to be decremented.
- *t*: [out] If this was the last outstanding global reference for the given gid (the return value of this function is zero), *t* will be set to the component type of the corresponding element. Otherwise *t* will not be modified.
- *credits*: [in] The number of reference counts to add for the given id.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
hpx::future<iterate_names_return_type> iterate_ids (std::string const &pattern)
```

Invoke the supplied *hpx::function* for every registered global name.

This function iterates over all registered global ids and returns every found entry matching the given name pattern. Any error results in an exception thrown (or reported) from this function.

Parameters

- *pattern*: [in] pattern (possibly using wildcards) to match all existing entries against

```
bool register_name (std::string const &name, naming::gid_type const &id, error_code  
&ec = throws)
```

Register a global name with a global address (id)

This function registers an association between a global name (string) and a global address (id) usable with one of the functions above (bind, unbind, and resolve).

Return The function returns *true* if the global name was registered. It returns false if the global name is not registered.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *name*: [in] The global name (string) to be associated with the global address.
- *id*: [in] The global address (id) to be associated with the global address.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
hpx::future<bool> register_name_async (std::string const &name, hpx::id_type const  
&id)
```

```
bool register_name (std::string const &name, hpx::id_type const &id, error_code &ec  
= throws)
```

```
hpx::future<hpx::id_type> unregister_name_async (std::string const &name)
```

Unregister a global name (release any existing association)

This function releases any existing association of the given global name with a global address (id).

Return The function returns *true* if an association of this global name has been released, and it returns *false*, if no association existed. Any error results in an exception thrown from this function.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *name*: [in] The global name (string) for which any association with a global address (*id*) has to be released.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

hpx::id_type unregister_name (std::string const &name, error_code &ec = throws)

hpx::future<hpx::id_type> resolve_name_async (std::string const &name)

Query for the global address associated with a given global name.

This function returns the global address associated with the given global name.

This function returns true if it returned global address (*id*), which is currently associated with the given global name, and it returns false, if currently there is no association for this global name. Any error results in an exception thrown from this function.

Return [out] The id currently associated with the given global name (valid only if the return value is true).

Parameters

- *name*: [in] The global name (string) for which the currently associated global address has to be retrieved.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

hpx::id_type resolve_name (std::string const &name, error_code &ec = throws)

future<hpx::id_type> on_symbol_namespace_event (std::string const &name, bool call_for_past_events = false)

Install a listener for a given symbol namespace event.

This function installs a listener for a given symbol namespace event. It returns a future which becomes ready as a result of the listener being triggered.

Return A future instance encapsulating the global id which is causing the registered listener to be triggered.

Note The only event type which is currently supported is *symbol_ns_bind*, i.e. the listener is triggered whenever a global id is registered with the given name.

Parameters

- *name*: [in] The global name (string) for which the given event should be triggered.
- *evt*: [in] The event for which a listener should be installed.
- *call_for_past_events*: [in, optional] Trigger the listener even if the given event has already happened in the past. The default for this parameter is *false*.

void update_cache_entry (naming::gid_type const &gid, gva const &gva, error_code &ec = throws)

Warning This function is for internal use only. It is dangerous and may break your code if you use it.

```
void update_cache_entry(naming::gid_type const &gid, naming::address const &addr, std::uint64_t count = 0, std::uint64_t offset = 0, error_code &ec = throws)
```

Warning This function is for internal use only. It is dangerous and may break your code if you use it.

```
bool get_cache_entry(naming::gid_type const &gid, gva &gva, naming::gid_type &id_base, error_code &ec = throws)
```

Warning This function is for internal use only. It is dangerous and may break your code if you use it.

```
void remove_cache_entry(naming::gid_type const &id, error_code &ec = throws)
```

Warning This function is for internal use only. It is dangerous and may break your code if you use it.

```
void clear_cache(error_code &ec = throws)
```

Warning This function is for internal use only. It is dangerous and may break your code if you use it.

```
void start_shutdown(error_code &ec = throws)
```

```
hpx::future<std::pair<hpx::id_type, naming::address>> begin_migration(hpx::id_type const &id)
```

start/stop migration of an object

Return Current locality and address of the object to migrate

```
bool end_migration(hpx::id_type const &id)
```

```
std::pair<bool, components::pinned_ptr> was_object_migrated(naming::gid_type const &gid, hpx::move_only_function<components::pinned> > &&fMaintain list of migrated objects.
```

```
hpx::future<void> mark_as_migrated(naming::gid_type const &gid, hpx::move_only_function<std::pair<bool, hpx::future<void>> > &&fbool expect_to_be_marked_as_migratingMark the given object as being migrated (if the object is unpinned). Delay migration until the object is unpinned otherwise.
```

```
void unmark_as_migrated(naming::gid_type const &gid)
```

Remove the given object from the table of migrated objects.

```
void pre_cache_endpoints(std::vector<parcelset::endpoints_type> const &)
```

Public Members

```
mutex_type gva_cache_mtx_
std::shared_ptr<gva_cache_type> gva_cache_
mutex_type migrated_objects_mtx_
migrated_objects_table_type migrated_objects_table_
mutex_type console_cache_mtx_
std::uint32_t console_cache_
```

```

const std::size_t max_refcnt_requests_
mutex_type refcnt_requests_mtx_
std::size_t refcnt_requests_count_
bool enable_refcnt_caching_
std::shared_ptr<refcnt_requests_type> refcnt_requests_
const service_mode service_type
const runtime_mode runtime_type
const bool caching_
const bool range_caching_
const threads::thread_priority action_priority_
std::uint64_t rts_lva_
std::unique_ptr<component_namespace> component_ns_
std::unique_ptr<locality_namespace> locality_ns_
symbol_namespace symbol_ns_
primary_namespace primary_ns_
std::atomic<hpx::state> state_
naming::gid_type locality_
mutex_type resolved_localities_mtx_
resolved_localities_type resolved_localities_

```

Protected Functions

```

void launch_bootstrap (parcelset::endpoints_type const &endpoints,
                      util::runtime_configuration &rte)
naming::address resolve_full_postproc (naming::gid_type const &id, future<primary_namespace::resolved_type> f)
bool bind_postproc (naming::gid_type const &id, gva const &g, future<bool> f)
bool was_object_migrated_locked (naming::gid_type const &id)
    Maintain list of migrated objects.

```

Private Functions

```

void send_refcnt_requests (std::unique_lock<mutex_type> &l, error_code &ec = throws)
    Assumes that refcnt_requests_mtx_ is locked.
void send_refcnt_requests_non_blocking (std::unique_lock<mutex_type> &l, error_code &ec)
    Assumes that refcnt_requests_mtx_ is locked.

```

```
std::vector<hpx::future<std::vector<std::int64_t>>> send_refptr_requests_async(std::unique_lock<mutex_type> &l)
```

Assumes that *refcnt_requests_mtx*_ is locked.

```
void send_refptr_requests_sync(std::unique_lock<mutex_type> &l, error_code &ec)
```

Assumes that *refcnt_requests_mtx*_ is locked.

agas_base

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/agas_base/server/primary_namespace.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

Variables

```
HPX_ACTIONUSES_MEDIUM_STACK ( hpx::agas::server::primary_namespace::allocate_action) HPX_1  
namespace hpx
```

```
namespace agas
```

Functions

```
naming::gid_type bootstrap_primary_namespace_gid()
```

```
hpx::id_type bootstrap_primary_namespace_id()
```

```
namespace server
```

AGAS's primary namespace maps 128-bit global identifiers (GIDs) to resolved addresses.

The following is the canonical description of the partitioning of AGAS's primary namespace.

-----MSB----- -----LSB-----
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
prefix RC ---identifier---
MSB - Most significant bits (bit 64 to bit 127)
LSB - Least significant bits (bit 0 to bit 63)
prefix - Highest 32 bits (bit 96 to bit 127) of the MSB. Each locality is assigned a prefix. This creates a 96-bit address space for each locality.
RC - Bit 88 to bit 92 of the MSB. This is the log2 of the number of reference counting credits on the GID. Bit 93 is used by the locking scheme for gid_types. Bit 94 is a flag which is set if the credit value is valid. Bit 95 is a flag that is set if a GID's credit count is ever split (e.g. if the GID is ever passed to another locality). - Bit 87 marks the gid such that it will not be stored in

(continues on next page)

(continued from previous page)

any of the AGAS caches. This is used mainly for ids which represent 'one-shot' objects (like promises). identifier - Bit 64 to bit 86 of the MSB, and the entire LSB. The content of these bits depends on the component type of the underlying object. For all user-defined components, these bits contain a unique 88-bit number which is assigned sequentially for each locality. For `\a hpx#components#component_runtime_support` the high 24 bits are zeroed and the low 64 bits hold the LVA of the component.

Note The layout of the address space is implementation defined, and subject to change. Never write application code that relies on the internal layout of GIDs. AGAS only guarantees that all assigned GIDs will be unique.

The following address ranges are reserved. Some are either explicitly or implicitly protected by AGAS. The letter x represents a single-byte wild card.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Historically unused address space reserved for future use.
xxxxxxxxxxxx0000xxxxxxxxxxxxxx
Address space for LVA-encoded GIDs.
00000001xxxxxxxxxxxxxxxxxxxxxx
Prefix of the bootstrap AGAS locality.
0000000100000001000000000000000001
Address of the primary_namespace component on the bootstrap AGAS
locality.
0000000100000001000000000000000002
Address of the component_namespace component on the bootstrap AGAS
locality.
0000000100000001000000000000000003
Address of the symbol_namespace component on the bootstrap AGAS
locality.
0000000100000001000000000000000004
Address of the locality_namespace component on the bootstrap AGAS
locality.
```

Variables

```
constexpr char const *const primary_namespace_service_name = "primary/"

struct primary_namespace : public components::fixed_component_base<primary_namespace>
```

Public Types

```
using mutex_type = hpx::spinlock

using base_type = components::fixed_component_base<primary_namespace>

using component_type = std::int32_t

using gva_table_data_type = std::pair<gva, naming::gid_type>

using gva_table_type = std::map<naming::gid_type, gva_table_data_type>

using refcnt_table_type = std::map<naming::gid_type, std::int64_t>
```

```
using resolved_type = hpx::tuple<naming::gid_type, gva, naming::gid_type>
```

Public Functions

```
primary_namespace()

void finalize()

void set_local_locality(naming::gid_type const &g)

void register_server_instance(char const *servicename, std::uint32_t locality_id
    = naming::invalid_locality_id, error_code &ec =
    throws)

void unregister_server_instance(error_code &ec = throws)

bool bind_gid(gva const &g, naming::gid_type id, naming::gid_type const &locality)
    std::pair<hpx::id_type, naming::address> begin_migration(naming::gid_type id)

bool end_migration(naming::gid_type const &id)

resolved_type resolve_gid(naming::gid_type const &id)

hpx::id_type colocate(naming::gid_type const &id)

naming::address unbind_gid(std::uint64_t count, naming::gid_type id)

std::int64_t increment_credit(std::int64_t credits, naming::gid_type lower, naming::gid_type upper)

std::vector<std::int64_t> decrement_credit(std::vector<hpx::tuple<std::int64_t, naming::gid_type, naming::gid_type>> const
    &requests)

std::pair<naming::gid_type, naming::gid_type> allocate(std::uint64_t count)
```

Public Members

```
counter_data counter_data_
```

Private Types

```
using migration_table_type = std::map<naming::gid_type, hpx::tuple<bool, std::size_t, lcos::local::data>>
using free_entry_allocator_type = util::internal_allocator<free_entry>
using free_entry_list_type = std::list<free_entry, free_entry_allocator_type>
```

Private Functions

```
void wait_for_migration_locked(std::unique_lock<mutex_type> &l, naming::gid_type const &id, error_code &ec)

resolved_type resolve_gid_locked(std::unique_lock<mutex_type> &l, naming::gid_type const &gid, error_code &ec)

void increment (naming::gid_type const &lower, naming::gid_type const &upper,
                std::int64_t &credits, error_code &ec)

void resolve_free_list (std::unique_lock<mutex_type>
                        &l,
                        std::list<refcnt_table_type::iterator> const &free_list,
                        free_entry_list_type &free_entry_list, naming::gid_type
                        const &lower, naming::gid_type const &upper, error_code &ec)

void decrement_sweep (free_entry_list_type &free_list, naming::gid_type const
                      &lower, naming::gid_type const &upper, std::int64_t credits,
                      error_code &ec)

void free_components_sync (free_entry_list_type &free_list, naming::gid_type const
                           &lower, naming::gid_type const &upper, error_code
                           &ec)
```

Private Members

```
mutex_type mutex_
gva_table_type gvas_
refcnt_table_type refcnts_
std::string instance_name_
naming::gid_type next_id_
naming::gid_type locality_
migration_table_type migrating_objects_

struct counter_data
```

Public Functions

```
HPX_NON_COPYABLE (counter_data)

counter_data()

std::int64_t get_bind_gid_count (bool)

std::int64_t get_resolve_gid_count (bool)

std::int64_t get_unbind_gid_count (bool)

std::int64_t get_increment_credit_count (bool)

std::int64_t get_decrement_credit_count (bool)

std::int64_t get_allocate_count (bool)
```

```
std::int64_t get_begin_migration_count (bool)
std::int64_t get_end_migration_count (bool)
std::int64_t get_overall_count (bool)
std::int64_t get_bind_gid_time (bool)
std::int64_t get_resolve_gid_time (bool)
std::int64_t get_unbind_gid_time (bool)
std::int64_t get_increment_credit_time (bool)
std::int64_t get_decrement_credit_time (bool)
std::int64_t get_allocate_time (bool)
std::int64_t get_begin_migration_time (bool)
std::int64_t get_end_migration_time (bool)
std::int64_t get_overall_time (bool)
void increment_bind_gid_count ()
void increment_resolve_gid_count ()
void increment_unbind_gid_count ()
void increment_increment_credit_count ()
void increment_decrement_credit_count ()
void increment_allocate_count ()
void increment_begin_migration_count ()
void increment_end_migration_count ()
void enable_all ()
```

Public Members

```
api_counter_data bind_gid_
api_counter_data resolve_gid_
api_counter_data unbind_gid_
api_counter_data increment_credit_
api_counter_data decrement_credit_
api_counter_data allocate_
api_counter_data begin_migration_
api_counter_data end_migration_
struct api_counter_data
```

Public Functions

```
api_counter_data()
```

Public Members

```
std::atomic<std::int64_t> count_
std::atomic<std::int64_t> time_
bool enabled_

struct free_entry
```

Public Functions

```
free_entry(agas::gva gva, naming::gid_type const &gid, naming::gid_type const
&loc)
```

Public Members

```
agas::gva gva_
naming::gid_type gid_
naming::gid_type locality_
```

async_colocated

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/async_colocated/get_colocation_id.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
hpx::id_type get_colocation_id(launch::sync_policy, hpx::id_type const &id, error_code &ec =
throws)
```

Return the id of the locality where the object referenced by the given id is currently located on.

The function `hpx::get_colocation_id()` returns the id of the locality where the given object is currently located.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

See `hpx::get_colocation_id()`

Parameters

- `id`: [in] The id of the object to locate.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`hpx::future<hpx::id_type> get_colocation_id(hpx::id_type const &id)`

Asynchronously return the id of the locality where the object referenced by the given id is currently located on.

See `hpx::get_colocation_id(launch::sync_policy)`

Parameters

- `id`: [in] The id of the object to locate.

async_distributed

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/async_distributed/base_lco.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
template<>
struct get_lva<lcos::base_lco>
```

Public Static Functions

```
static constexpr lcos::base_lco *call(naming::address_type lva)
```

```
template<>
struct get_lva<lcos::base_lco const>
```

Public Static Functions

```
static constexpr lcos::base_lco const *call(naming::address_type lva)
```

```
namespace hpx
```

```
template<>
struct get_lva<lcos::base_lco>
```

Public Static Functions

```
static constexpr lcos::base_lco *call (naming::address_type lva)
template<>
struct get_lva<lcos::base_lco const>
```

Public Static Functions

```
static constexpr lcos::base_lco const *call (naming::address_type lva)
namespace lcos

class base_lco
    #include <base_lco.hpp> The base_lco class is the common base class for all LCO's implementing a simple set_event action
    Subclassed by hpx::lcos::base_lco_with_value< Result, RemoteResult, ComponentTag >, hpx::lcos::base_lco_with_value< void, void, ComponentTag >
```

Public Types

```
typedef components::managed_component<base_lco> wrapping_type
typedef base_lco base_type_holder
```

Public Functions

```
virtual void set_event () = 0
virtual void set_exception (std::exception_ptr const &e)
virtual void connect (hpx::id_type const&)
virtual void disconnect (hpx::id_type const&)
virtual ~base_lco ()
    Destructor, needs to be virtual to allow for clean destruction of derived objects
virtual void finalize ()
    finalize() will be called just before the instance gets destructed
void set_event_nonvirt ()
    The function set_event_nonvirt is called whenever a set_event_action is applied on a instance of a LCO. This function just forwards to the virtual function set_event, which is overloaded by the derived concrete LCO.
void set_exception_nonvirt (std::exception_ptr const &e)
    The function set_exception is called whenever a set_exception_action is applied on a instance of a LCO. This function just forwards to the virtual function set_exception, which is overloaded by the derived concrete LCO.
```

Parameters

- `e`: [in] The exception encapsulating the error to report to this LCO instance.

```
void connect_nonvirt (hpx::id_type const &id)
```

The function `connect_nonvirt` is called whenever a `connect_action` is applied on a instance of a LCO. This function just forwards to the virtual function `connect`, which is overloaded by the derived concrete LCO.

Parameters

- `id`: [in] target id

```
void disconnect_nonvirt (hpx::id_type const &id)
```

The function `disconnect_nonvirt` is called whenever a `disconnect_action` is applied on a instance of a LCO. This function just forwards to the virtual function `disconnect`, which is overloaded by the derived concrete LCO.

Parameters

- `id`: [in] target id

```
HPX_DEFINE_COMPONENT_DIRECT_ACTION (base_lco, set_event_nonvirt,  
                                     set_event_action)
```

Each of the exposed functions needs to be encapsulated into an action type, allowing to generate all required boilerplate code for threads, serialization, etc.

The `set_event_action` may be used to unconditionally trigger any LCO instances, it carries no additional parameters. The `set_exception_action` may be used to transfer arbitrary error information from the remote site to the LCO instance specified as a continuation. This action carries 2 parameters:

Parameters

- `std::exception_ptr`: [in] The exception encapsulating the error to report to this LCO instance.

```
set_exception_action HPX_DEFINE_COMPONENT_DIRECT_ACTION (base_lco, con-  
                                                       nect_nonvirt, con-  
                                                       nect_action)
```

The `connect_action` may be used to.

The `set_exception_action` may be used to

Public Members

```
hpx::lcos::base_lco::set_exception_nonvirt  
set_exception_action disconnect_nonvirt
```

Public Static Functions

```
static components::component_type get_component_type()

static void set_component_type(components::component_type type)
```

hpx/async_distributed/base_lco_with_value.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION(...)

HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_(...)

HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION2 (Value, RemoteValue, Name)
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_1 (Value)
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_2 (Value, Name)
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_3 (Value, RemoteValue, Name)
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_4 (Value, RemoteValue, Name, Tag)
HPX_REGISTER_BASE_LCO_WITH_VALUE (...)

HPX_REGISTER_BASE_LCO_WITH_VALUE_(...)

HPX_REGISTER_BASE_LCO_WITH_VALUE_1 (Value)
HPX_REGISTER_BASE_LCO_WITH_VALUE_2 (Value, Name)
HPX_REGISTER_BASE_LCO_WITH_VALUE_3 (Value, RemoteValue, Name)
HPX_REGISTER_BASE_LCO_WITH_VALUE_4 (Value, RemoteValue, Name, Tag)
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID (...)

HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_(...)

HPX_REGISTER_BASE_LCO_WITH_VALUE_ID2 (Value, RemoteValue, Name, ActionIdGet, ActionIdSet)
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_4 (Value, Name, ActionIdGet, ActionIdSet)
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_5 (Value, RemoteValue, Name, ActionIdGet, ActionIdSet)
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_6 (Value, RemoteValue, Name, ActionIdGet, ActionIdSet, Tag)

namespace hpx

namespace lcos

template<typename Result, typename RemoteResult, typename ComponentTag>
```

```
class base_lco_with_value : public hpx::lcos::base_lco, public ComponentTag
    #include <base_lco_with_value.hpp> The base_lco_with_value class is the common base class for
    all LCO's synchronizing on a value. The RemoteResult template argument should be set to the type
    of the argument expected for the set_value action.
```

Template Parameters

- `RemoteResult`: The type of the result value to be carried back to the LCO instance.
 - `ComponentTag`: The tag type representing the type of the component (either `component_tag` or `managed_component_tag`).

Public Types

```
template<>
using wrapping_type = typename detail::base_lco_wrapping_type<ComponentTag, base_lco_with_value>;  
template<>
using base_type_holder = base_lco_with_value
```

Public Functions

```
void set_value_nonvirt (RemoteResult &&result)
```

The `function set_value_nonvirt` is called whenever a `set_value_action` is applied on this LCO instance. This function just forwards to the virtual function `set_value`, which is overloaded by the derived concrete LCO.

Parameters

- **result**: [in] The result value to be transferred from the remote operation back to this LCO instance.

Result **get_value_nonvirt** ()

The `function get_result_nonvirt` is called whenever a `get_result_action` is applied on this LCO instance. This function just forwards to the virtual function `get_result`, which is overloaded by the derived concrete LCO.

Hpx_Define_Component_Direct_Action(base_lco_with_value, set_value_nonvirt,
set value action)

The `set_value_action` may be used to trigger any LCO instances while carrying an additional parameter of any type.

`RemoteResult` is taken by rvalue ref. This allows for perfect forwarding. When the action thread function is created, the values are moved into the called function. If we took it by const lvalue reference, we would disable the possibility to further move the result to the designated destination.

Parameters

- **RemoteResult**: [in] The type of the result to be transferred back to this LCO instance. The `get_value_action` may be used to query the value this LCO instance exposes as its ‘result’ value.

Public Members

```
hpx::lcos::base_lco_with_value::get_value_nonvirt
```

Public Static Functions

```
static components::component_type get_component_type()
static void set_component_type (components::component_type type)
```

Protected Types

```
template<>
using result_type = std::conditional_t<std::is_void_v<Result>, util::unused_type, Result>
```

Protected Functions

```
~base_lco_with_value()
Destructor, needs to be virtual to allow for clean destruction of derived objects

void set_event()
void set_event_nonvirt (std::false_type)
void set_event_nonvirt (std::true_type)

virtual void set_value (RemoteResult &&result) = 0
virtual result_type get_value () = 0
virtual result_type get_value (error_code&)

template<typename ComponentTag>
class base_lco_with_value<void, void, ComponentTag> : public hpx::lcos::base_lco, public ComponentTag
#include <base_lco_with_value.hpp> The base_lco<void> specialization is used whenever the
set_event action for a particular LCO doesn't carry any argument.
```

Template Parameters

- void: This specialization expects no result value and is almost completely equivalent to the plain `base_lco`.

Public Types

```
template<>
using wrapping_type = typename detail::base_lco_wrapping_type<ComponentTag, base_lco_with_value>::type

template<>
using base_type_holder = base_lco_with_value

template<>
using set_value_action = typename base_lco::set_event_action
```

Public Functions

```
void get_value()
```

Protected Functions

```
~base_lco_with_value()
```

Destructor, needs to be virtual to allow for clean destruction of derived objects

hpx/async_distributed/lcos_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace distributed
```

```
template<typename Result, typename RemoteResult>
class promise
```

`#include <promise.hpp>` A promise can be used by a single *thread* to invoke a (remote) action and wait for the result. The result is expected to be sent back to the promise using the LCO's set_event action

A promise is one of the simplest synchronization primitives provided by HPX. It allows to synchronize on a eager evaluated remote operation returning a result of the type *Result*. The *promise* allows to synchronize exactly one *thread* (the one passed during construction time).

```
// Create the promise (the expected result is a id_type)
hpx::distributed::promise<hpx::id_type> p;

// Get the associated future
future<hpx::id_type> f = p.get_future();

// initiate the action supplying the promise as a
// continuation
apply<some_action>(new continuation(p.get_id()), ...);

// Wait for the result to be returned, yielding control
// in the meantime.
hpx::id_type result = f.get();
// ...
```

Note The action executed by the promise must return a value of a type convertible to the type as specified by the template parameter *RemoteResult*

Template Parameters

- *Result*: The template parameter *Result* defines the type this promise is expected to return from `promise::get`.
- *RemoteResult*: The template parameter *RemoteResult* defines the type this promise is expected to receive from the remote action.

```
namespace lcos
```

Typedefs

```
using instead = hpx::distributed::promise<Result, RemoteResult>
template<typename Action, typename Result = typename traits::promise_local_result<typename Action::remote_r
class packaged_action
```

#include <packaged_action.hpp> A *packaged_action* can be used by a single *thread* to invoke a (remote) action and wait for the result. The result is expected to be sent back to the *packaged_action* using the LCO's set_event action

A *packaged_action* is one of the simplest synchronization primitives provided by HPX. It allows to synchronize on a eager evaluated remote operation returning a result of the type *Result*.

Note The action executed using the *packaged_action* as a continuation must return a value of a type convertible to the type as specified by the template parameter *Result*.

Template Parameters

- *Action*: The template parameter *Action* defines the action to be executed by this *packaged_action* instance. The arguments *arg0, ..., argN* are used as parameters for this action.
- *Result*: The template parameter *Result* defines the type this *packaged_action* is expected to return from its associated future *packaged_action::get_future*.
- *DirectExecute*: The template parameter *DirectExecute* is an optimization aid allowing to execute the action directly if the target is local (without spawning a new thread for this). This template does not have to be supplied explicitly as it is derived from the template parameter *Action*.

hpx/async_distributed/packaged_action.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace lcos
```

```
template<typename Action, typename Result>
class packaged_action<Action, Result, false> : public hpx::distributed::promise<Result, hpx::traits::extract_acti
Subclassed by hpx::lcos::packaged_action<Action, Result, true>
```

Public Functions

```
packaged_action()

template<typename Allocator>
packaged_action(std::allocator_arg_t, Allocator const &alloc)

template<typename ...Ts>
void apply(hpx::id_type const &id, Ts&&... vs)

template<typename ...Ts>
void apply(naming::address &&addr, hpx::id_type const &id, Ts&&... vs)

template<typename Callback, typename ...Ts>
void apply_cb(hpx::id_type const &id, Callback &&cb, Ts&&... vs)
```

```
template<typename Callback, typename ...Ts>
void apply_cb(naming::address &&addr, hpx::id_type const &id, Callback &&cb, Ts&&... vs)
```

```
template<typename ...Ts>
void apply_p(hpx::id_type const &id, threads::thread_priority priority, Ts&&... vs)
```

```
template<typename ...Ts>
void apply_p(naming::address &&addr, hpx::id_type const &id, threads::thread_priority priority, Ts&&... vs)
```

```
template<typename Callback, typename ...Ts>
void apply_p_cb(hpx::id_type const &id, threads::thread_priority priority, Callback &&cb, Ts&&... vs)
```

```
template<typename Callback, typename ...Ts>
void apply_p_cb(naming::address &&addr, hpx::id_type const &id, threads::thread_priority &id, threads::thread_priority priority, Callback &&cb, Ts&&... vs)
```

```
template<typename ...Ts>
void apply_deferred(naming::address &&addr, hpx::id_type const &id, Ts&&... vs)
```

```
template<typename Callback, typename ...Ts>
void apply_deferred_cb(naming::address &&addr, hpx::id_type const &id, Callback &&cb, Ts&&... vs)
```

Protected Types

```
template<>
using action_type = typename hpx::traits::extract_action<Action>::type
```

```
template<>
using remote_result_type = typename action_type::remote_result_type
```

```
template<>
using base_type = hpx::distributed::promise<Result, remote_result_type>
```

Protected Functions

```
template<typename ...Ts>
void do_apply(naming::address &&addr, hpx::id_type const &id, threads::thread_priority priority, Ts&&... vs)
```

```
template<typename ...Ts>
void do_apply(hpx::id_type const &id, threads::thread_priority priority, Ts&&... vs)
```

```
template<typename Callback, typename ...Ts>
void do_apply_cb(naming::address &&addr, hpx::id_type const &id, threads::thread_priority priority, Callback &&cb, Ts&&... vs)
```

```
template<typename Callback, typename ...Ts>
void do_apply_cb(hpx::id_type const &id, threads::thread_priority priority, Callback &&cb, Ts&&... vs)
```

```
template<typename Action, typename Result>
class packaged_action<Action, Result, true> : public hpx::lcos::packaged_action<Action, Result, false>
```

Public Functions

```
packaged_action()
    Construct a (non-functional) instance of an packaged_action. To use this instance its member function apply needs to be directly called.

template<typename Allocator>
packaged_action (std::allocator_arg_t, Allocator const &alloc)

template<typename ...Ts>
void apply (hpx::id_type const &id, Ts&&... vs)

template<typename ...Ts>
void apply (naming::address &&addr, hpx::id_type const &id, Ts&&... vs)

template<typename Callback, typename ...Ts>
void apply_cb (hpx::id_type const &id, Callback &&cb, Ts&&... vs)

template<typename Callback, typename ...Ts>
void apply_cb (naming::address &&addr, hpx::id_type const &id, Callback &&cb, Ts&&... vs)
```

Private Types

```
template<>
using action_type = typename packaged_action::action_type
```

hpx/async_distributed/promise.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
template<>
class promise<void, hpx::util::unused_type> : public lcos::detail::promise_base<void, hpx::util::unused_type, lcos::detail::pron
```

Public Functions

```
promise()
    constructs a promise object and a shared state.

template<typename Allocator>
promise (std::allocator_arg_t, Allocator const &a)
    constructs a promise object and a shared state. The constructor uses the allocator a to allocate the memory for the shared state.

promise (promise &&other)
    constructs a new promise object and transfers ownership of the shared state of other (if any) to the newly-constructed object.
```

Post other has no shared state.

```
~promise()
    Abandons any shared state.
```

```
promise &operator=(promise &&other)
    Abandons any shared state (30.6.4) and then as if promise(HPX_MOVE(other)).swap(*this).
```

Return *this.

```
void swap(promise &other)
    Exchanges the shared state of *this and other.
```

Post *this has the shared state (if any) that other had prior to the call to swap. other has the shared state (if any) that *this had prior to the call to swap.

```
void set_value()
    atomically stores the value r in the shared state and makes that state ready (30.6.4).
```

Exceptions

- `future_error`: if its shared state already has a stored value. if shared state has no stored value exception is raised. `promise_already_satisfied` if its shared state already has a stored value or exception. `no_state` if *this has no shared state.

Private Types

```
template<>
using base_type = lcos::detail::promise_base<void, hpx::util::unused_type, lcos::detail::promise_data<void>>

template<typename R, typename Allocator>
struct uses_allocator<hpx::distributed::promise<R>, Allocator> : public true_type
    #include <promise.hpp> Requires: Allocator shall be an allocator (17.6.3.5)

namespace hpx
```

```
namespace distributed
```

Functions

```
template<typename Result, typename RemoteResult>
void swap(promise<Result, RemoteResult> &x, promise<Result, RemoteResult> &y)

template<>
class promise<void, hpx::util::unused_type> : public lcos::detail::promise_base<void, hpx::util::unused_type, lcos::
```

Public Functions

promise()

constructs a promise object and a shared state.

template<typename **Allocator**>

promise (*std*::allocator_arg_t, *Allocator* const &*a*)

constructs a promise object and a shared state. The constructor uses the allocator *a* to allocate the memory for the shared state.

promise (*promise* &&*other*)

constructs a new promise object and transfers ownership of the shared state of *other* (if any) to the newly-constructed object.

Post *other* has no shared state.

~promise()

Abandons any shared state.

promise &operator= (*promise* &&*other*)

Abandons any shared state (30.6.4) and then as if *promise*(HPX_MOVE(*other*)).swap(*this).

Return *this.

void swap (*promise* &*other*)

Exchanges the shared state of *this and *other*.

Post *this has the shared state (if any) that *other* had prior to the call to swap. *other* has the shared state (if any) that *this had prior to the call to swap.

void set_value()

atomically stores the value *r* in the shared state and makes that state ready (30.6.4).

Exceptions

- *future_error*: if its shared state already has a stored value. if shared state has no stored value exception is raised. *promise_already_satisfied* if its shared state already has a stored value or exception. *no_state* if *this has no shared state.

Private Types

```
template<>
using base_type = lcos::detail::promise_base<void, hpx::util::unused_type, lcos::detail::promise_data<void>>

namespace std

template<typename R, typename Allocator>
struct uses_allocator<hpx::distributed::promise<R>, Allocator> : public true_type
#include <promise.hpp> Requires: Allocator shall be an allocator (17.6.3.5)
```

hpx/async_distributed/transfer_continuation_action.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/async_distributed/trigger_lco.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/async_distributed/trigger_lco_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

```
void trigger_lco_event (hpx::id_type const &id, naming::address &&addr, bool move_credits = true)
```

Trigger the LCO referenced by the given id.

Parameters

- *id*: [in] This represents the id of the LCO which should be triggered.
- *addr*: [in] This represents the addr of the LCO which should be triggered.
- *move_credits*: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void trigger_lco_event (hpx::id_type const &id, bool move_credits = true)
```

Trigger the LCO referenced by the given id.

Parameters

- *id*: [in] This represents the id of the LCO which should be triggered.
- *move_credits*: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void trigger_lco_event (hpx::id_type const &id, naming::address &&addr, hpx::id_type const &cont, bool move_credits = true)
```

Trigger the LCO referenced by the given id.

Parameters

- *id*: [in] This represents the id of the LCO which should be triggered.
- *addr*: [in] This represents the addr of the LCO which should be triggered.
- *cont*: [in] This represents the LCO to trigger after completion.
- *move_credits*: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void trigger_lco_event(hpx::id_type const &id, hpx::id_type const &cont, bool move_credits
                      = true)
Trigger the LCO referenced by the given id.
```

Parameters

- *id*: [in] This represents the id of the LCO which should be triggered.
- *cont*: [in] This represents the LCO to trigger after completion.
- *move_credits*: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
template<typename Result>
void set_lco_value(hpx::id_type const &id, naming::address &&addr, Result &&t, bool
                   move_credits = true)
Set the result value for the LCO referenced by the given id.
```

Parameters

- *id*: [in] This represents the id of the LCO which should receive the given value.
- *addr*: [in] This represents the addr of the LCO which should be triggered.
- *t*: [in] This is the value which should be sent to the LCO.
- *move_credits*: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
template<typename Result>
std::enable_if<!std::is_same<typename std::decay<Result>::type, naming::address>::value>::type set_lco_value(hpx::id
                                                 const &id,
                                                 Re-
                                                 sult
                                                 &&t,
                                                 bool
                                                 move_c
                                                 =
                                                 true)
```

Set the result value for the (managed) LCO referenced by the given id.

Parameters

- *id*: [in] This represents the id of the LCO which should receive the given value.
- *t*: [in] This is the value which should be sent to the LCO.
- *move_credits*: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
template<typename Result>
```

```
std::enable_if<!std::is_same<typename std::decay<Result>::type, naming::address>::value>::type set_lco_value_unman
```

Set the result value for the (unmanaged) LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the given value.
- `t`: [in] This is the value which should be sent to the LCO.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
template<typename Result>
void set_lco_value(hpx::id_type const &id, naming::address &&addr, Result &&t, hpx::id_type
const &cont, bool move_credits = true)
```

Set the result value for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the given value.
- `addr`: [in] This represents the addr of the LCO which should be triggered.
- `t`: [in] This is the value which should be sent to the LCO.
- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
template<typename Result>
std::enable_if<!std::is_same<typename std::decay<Result>::type, naming::address>::value>::type set_lco_value(hpx::id_
const
&id,
Re-
sult
&&t,
hpx::id_
const
&cont,
bool
move_c
=
true)
```

Set the result value for the (managed) LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the given value.
- `t`: [in] This is the value which should be sent to the LCO.
- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to `true` then it is ok to send all credits in `id` along with the generated message. The default value is `true`.

```
template<typename Result>
std::enable_if<!std::is_same<typename std::decay<Result>::type, naming::address>::value>::type set_lco_value_unman-
```

Set the result value for the (unmanaged) LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the given value.
- `t`: [in] This is the value which should be sent to the LCO.
- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to `true` then it is ok to send all credits in `id` along with the generated message. The default value is `true`.

```
void set_lco_error(hpx::id_type const &id, naming::address &&addr, std::exception_ptr const &e,
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `addr`: [in] This represents the addr of the LCO which should be triggered.
- `e`: [in] This is the error value which should be sent to the LCO.
- `move_credits`: [in] If this is set to `true` then it is ok to send all credits in `id` along with the generated message. The default value is `true`.

```
void set_lco_error(hpx::id_type const &id, naming::address &&addr, std::exception_ptr &&e,
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.

- `addr`: [in] This represents the addr of the LCO which should be triggered.
- `e`: [in] This is the error value which should be sent to the LCO.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void set_lco_error(hpx::id_type const &id, std::exception_ptr const &e, bool move_credits = true)
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `e`: [in] This is the error value which should be sent to the LCO.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void set_lco_error(hpx::id_type const &id, std::exception_ptr &&e, bool move_credits = true)
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `e`: [in] This is the error value which should be sent to the LCO.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void set_lco_error(hpx::id_type const &id, naming::address &&addr, std::exception_ptr const &e, hpx::id_type const &cont, bool move_credits = true)
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `addr`: [in] This represents the addr of the LCO which should be triggered.
- `e`: [in] This is the error value which should be sent to the LCO.
- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void set_lco_error(hpx::id_type const &id, naming::address &&addr, std::exception_ptr &&e, hpx::id_type const &cont, bool move_credits = true)
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `addr`: [in] This represents the addr of the LCO which should be triggered.
- `e`: [in] This is the error value which should be sent to the LCO.

- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void set_lco_error(hpx::id_type const &id, std::exception_ptr const &e, hpx::id_type const&cont, bool move_credits = true)
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `e`: [in] This is the error value which should be sent to the LCO.
- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

```
void set_lco_error(hpx::id_type const &id, std::exception_ptr &&e, hpx::id_type const &cont, bool move_credits = true)
```

Set the error state for the LCO referenced by the given id.

Parameters

- `id`: [in] This represents the id of the LCO which should receive the error value.
- `e`: [in] This is the error value which should be sent to the LCO.
- `cont`: [in] This represents the LCO to trigger after completion.
- `move_credits`: [in] If this is set to *true* then it is ok to send all credits in *id* along with the generated message. The default value is *true*.

checkpoint

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/checkpoint/checkpoint.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

This header defines the `save_checkpoint` and `restore_checkpoint` functions. These functions are designed to help HPX application developer's checkpoint their applications. `Save_checkpoint` serializes one or more objects and saves them as a byte stream. `Restore_checkpoint` converts the byte stream back into instances of the objects.

```
namespace hpx
```

```
namespace util
```

Functions

```
std::ostream &operator<< (std::ostream &ost, checkpoint const &ckp)
    Operator<< Overload
```

This overload is the main way to write data from a checkpoint to an object such as a file. Inside the function, the size of the checkpoint will be written to the stream before the checkpoint's data. The operator>> overload uses this to read the correct number of bytes. Be mindful of this additional write and read when you use different facilities to write out or read in data to a checkpoint!

Parameters

- `ost`: Output stream to write to.
- `ckp`: Checkpoint to copy from.

Return Operator<< returns the ostream object.

```
std::istream &operator>> (std::istream &ist, checkpoint &ckp)
    Operator>> Overload
```

This overload is the main way to read in data from an object such as a file to a checkpoint. It is important to note that inside the function, the first variable to be read is the size of the checkpoint. This size variable is written to the stream before the checkpoint's data in the operator<< overload. Be mindful of this additional read and write when you use different facilities to read in or write out data from a checkpoint!

Parameters

- `ist`: Input stream to write from.
- `ckp`: Checkpoint to write to.

Return Operator>> returns the ostream object.

```
template<typename T, typename ...Ts, typename U = typename std::enable_if<!hpx::traits::is_launch_policy<T>::value &
hp::future<checkpoint> save_checkpoint (T &&t, Ts&&... ts)
    Save_checkpoint
```

`Save_checkpoint` takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a `shared_ptr` to the component or by passing a component's client instance to `save_checkpoint`. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used `save_checkpoint` will simply return a checkpoint object.

Template Parameters

- `T`: Containers passed to `save_checkpoint` to be serialized and placed into a checkpoint object.
- `Ts`: More containers passed to `save_checkpoint` to be serialized and placed into a checkpoint object.
- `U`: This parameter is used to make sure that `T` is not a launch policy or a checkpoint. This forces the compiler to choose the correct overload.

Parameters

- `t`: A container to restore.
- `ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Return `Save_checkpoint` returns a future to a checkpoint with one exception: if you pass

hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```
template<typename T, typename ...Ts>
hpx::future<checkpoint> save_checkpoint (checkpoint &&c, T &&t, Ts&&... ts)
    Save_checkpoint - Take a pre-initialized checkpoint
```

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component's client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

Template Parameters

- T: Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- Ts: More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.

Parameters

- c: Takes a pre-initialized checkpoint to copy data into.
- t: A container to restore.
- ts: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

Return Save_checkpoint returns a future to a checkpoint with one exception: if you pass hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```
template<typename T, typename ...Ts, typename U = typename std::enable_if<!std::is_same<typename std::decay<T>
```

hpx::future<checkpoint> **save_checkpoint** (hpx::launch p, T &&t, Ts&&... ts)

Save_checkpoint - Policy overload

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component's client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

Template Parameters

- T: Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- Ts: More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.

Parameters

- p: Takes an HPX launch policy. Allows the user to change the way the function is launched i.e. async, sync, etc.
- t: A container to restore.
- ts: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

Return Save_checkpoint returns a future to a checkpoint with one exception: if you pass hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```
template<typename T, typename ...Ts>
```

```
hpx::future<checkpoint> save_checkpoint (hpx::launch p, checkpoint &&c, T &&t, Ts&&...  
                                         ts)  
Save_checkpoint - Policy overload & pre-initialized checkpoint
```

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component's client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

Template Parameters

- T: Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- Ts: More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.

Parameters

- p: Takes an HPX launch policy. Allows the user to change the way the function is launched i.e. async, sync, etc.
- c: Takes a pre-initialized checkpoint to copy data into.
- t: A container to restore.
- ts: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

Return Save_checkpoint returns a future to a checkpoint with one exception: if you pass hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```
template<typename T, typename ...Ts, typename U = typename std::enable_if<!std::is_same<typename std::decay<T>  
checkpoint save_checkpoint (hpx::launch::sync_policy sync_p, T &&t, Ts&&... ts)  
Save_checkpoint - Sync_policy overload
```

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component's client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

Template Parameters

- T: Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- Ts: More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- U: This parameter is used to make sure that T is not a checkpoint. This forces the compiler to choose the correct overload.

Parameters

- sync_p: hpx::launch::sync_policy
- t: A container to restore.
- ts: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

Return Save_checkpoint which is passed hpx::launch::sync_policy will return a checkpoint which contains the serialized values checkpoint.

```
template<typename T, typename ...Ts>  
checkpoint save_checkpoint (hpx::launch::sync_policy sync_p, checkpoint &&c, T &&t,  
                                         Ts&&... ts)  
Save_checkpoint - Sync_policy overload & pre-init. checkpoint
```

`Save_checkpoint` takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a `shared_ptr` to the component or by passing a component's client instance to `save_checkpoint`. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used `save_checkpoint` will simply return a checkpoint object.

Template Parameters

- `T`: Containers passed to `save_checkpoint` to be serialized and placed into a checkpoint object.
- `Ts`: More containers passed to `save_checkpoint` to be serialized and placed into a checkpoint object.

Parameters

- `sync_p`: `hpx::launch::sync_policy`
- `c`: Takes a pre-initialized checkpoint to copy data into.
- `t`: A container to restore.
- `ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Return `Save_checkpoint` which is passed `hpx::launch::sync_policy` will return a checkpoint which contains the serialized values `checkpoint`.

```
template<typename T, typename ...Ts, typename U = typename std::enable_if<!hpx::traits::is_launch_policy<T>::value &
hpx::future<checkpoint> prepare_checkpoint (T const &t, Ts const&... ts)
    prepare_checkpoint
```

`prepare_checkpoint` takes the containers which have to be filled from the byte stream by a subsequent `restore_checkpoint` invocation. `prepare_checkpoint` will calculate the necessary buffer size and will return an appropriately sized checkpoint object.

Return `prepare_checkpoint` returns a properly resized checkpoint object that can be used for a subsequent `restore_checkpoint` operation.

Template Parameters

- `T`: A container to restore.
- `Ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Parameters

- `t`: A container to restore.
- `ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

```
template<typename T, typename ...Ts>
hpx::future<checkpoint> prepare_checkpoint (checkpoint &&c, T const &t, Ts const&...
    ts)
    prepare_checkpoint
```

`prepare_checkpoint` takes the containers which have to be filled from the byte stream by a subsequent `restore_checkpoint` invocation. `prepare_checkpoint` will calculate the necessary buffer size and will return an appropriately sized checkpoint object.

Return `prepare_checkpoint` returns a properly resized checkpoint object that can be used for a subsequent `restore_checkpoint` operation.

Template Parameters

- `T`: A container to restore.
- `Ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Parameters

- `c`: Takes a pre-initialized checkpoint to prepare
- `t`: A container to restore.
- `ts`: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

```
template<typename T, typename ...Ts, typename U = typename std::enable_if<!std::is_same<T, checkpoint>::value>::type  
hpx::future<checkpoint> prepare_checkpoint (hpx::launch p, T const &t, Ts const&... ts)  
    prepare_checkpoint
```

`prepare_checkpoint` takes the containers which have to be filled from the byte stream by a subsequent `restore_checkpoint` invocation. `prepare_checkpoint` will calculate the necessary buffer size and will return an appropriately sized checkpoint object.

Return `prepare_checkpoint` returns a properly resized checkpoint object that can be used for a subsequent `restore_checkpoint` operation.

Template Parameters

- `T`: A container to restore.
- `Ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Parameters

- `p`: Takes an HPX launch policy. Allows the user to change the way the function is launched i.e. `async`, `sync`, etc.
- `t`: A container to restore.
- `ts`: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

```
template<typename T, typename ...Ts>  
hpx::future<checkpoint> prepare_checkpoint (hpx::launch p, checkpoint &&c, T const &t,  
                                         Ts const&... ts)  
    prepare_checkpoint
```

`prepare_checkpoint` takes the containers which have to be filled from the byte stream by a subsequent `restore_checkpoint` invocation. `prepare_checkpoint` will calculate the necessary buffer size and will return an appropriately sized checkpoint object.

Return `prepare_checkpoint` returns a properly resized checkpoint object that can be used for a subsequent `restore_checkpoint` operation.

Template Parameters

- `T`: A container to restore.
- `Ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Parameters

- `p`: Takes an HPX launch policy. Allows the user to change the way the function is launched i.e. `async`, `sync`, etc.
- `c`: Takes a pre-initialized checkpoint to prepare
- `t`: A container to restore.
- `ts`: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

```
template<typename T, typename ...Ts>  
void restore_checkpoint (checkpoint const &c, T &t, Ts&... ts)  
    Restore_checkpoint
```

`Restore_checkpoint` takes a checkpoint object as a first argument and the containers which will be filled from the byte stream (in the same order as they were placed in `save_checkpoint`). Re-

store_checkpoint can resurrect a stored component in two ways: by passing in a instance of a component's shared_ptr or by passing in an instance of the component's client.

Return Restore_checkpoint returns void.

Template Parameters

- T: A container to restore.
- Ts: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Parameters

- c: The checkpoint to restore.
- t: A container to restore.
- ts: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

```
class checkpoint
#include <checkpoint.hpp> Checkpoint Object
```

Checkpoint is the container object which is produced by save_checkpoint and is consumed by a restore_checkpoint. A checkpoint may be moved into the save_checkpoint object to write the byte stream to the pre-created checkpoint object.

Checkpoints are able to store all containers which are able to be serialized including components.

Public Types

```
using const_iterator = std::vector::const_iterator
```

Public Functions

```
checkpoint()
~checkpoint()

checkpoint(checkpoint const &c)
checkpoint(checkpoint &&c)

checkpoint(std::vector<char> const &vec)
checkpoint(std::vector<char> &&vec)

checkpoint &operator=(checkpoint const &c)
checkpoint &operator=(checkpoint &&c)

const_iterator begin() const
const_iterator end() const

std::size_t size() const

char *data()
char const *data() const
```

Private Functions

```
template<typename Archive>
void serialize (Archive &arch, const unsigned int)
```

Private Members

```
std::vector<char> data_
```

Friends

```
friend hpx::util::hpx::serialization::access
std::ostream &operator<< (std::ostream &ost, checkpoint const &ckp)
    Operator<< Overload
```

This overload is the main way to write data from a checkpoint to an object such as a file. Inside the function, the size of the checkpoint will be written to the stream before the checkpoint's data. The operator>> overload uses this to read the correct number of bytes. Be mindful of this additional write and read when you use different facilities to write out or read in data to a checkpoint!

Parameters

- ost: Output stream to write to.
- ckp: Checkpoint to copy from.

Return Operator<< returns the ostream object.

```
std::istream &operator>> (std::istream &ist, checkpoint &ckp)
    Operator>> Overload
```

This overload is the main way to read in data from an object such as a file to a checkpoint. It is important to note that inside the function, the first variable to be read is the size of the checkpoint. This size variable is written to the stream before the checkpoint's data in the operator<< overload. Be mindful of this additional read and write when you use different facilities to read in or write out data from a checkpoint!

Parameters

- ist: Input stream to write from.
- ckp: Checkpoint to write to.

Return Operator>> returns the ostream object.

```
template<typename T, typename ...Ts>
void restore_checkpoint (checkpoint const &c, T &t, Ts&... ts)
    Restore_checkpoint
```

Restore_checkpoint takes a checkpoint object as a first argument and the containers which will be filled from the byte stream (in the same order as they were placed in save_checkpoint). Restore_checkpoint can resurrect a stored component in two ways: by passing in a instance of a component's shared_ptr or by passing in an instance of the component's client.

Return Restore_checkpoint returns void.

Template Parameters

- `T`: A container to restore.
- `Ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

Parameters

- `c`: The checkpoint to restore.
- `t`: A container to restore.
- `ts`: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

```
bool operator== (checkpoint const &lhs, checkpoint const &rhs)
```

```
bool operator!= (checkpoint const &lhs, checkpoint const &rhs)
```

checkpoint_base

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/checkpoint_base/checkpoint_data.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace util
```

Functions

```
template<typename Container, typename ...Ts>
void save_checkpoint_data (Container &data, Ts&&... ts)
    save_checkpoint_data
```

`Save_checkpoint_data` takes any number of objects which a user may wish to store in the given container.

Template Parameters

- `Container`: Container used to store the check-pointed data.
- `Ts`: Types of variables to checkpoint

Parameters

- `cont`: Container instance used to store the checkpoint data
- `ts`: Variable instances to be inserted into the checkpoint.

```
template<typename ...Ts>
std::size_t prepare_checkpoint_data (Ts const&... ts)
    prepare_checkpoint_data
```

`prepare_checkpoint_data` takes any number of objects which a user may wish to store in a subsequent `save_checkpoint_data` operation. The function will return the number of bytes necessary to store the data that will be produced.

Template Parameters

- `Ts`: Types of variables to checkpoint

Parameters

- `ts`: Variable instances to be inserted into the checkpoint.

```
template<typename Container, typename ...Ts>
void restore_checkpoint_data(Container const &cont, Ts&... ts)
    restore_checkpoint_data
```

`restore_checkpoint_data` takes any number of objects which a user may wish to restore from the given container. The sequence of objects has to correspond to the sequence of objects for the corresponding call to `save_checkpoint_data` that had used the given container instance.

Template Parameters

- `Container`: Container used to restore the check-pointed data.
- `Ts`: Types of variables to restore

Parameters

- `cont`: Container instance used to restore the checkpoint data
- `ts`: Variable instances to be restored from the container

collectives

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/collectives/all_gather.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
template<typename T>
hpx::future<std::vector<std::decay_t<T>>> all_gather(char const *basename, T &&result,
                                         num_sites_arg num_sites =
                                         num_sites_arg(), this_site_arg this_site =
                                         this_site_arg(), generation_arg generation =
                                         generation_arg(), root_site_arg root_site =
                                         root_site_arg())
```

AllGather a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.

It will become ready once the `all_gather` operation has been completed.

Parameters

- `basename`: The base name identifying the `all_gather` operation
- `local_result`: The value to transmit to all participating sites from this call site.
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the `all_gather` operation performed on the given base name. This is optional and needs to be supplied only if the `all_gather` operation on the given base name has to be performed more than once.

- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns. \params `root_site` The site that is responsible for creating the `all_gather` support object. This value is optional and defaults to ‘0’ (zero).

```
template<typename T>
hpx::future<std::vector<decay_t<T>>> all_gather(communicator comm, T &&result,
                                                    this_site_arg this_site =
                                                    this_site_arg())
```

AllGather a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.
It will become ready once the `all_gather` operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `local_result`: The value to transmit to all participating sites from this call site.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

hpx/collectives/all_reduce.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
template<typename T, typename F>
hpx::future<decay_t<T>> all_reduce(char const *basename, T &&result, F &&op,
                                         num_sites_arg num_sites = num_sites_arg(),
                                         this_site_arg this_site = this_site_arg(), generation_arg generation = generation_arg(),
                                         root_site_arg root_site = root_site_arg())
```

AllReduce a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.
It will become ready once the `all_reduce` operation has been completed.

Parameters

- `basename`: The base name identifying the `all_reduce` operation
- `local_result`: The value to transmit to all participating sites from this call site.
- `op`: Reduction operation to apply to all values supplied from all participating sites
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the `all_reduce` operation performed on the given base name. This is optional and needs to be supplied only if the `all_reduce` operation on the given base name has to be performed more than once.

- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns. \params `root_site` The site that is responsible for creating the all_reduce support object. This value is optional and defaults to '0' (zero).

```
template<typename T, typename F>
hpx::future<std::decay_t<T>> all_reduce(communicator comm, T &&result, F &&op,
                                              this_site_arg this_site = this_site_arg())
```

AllReduce a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.

It will become ready once the all_reduce operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `local_result`: The value to transmit to all participating sites from this call site.
- `op`: Reduction operation to apply to all values supplied from all participating sites
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

hpx/collectives/all_to_all.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
template<typename T>
hpx::future<std::vector<std::decay_t<T>>> all_to_all(char const *basename, T &&result,
                                                       num_sites_arg num_sites =
                                                       num_sites_arg(), this_site_arg this_site =
                                                       this_site_arg(), generation_arg generation =
                                                       generation_arg(), root_site_arg root_site =
                                                       root_site_arg())
```

AllToAll a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.

It will become ready once the all_to_all operation has been completed.

Parameters

- `basename`: The base name identifying the all_to_all operation
- `local_result`: The value to transmit to all participating sites from this call site.
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the all_to_all operation performed on the given base name. This is optional and needs to be supplied only if the all_to_all operation on the given base name has to be performed more than once.

- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns. \params `root_site` The site that is responsible for creating the `all_to_all` support object. This value is optional and defaults to '0' (zero).

```
template<typename T>
hpx::future<std::vector<std::decay_t<T>>> all_to_all(communicator comm, T &&  

result, this_site_arg this_site =  

this_site_arg())
```

AllToAll a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.
It will become ready once the `all_to_all` operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `local_result`: The value to transmit to all participating sites from this call site.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

hpx/collectives/argument_types.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

```
struct generation_arg
```

Public Functions

```
constexpr generation_arg (std::size_t generation = std::size_t(-1))
```

```
constexpr generation_arg &operator= (std::size_t generation)
```

```
constexpr operator std::size_t () const
```

Public Members

```
std::size_t generation
```

```
struct num_sites_arg
```

Public Functions

```
constexpr num_sites_arg (std::size_t num_sites = std::size_t(-1))  
constexpr num_sites_arg &operator=(std::size_t num_sites)  
constexpr operator std::size_t () const
```

Public Members

```
std::size_t num_sites_  
  
struct root_site_arg
```

Public Functions

```
constexpr root_site_arg (std::size_t root_site = std::size_t(0))  
constexpr root_site_arg &operator=(std::size_t root_site)  
constexpr operator std::size_t () const
```

Public Members

```
std::size_t root_site_  
  
struct tag_arg
```

Public Functions

```
constexpr tag_arg (std::size_t tag = std::size_t(0))  
constexpr tag_arg &operator=(std::size_t tag)  
constexpr operator std::size_t () const
```

Public Members

```
std::size_t tag_  
  
struct that_site_arg
```

Public Functions

```
constexpr that_site_arg (std::size_t that_site = std::size_t(-1))  
constexpr that_site_arg &operator=(std::size_t that_site)  
constexpr operator std::size_t () const
```

Public Members

```
std::size_t that_site_
struct this_site_arg
```

Public Functions

```
constexpr this_site_arg (std::size_t this_site = std::size_t(-1))
constexpr this_site_arg &operator= (std::size_t this_site)
constexpr operator std::size_t () const
```

Public Members

```
std::size_t this_site_
```

hpx/collectives/barrier.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace distributed
```

```
class barrier
```

#include <barrier.hpp> The barrier is an implementation performing a barrier over a number of participating threads. The different threads don't have to be on the same locality. This barrier can be invoked in a distributed application.

For a local only barrier

See [hpx::barrier](#).

Public Functions

```
barrier (std::string const &base_name)
```

Creates a barrier, rank is locality id, size is number of localities

A barrier *base_name* is created. It expects that `hpx::get_num_localities()` participate and the local rank is `hpx::get_locality_id()`.

Parameters

- *base_name*: The name of the barrier

```
barrier (std::string const &base_name, std::size_t num)
```

Creates a barrier with a given size, rank is locality id

A barrier *base_name* is created. It expects that *num* participate and the local rank is `hpx::get_locality_id()`.

Parameters

- `base_name`: The name of the barrier
- `num`: The number of participating threads

barrier (`std::string const &base_name, std::size_t num, std::size_t rank`)

Creates a barrier with a given size and rank

A barrier `base_name` is created. It expects that `num` participate and the local rank is `rank`.

Parameters

- `base_name`: The name of the barrier
- `num`: The number of participating threads
- `rank`: The rank of the calling site for this invocation

barrier (`std::string const &base_name, std::vector<std::size_t> const &ranks, std::size_t rank`)

Creates a barrier with a vector of ranks

A barrier `base_name` is created. It expects that `ranks.size()` and the local rank is `rank` (must be contained in `ranks`).

Parameters

- `base_name`: The name of the barrier
- `ranks`: Gives a list of participating ranks (this could be derived from a list of locality ids)
- `rank`: The rank of the calling site for this invocation

void wait ()

Wait until each participant entered the barrier. Must be called by all participants

Return This function returns once all participants have entered the barrier (have called `wait`).

hpx::future<void> wait (hpx::launch::async_policy)

Wait until each participant entered the barrier. Must be called by all participants

Return a future that becomes ready once all participants have entered the barrier (have called `wait`).

Public Static Functions

static void synchronize ()

Perform a global synchronization using the default global barrier. The barrier is created once at startup and can be reused throughout the lifetime of an HPX application.

Note This function currently does not support dynamic connection and disconnection of localities.

hpx/collectives/broadcast.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace collectives

Functions

```
template<typename T>
hpx::future<void> broadcast_to(char const *basename, T &&local_result, num_sites_arg
num_sites = num_sites_arg(), this_site_arg this_site = this_site_arg(),
generation_arg generation = generation_arg())
```

Broadcast a value to different call sites

This function sends a set of values to all call sites operating on the given base name.

Return This function returns a future that will become ready once the broadcast operation has been completed.

Parameters

- `basename`: The base name identifying the broadcast operation
- `local_result`: A value to transmit to all participating sites from this call site.
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the broadcast operation performed on the given base name. This is optional and needs to be supplied only if the broadcast operation on the given base name has to be performed more than once.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```
template<typename T>
hpx::future<void> broadcast_to(communicator comm, T &&local_result, this_site_arg
this_site = this_site_arg())
```

Broadcast a value to different call sites

This function sends a set of values to all call sites operating on the given base name.

Return This function returns a future that will become ready once the broadcast operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `local_result`: A value to transmit to all participating sites from this call site.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```
template<typename T>
hpx::future<T> broadcast_from(char const *basename, this_site_arg this_site =
this_site_arg(), generation_arg generation = generation_arg())
```

Receive a value that was broadcast to different call sites

This function sends a set of values to all call sites operating on the given base name.

Return This function returns a future holding the value that was sent to all participating sites. It will become ready once the broadcast operation has been completed.

Parameters

- `basename`: The base name identifying the broadcast operation
- `generation`: The generational counter identifying the sequence number of the broadcast operation performed on the given base name. This is optional and needs to be supplied only if the broadcast operation on the given base name has to be performed more than once.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```
template<typename T>
hpx::future<T> broadcast_from(communicator comm, this_site_arg this_site = this_site_arg())
```

Receive a value that was broadcast to different call sites

This function sends a set of values to all call sites operating on the given base name.

Return This function returns a future holding the value that was sent to all participating sites. It will become ready once the broadcast operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

hpx/collectives/broadcast_direct.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace lcos
```

Functions

```
template<typename Action, typename ArgN, ...> hpx::future<std::vector<decltype(Action)>> broadcast(global_ids, argN, ...)
```

Perform a distributed broadcast operation.

The function `hpx::lcos::broadcast` performs a distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

Return This function returns a future representing the result of the overall reduction operation.

Note If `decltype(Action(...))` is void, then the result of this function is `future<void>`.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```
template<typename Action, typename ArgN, ...>void hpx::lcos::broadcast_apply(std::vector<Action>, std::vector<ArgN>)
```

Perform an asynchronous (fire&forget) distributed broadcast operation.

The function `hpx::lcos::broadcast_apply` performs an asynchronous (fire&forget) distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```
template<typename Action, typename ArgN, ...>hpx::future< std::vector<decltype(Action)>> hpx::lcos::broadcast_with_index(std::vector<Action>, std::vector<ArgN>, size_t index)
```

Perform a distributed broadcast operation.

The function `hpx::lcos::broadcast_with_index` performs a distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

Return This function returns a future representing the result of the overall reduction operation.

Note If `decltype(Action(...))` is void, then the result of this function is `future<void>`.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```
template<typename Action, typename ArgN, ...>void hpx::lcos::broadcast_apply_with_index(std::vector<Action>, std::vector<ArgN>, size_t index)
```

Perform an asynchronous (fire&forget) distributed broadcast operation.

The function `hpx::lcos::broadcast_apply_with_index` performs an asynchronous (fire&forget) distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

hpx/collectives/channel_communicator.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace collectives`

Functions

```
hpx::future<channel_communicator> create_channel_communicator (char      const
                                                               *basename,
                                                               num_sites_arg
                                                               num_sites      =
                                                               num_sites_arg(),
                                                               this_site_arg
                                                               this_site      =
                                                               this_site_arg())
```

Create a new communicator object usable with peer-to-peer channel-based operations

This functions creates a new communicator object that can be called in order to pre-allocate a communicator object usable with multiple invocations of channel-based peer-to-peer operations.

Return This function returns a future to a new communicator object usable with the collective operation.

Parameters

- `basename`: The base name identifying the collective operation
- `num_sites`: The number of participating sites (default: all localities).
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```
channel_communicator create_channel_communicator (hpx::launch::sync_policy,
                                                       char      const      *basename,
                                                       num_sites_arg   num_sites      =
                                                       num_sites_arg(),   this_site_arg
                                                       this_site = this_site_arg())
```

Create a new communicator object usable with peer-to-peer channel-based operations

This functions creates a new communicator object that can be called in order to pre-allocate a communicator object usable with multiple invocations of channel-based peer-to-peer operations.

Return This function returns a new communicator object usable with the collective operation.

Parameters

- `basename`: The base name identifying the collective operation
- `num_sites`: The number of participating sites (default: all localities).
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```
template<typename T>
hpx::future<void> set (channel_communicator comm, that_site_arg site, T &&value, tag_arg tag = 0)
Send a value to the given site
```

This function sends a value to the given site based on the given communicator.

Return This function returns a future<void> that becomes ready once the data transfer operation has finished.

Parameters

- *comm*: The channel communicator object to use for the data transfer
- *site*: The destination site
- *value*: The value to send
- *tag*: The (optional) tag identifying the concrete operation

```
template<typename T>
hpx::future<T> get (channel_communicator comm, that_site_arg site, tag_arg tag = 0)
Send a value to the given site
```

This function receives a value from the given site based on the given communicator.

Return This function returns a future<*T*> that becomes ready once the data transfer operation has finished. The future will hold the received value.

Parameters

- *comm*: The channel communicator object to use for the data transfer
- *site*: The source site

hpx/collectives/communication_set.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace lcos

Functions

```
hpx::future<hpx::id_type> create_communication_set (char const *basename, std::size_t num_sites = std::size_t(-1),
std::size_t this_site = std::size_t(-1),
std::size_t arity = std::size_t(-1))
```

The function *create_communication_set* sets up a (distributed) tree-like communication structure that can be used with any of the collective APIs (such like *all_to_all* and similar).

Return This function returns a future holding an id_type of the communicator object to be used on the current locality.

Parameters

- *basename*: The base name identifying the all_to_all operation
- *num_sites*: The number of participating sites (default: all localities).
- *this_site*: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever *hpx::get_locality_id()* returns.

- **arity**: The number of children each of the communication nodes is connected to (default: picked based on num_sites)

hpx/collectives/create_communicator.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
communicator create_communicator(char const *basename, num_sites_arg num_sites
= num_sites_arg(), this_site_arg this_site = this_site_arg(),
generation_arg generation = generation_arg(),
root_site_arg root_site = root_site_arg())
```

Create a new communicator object usable with any collective operation

This function creates a new communicator object that can be called in order to pre-allocate a communicator object usable with multiple invocations of any of the collective operations (such as *all_gather*, *all_reduce*, *all_to_all*, *broadcast*, etc.).

Return This function returns a new communicator object usable with the collective operation.

Parameters

- **basename**: The base name identifying the collective operation
- **num_sites**: The number of participating sites (default: all localities).
- **generation**: The generational counter identifying the sequence number of the collective operation performed on the given base name. This is optional and needs to be supplied only if the collective operation on the given base name has to be performed more than once.
- **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns. \params root_site The site that is responsible for creating the collective support object. This value is optional and defaults to '0' (zero).

hpx/collectives/exclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
template<typename T, typename F>
hpx::future<std::decay_t<T>> exclusive_scan (char const *basename, T &&result, F &&op,
                                                num_sites_arg num_sites = num_sites_arg(),
                                                this_site_arg this_site = this_site_arg(), generation_arg generation = generation_arg(),
                                                root_site_arg root_site = root_site_arg())
```

Exclusive scan a set of values from different call sites

This function performs an exclusive scan operation on a set of values received from all call sites operating on the given base name.

Note The result returned on the root_site is always the same as the result returned on thus_site == 1 and is the same as the value provided by the thje root_site.

Return This function returns a future holding a vector with all values send by all participating sites. It will become ready once the exclusive_scan operation has been completed.

Parameters

- basename: The base name identifying the exclusive_scan operation
- local_result: The value to transmit to all participating sites from this call site.
- op: Reduction operation to apply to all values supplied from all participating sites
- num_sites: The number of participating sites (default: all localities).
- generation: The generational counter identifying the sequence number of the exclusive_scan operation performed on the given base name. This is optional and needs to be supplied only if the exclusive_scan operation on the given base name has to be performed more than once.
- this_site: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns. \params root_site The site that is responsible for creating the exclusive_scan support object. This value is optional and defaults to '0' (zero).

```
template<typename T, typename F>
hpx::future<std::decay_t<T>> exclusive_scan (communicator comm, T &&result, F &&op,
                                                this_site_arg this_site = this_site_arg())
```

Exclusive scan a set of values from different call sites

This function performs an exclusive scan operation on a set of values received from all call sites operating on the given base name.

Note The result returned on the root_site is always the same as the result returned on thus_site == 1 and is the same as the value provided by the thje root_site.

Return This function returns a future holding a vector with all values send by all participating sites. It will become ready once the exclusive_scan operation has been completed.

Parameters

- comm: A communicator object returned from *create_communicator*
- local_result: The value to transmit to all participating sites from this call site.
- op: Reduction operation to apply to all values supplied from all participating sites
- this_site: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

hpx/collectives/fold.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace lcos

Functions

template<typename Action, typename FoldOp, typename Init, typename ArgN, ...>hpx::lcos::fold

Perform a distributed fold operation.

The function `hpx::lcos::fold` performs a distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

Note The type of the initial value must be convertible to the result type returned from the invoked action.

Return This function returns a future representing the result of the overall folding operation.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `fold_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
- `init`: [in] The initial value to be used for the folding operation
- `argN`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

template<typename Action, typename FoldOp, typename Init, typename ArgN, ...>hpx::lcos::fold_with_index

Perform a distributed folding operation.

The function `hpx::lcos::fold_with_index` performs a distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

Note The type of the initial value must be convertible to the result type returned from the invoked action.

Return This function returns a future representing the result of the overall folding operation.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `fold_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
- `init`: [in] The initial value to be used for the folding operation
- `argN`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

```
template<typename Action, typename FoldOp, typename Init, typename ArgN, ...>hpx::f
```

Perform a distributed inverse folding operation.

The function `hpx::lcos::inverse_fold` performs an inverse distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

Note The type of the initial value must be convertible to the result type returned from the invoked action.

Return This function returns a future representing the result of the overall folding operation.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `fold_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
- `init`: [in] The initial value to be used for the folding operation
- `argN`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

```
template<typename Action, typename FoldOp, typename Init, typename ArgN, ...>hpx::f
```

Perform a distributed inverse folding operation.

The function `hpx::lcos::inverse_fold_with_index` performs an inverse distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

Note The type of the initial value must be convertible to the result type returned from the invoked action.

Return This function returns a future representing the result of the overall folding operation.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `fold_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
- `init`: [in] The initial value to be used for the folding operation
- `argN`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

hpx/collectives/gather.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace collectives

Functions

```
template<typename T>
hpx::future<std::vector<decay_t<T>>> gather_here(char const *basename, T &&result,
                                                    num_sites_arg num_sites =
                                                    num_sites_arg(), this_site_arg this_site =
                                                    this_site_arg(), generation_arg generation =
                                                    generation_arg())
```

Gather a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all gathered values. It will become ready once the gather operation has been completed.

Parameters

- **basename**: The base name identifying the gather operation
- **result**: The value to transmit to the central gather point from this call site.
- **num_sites**: The number of participating sites (default: all localities).
- **generation**: The generational counter identifying the sequence number of the gather operation performed on the given base name. This is optional and needs to be supplied only if the gather operation on the given base name has to be performed more than once.
- **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

```
template<typename T>
hpx::future<std::vector<decay_t<T>>> gather_here(communicator comm, T &&result,
                                                       this_site_arg this_site = this_site_arg())
```

Gather a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all gathered values. It will become ready once the gather operation has been completed.

Parameters

- **comm**: A communicator object returned from *create_communicator*
- **result**: The value to transmit to the central gather point from this call site.
- **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

```
template<typename T>
hpx::future<std::vector<decay_t<T>>> gather_there(char const *basename, T &&result,
                                                       this_site_arg this_site = this_site_arg(),
                                                       generation_arg generation = generation_arg(),
                                                       root_site_arg root_site = root_site_arg())
```

Gather a given value at the given call site

This function transmits the value given by *result* to a central gather site (where the corresponding *gather_here* is executed)

Return This function returns a future holding a vector with all gathered values. It will become ready once the gather operation has been completed.

Parameters

- *basename*: The base name identifying the gather operation
- *result*: The value to transmit to the central gather point from this call site.
- *generation*: The generational counter identifying the sequence number of the gather operation performed on the given base name. This is optional and needs to be supplied only if the gather operation on the given base name has to be performed more than once.
- *this_site*: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.
- *root_site*: The sequence number of the central gather point (usually the locality id). This value is optional and defaults to 0.

```
template<typename T>
hpx::future<std::vector<decay_t<T>>> gather_there(communicator comm, T &&result,
this_site_arg this_site = this_site_arg())
```

Gather a given value at the given call site

This function transmits the value given by *result* to a central gather site (where the corresponding *gather_here* is executed)

Return This function returns a future holding a vector with all gathered values. It will become ready once the gather operation has been completed.

Parameters

- *comm*: A communicator object returned from *create_communicator*
- *result*: The value to transmit to the central gather point from this call site.
- *this_site*: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

hpx/collectives/inclusive_scan.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
template<typename T, typename F>
hpx::future<std::decay_t<T>>> inclusive_scan(char const *basename, T &&result, F &&op,
num_sites_arg num_sites = num_sites_arg(),
this_site_arg this_site = this_site_arg(), gen-
eration_arg generation = generation_arg(),
root_site_arg root_site = root_site_arg())
```

Inclusive inclusive_scan a set of values from different call sites

This function performs an inclusive scan operation on a set of values received from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.
It will become ready once the inclusive_scan operation has been completed.

Parameters

- `basename`: The base name identifying the inclusive_scan operation
- `local_result`: The value to transmit to all participating sites from this call site.
- `op`: Reduction operation to apply to all values supplied from all participating sites
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the inclusive_scan operation performed on the given base name. This is optional and needs to be supplied only if the inclusive_scan operation on the given base name has to be performed more than once.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns. \params `root_site` The site that is responsible for creating the inclusive_scan support object. This value is optional and defaults to '0' (zero).

```
template<typename T, typename F>
hpx::future<std::decay_t<T>> inclusive_scan(communicator comm, T &&result, F &&op,
                                                 this_site_arg this_site = this_site_arg())
```

Inclusive inclusive_scan a set of values from different call sites

This function performs an inclusive scan operation on a set of values received from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.
It will become ready once the inclusive_scan operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `local_result`: The value to transmit to all participating sites from this call site.
- `op`: Reduction operation to apply to all values supplied from all participating sites
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

hpx/collectives/latch.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace distributed
```

```
class latch : public components::client_base<latch, hpx::lcos::server::latch>
```

Public Functions

latch ()

latch (*std::ptrdiff_t count*)
Initialize the latch

Requires: count >= 0. Synchronization: None Postconditions: counter_ == count.

latch (*hpx::id_type const &id*)
Extension: Create a client side representation for the existing *server::latch* instance with the given global id *id*.

latch (*hpx::future<hpx::id_type> &&f*)
Extension: Create a client side representation for the existing *server::latch* instance with the given global id *id*.

latch (*hpx::shared_future<hpx::id_type> const &id*)
Extension: Create a client side representation for the existing *server::latch* instance with the given global id *id*.

latch (*hpx::shared_future<hpx::id_type> &&id*)

void count_down_and_wait ()
Decrement counter_ by 1 . Blocks at the synchronization point until counter_ reaches 0.

Requires: counter_ > 0.

Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.

Exceptions

- Nothing.:

void arrive_and_wait ()
Decrement counter_ by update . Blocks at the synchronization point until counter_ reaches 0.

Requires: counter_ > 0.

Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.

Exceptions

- Nothing.:

void count_down (*std::ptrdiff_t n*)
Decrement counter_ by n. Does not block.

Requires: counter_ >= n and n >= 0.

Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true .

Exceptions

- Nothing.:

```
bool is_ready() const
    Returns: counter_ == 0. Does not block.
```

Exceptions

- Nothing.:

```
bool try_wait() const
    Returns: counter_ == 0. Does not block.
```

Exceptions

- Nothing.:

```
void wait() const
```

If counter_ is 0, returns immediately. Otherwise, blocks the calling thread at the synchronization point until counter_ reaches 0.

Exceptions

- Nothing.:

Private Types

```
typedef components::client_base<latch, hpx::lcos::server::latch> base_type
```

hpx/collectives/reduce.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace collectives
```

Functions

```
template<typename T, typename F>
hpx::future<std::decay_t<T>> reduce_here(char const *basename, T &&result, F &&op,
                                              num_sites_arg num_sites = num_sites_arg(),
                                              this_site_arg this_site = this_site_arg(), generation_arg generation = generation_arg())
```

Reduce a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Return This function returns a future holding a vector with all values send by all participating sites.
It will become ready once the all_reduce operation has been completed.

Parameters

- basename: The base name identifying the all_reduce operation
- local_result: A value to reduce on the central reduction point from this call site.
- op: Reduction operation to apply to all values supplied from all participating sites
- num_sites: The number of participating sites (default: all localities).

- **generation:** The generational counter identifying the sequence number of the all_reduce operation performed on the given base name. This is optional and needs to be supplied only if the all_reduce operation on the given base name has to be performed more than once.
- **this_site:** The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

```
template<typename T, typename F>
hpx::future<decay_t<T>> reduce_here (communicator comm, T &&local_result, F &&op,
                                              this_site_arg this_site = this_site_arg())
Reduce a set of values from different call sites
```

This function receives a set of values that are the result of applying a given operator on values supplied from all call sites operating on the given base name.

Return This function returns a future holding a value calculated based on the values send by all participating sites. It will become ready once the all_reduce operation has been completed.

Parameters

- **comm:** A communicator object returned from *create_communicator*
- **local_result:** A value to reduce on the root_site from this call site.
- **op:** Reduction operation to apply to all values supplied from all participating sites
- **this_site:** The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

```
template<typename T, typename F>
hpx::future<void> reduce_there (char const *basename, T &&result, this_site_arg this_site =
                                              this_site_arg(), generation_arg generation = generation_arg(),
                                              root_site_arg root_site = root_site_arg())
Reduce a given value at the given call site
```

This function transmits the value given by *result* to a central reduce site (where the corresponding *reduce_here* is executed)

Return This function returns a future<void>. It will become ready once the reduction operation has been completed.

Parameters

- **basename:** The base name identifying the reduction operation
- **result:** A future referring to the value to transmit to the central reduction point from this call site.
- **generation:** The generational counter identifying the sequence number of the reduction operation performed on the given base name. This is optional and needs to be supplied only if the reduction operation on the given base name has to be performed more than once.
- **this_site:** The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.
- **root_site:** The sequence number of the central reduction point (usually the locality id). This value is optional and defaults to 0.

```
template<typename T>
hpx::future<void> reduce_there (communicator comm, T &&local_result, this_site_arg
                                              this_site = this_site_arg())
Reduce a given value at the given call site
```

This function transmits the value given by *result* to a central reduce site (where the corresponding *reduce_here* is executed)

Return This function returns a future holding a value calculated based on the values send by all participating sites. It will become ready once the all_reduce operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `local_result`: A value to reduce on the central reduction point from this call site.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

hpx/collectives/reduce_direct.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace lcos`

Functions

`template<typename Action, typename ReduceOp, typename ArgN, ...>hpx::future<decaytype<`

Perform a distributed reduction operation.

The function `hpx::lcos::reduce` performs a distributed reduction operation over results returned from action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

Return This function returns a future representing the result of the overall reduction operation.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `reduce_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the reduction operation performed on its arguments.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

`template<typename Action, typename ReduceOp, typename ArgN, ...>hpx::future<decaytype<`

Perform a distributed reduction operation.

The function `hpx::lcos::reduce_with_index` performs a distributed reduction operation over results returned from action invocations on a given set of global identifiers. The action can be either plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

Return This function returns a future representing the result of the overall reduction operation.

Parameters

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.

- `reduce_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the reduction operation performed on its arguments.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

hpx/collectives/scatter.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace collectives

Functions

```
template<typename T>
hpx::future<T> scatter_from(char const *basename, this_site_arg this_site = this_site_arg(),
                           generation_arg generation = generation_arg(), root_site_arg
                           root_site = root_site_arg())
```

Scatter (receive) a set of values to different call sites

This function receives an element of a set of values operating on the given base name.

Return This function returns a future holding a the scattered value. It will become ready once the scatter operation has been completed.

Parameters

- `basename`: The base name identifying the scatter operation
- `generation`: The generational counter identifying the sequence number of the scatter operation performed on the given base name. This is optional and needs to be supplied only if the scatter operation on the given base name has to be performed more than once.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.
- `root_site`: The sequence number of the central scatter point (usually the locality id). This value is optional and defaults to 0.

```
template<typename T>
hpx::future<T> scatter_from(communicator comm, this_site_arg this_site = this_site_arg())
```

Scatter (receive) a set of values to different call sites

This function receives an element of a set of values operating on the given base name.

Return This function returns a future holding a the scattered value. It will become ready once the scatter operation has been completed.

Parameters

- `comm`: A communicator object returned from `create_communicator`
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

`template<typename T>`

```
hpx::future<T> scatter_to (char const *basename, std::vector<T> &&result, num_sites_arg  
    num_sites = num_sites_arg(), this_site_arg this_site =  
    this_site_arg(), generation_arg generation = generation_arg())
```

Scatter (send) a part of the value set at the given call site

This function transmits the value given by *result* to a central scatter site (where the corresponding *scatter_from* is executed)

Return This function returns a future holding a the scattered value. It will become ready once the scatter operation has been completed.

Parameters

- *basename*: The base name identifying the scatter operation
- *result*: The value to transmit to the central scatter point from this call site.
- *num_sites*: The number of participating sites (default: all localities).
- *generation*: The generational counter identifying the sequence number of the scatter operation performed on the given base name. This is optional and needs to be supplied only if the scatter operation on the given base name has to be performed more than once.
- *this_site*: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

```
template<typename T>  
hpx::future<T> scatter_to (communicator comm, std::vector<T> &&result, this_site_arg  
    this_site = this_site_arg())
```

Scatter (send) a part of the value set at the given call site

This function transmits the value given by *result* to a central scatter site (where the corresponding *scatter_from* is executed)

Return This function returns a future holding a the scattered value. It will become ready once the scatter operation has been completed.

Parameters

- *comm*: A communicator object returned from *create_communicator*
- *num_sites*: The number of participating sites (default: all localities).
- *this_site*: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

components

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/components/basename_registration.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

```
template<typename Client>
std::vector<Client> find_all_from_basename (std::string base_name, std::size_t num_ids)
```

Return all registered clients from all localities from the given base name.

This function locates all ids which were registered with the given base name. It returns a list of futures representing those ids.

Return all registered ids from all localities from the given base name.

Return A list of futures representing the ids which were registered using the given base name.

Note The futures embedded in the returned client objects will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Template Parameters

- Client: The client type to return

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- num_ids: [in] The number of registered ids to expect.

This function locates all ids which were registered with the given base name. It returns a list of futures representing those ids.

Return A list of futures representing the ids which were registered using the given base name.

Note The futures will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- num_ids: [in] The number of registered ids to expect.

```
template<typename Client>
std::vector<Client> find_from_basename (std::string base_name, std::vector<std::size_t> const &ids)
```

Return registered clients from the given base name and sequence numbers.

This function locates the ids which were registered with the given base name and the given sequence numbers. It returns a list of futures representing those ids.

Return registered ids from the given base name and sequence numbers.

Return A list of futures representing the ids which were registered using the given base name and sequence numbers.

Note The futures embedded in the returned client objects will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Template Parameters

- **Client**: The client type to return

Parameters

- **base_name**: [in] The base name for which to retrieve the registered ids.
- **ids**: [in] The sequence numbers of the registered ids.

This function locates the ids which were registered with the given base name and the given sequence numbers. It returns a list of futures representing those ids.

Return A list of futures representing the ids which were registered using the given base name and sequence numbers.

Note The futures will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Parameters

- **base_name**: [in] The base name for which to retrieve the registered ids.
- **ids**: [in] The sequence numbers of the registered ids.

template<typename **Client**>

Client **find_from_basename** (*std::string base_name, std::size_t sequence_nr*)

Return registered id from the given base name and sequence number.

This function locates the id which was registered with the given base name and the given sequence number. It returns a future representing those id.

This function locates the id which was registered with the given base name and the given sequence number. It returns a future representing those id.

Return A representing the id which was registered using the given base name and sequence numbers.

Note The future embedded in the returned client object will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Template Parameters

- **Client**: The client type to return

Parameters

- **base_name**: [in] The base name for which to retrieve the registered ids.
- **sequence_nr**: [in] The sequence number of the registered id.

Return A representing the id which was registered using the given base name and sequence numbers.

Note The future will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Parameters

- **base_name**: [in] The base name for which to retrieve the registered ids.
- **sequence_nr**: [in] The sequence number of the registered id.

```
template<typename Client, typename Stub>
hpx::future<bool> register_with_basename (std::string      base_name,
                                           components::client_base<Client, Stub> &client, std::size_t
                                           sequence_nr)
```

Register the id wrapped in the given client using the given base name.

The function registers the object the given client refers to using the provided base name.

Return A future representing the result of the registration operation itself.

Note The operation will fail if the given sequence number is not unique.

Template Parameters

- Client: The client type to register

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- client: [in] The client which should be registered using the given base name.
- sequence_nr: [in, optional] The sequential number to use for the registration of the id. This number has to be unique system wide for each registration using the same base name. The default is the current locality identifier. Also, the sequence numbers have to be consecutive starting from zero.

```
template<typename Client>
Client unregister_with_basename (std::string base_name, std::size_t sequence_nr)
```

Unregister the given id using the given base name.

Unregister the given base name.

The function unregisters the given ids using the provided base name.

The function unregisters the given ids using the provided base name.

Return A future representing the result of the un-registration operation itself.

Template Parameters

- Client: The client type to return

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- sequence_nr: [in, optional] The sequential number to use for the un-registration. This number has to be the same as has been used with *register_with_basename* before.

Return A future representing the result of the un-registration operation itself.

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- sequence_nr: [in, optional] The sequential number to use for the un-registration. This number has to be the same as has been used with *register_with_basename* before.

hpx/components/basename_registration_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`hpx::future<bool> register_with_basename (std::string base_name, hpx::id_type id, std::size_t sequence_nr = ~static_cast<std::size_t>(0))`

Register the given id using the given base name.

The function registers the given ids using the provided base name.

Return A future representing the result of the registration operation itself.

Note The operation will fail if the given sequence number is not unique.

Parameters

- `base_name`: [in] The base name for which to retrieve the registered ids.
- `id`: [in] The id to register using the given base name.
- `sequence_nr`: [in, optional] The sequential number to use for the registration of the id. This number has to be unique system wide for each registration using the same base name. The default is the current locality identifier. Also, the sequence numbers have to be consecutive starting from zero.

`hpx::future<bool> register_with_basename (std::string base_name, hpx::future<hpx::id_type> f, std::size_t sequence_nr = ~static_cast<std::size_t>(0))`

Register the id wrapped in the given future using the given base name.

The function registers the object the given future refers to using the provided base name.

Return A future representing the result of the registration operation itself.

Note The operation will fail if the given sequence number is not unique.

Parameters

- `base_name`: [in] The base name for which to retrieve the registered ids.
- `f`: [in] The future which should be registered using the given base name.
- `sequence_nr`: [in, optional] The sequential number to use for the registration of the id. This number has to be unique system wide for each registration using the same base name. The default is the current locality identifier. Also, the sequence numbers have to be consecutive starting from zero.

hpx/components/components_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/components/get_ptr.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
template<typename Component>
```

```
hpx::future<std::shared_ptr<Component>> get_ptr(hpx::id_type const &id)
```

Returns a future referring to the pointer to the underlying memory of a component.

The function `hpx::get_ptr` can be used to extract a future referring to the pointer to the underlying memory of a given component.

Return This function returns a future representing the pointer to the underlying memory for the component instance with the given `id`.

Note This function will successfully return the requested result only if the given component is currently located on the calling locality. Otherwise the function will raise an error.

Note The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned `shared_ptr` alive.

Parameters

- `id`: [in] The global id of the component for which the pointer to the underlying memory should be retrieved.

Template Parameters

- The only template parameter has to be the type of the server side component.

```
template<typename Derived, typename Stub>
```

```
hpx::future<std::shared_ptr<typename components::client_base<Derived, Stub>::server_component_type>> get_ptr(compo  
Stu  
cons  
&c)
```

Returns a future referring to the pointer to the underlying memory of a component.

The function `hpx::get_ptr` can be used to extract a future referring to the pointer to the underlying memory of a given component.

Return This function returns a future representing the pointer to the underlying memory for the component instance with the given `id`.

Note This function will successfully return the requested result only if the given component is currently located on the calling locality. Otherwise the function will raise an error.

Note The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned `shared_ptr` alive.

Parameters

- `c`: [in] A client side representation of the component for which the pointer to the underlying memory should be retrieved.

```
template<typename Component>
std::shared_ptr<Component> get_ptr(launch::sync_policy p, hpx::id_type const &id, error_code &ec = throws)
```

Returns the pointer to the underlying memory of a component.

The function `hpx::get_ptr_sync` can be used to extract the pointer to the underlying memory of a given component.

Return This function returns the pointer to the underlying memory for the component instance with the given `id`.

Note This function will successfully return the requested result only if the given component is currently located on the requesting locality. Otherwise the function will raise and error.

Note The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned `shared_ptr` alive.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `p`: [in] The parameter `p` represents a placeholder type to turn make the call synchronous.
- `id`: [in] The global id of the component for which the pointer to the underlying memory should be retrieved.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

Template Parameters

- `The`: only template parameter has to be the type of the server side component.

```
template<typename Derived, typename Stub>
std::shared_ptr<typename components::client_base<Derived, Stub>::server_component_type> get_ptr(launch::sync_policy p,
com-
po-
nents::client_base<Stub> const &c,
er-
ror_code &ec =
throws)
```

Returns the pointer to the underlying memory of a component.

The function `hpx::get_ptr_sync` can be used to extract the pointer to the underlying memory of a given component.

Return This function returns the pointer to the underlying memory for the component instance with the given *id*.

Note This function will successfully return the requested result only if the given component is currently located on the requesting locality. Otherwise the function will raise an error.

Note The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned shared_ptr alive.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *p*: [in] The parameter *p* represents a placeholder type to turn make the call synchronous.
- *c*: [in] A client side representation of the component for which the pointer to the underlying memory should be retrieved.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

components_base

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/components_base/agas_interface.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace agas

Functions

```
bool is_console()

bool register_name(launch::sync_policy, std::string const &name, naming::gid_type const
&gid, error_code &ec = throws)

bool register_name(launch::sync_policy, std::string const &name, hpx::id_type const &id,
error_code &ec = throws)

hpx::future<bool> register_name(std::string const &name, hpx::id_type const &id)

hpx::id_type unregister_name(launch::sync_policy, std::string const &name, error_code
&ec = throws)

hpx::future<hpx::id_type> unregister_name(std::string const &name)

hpx::id_type resolve_name(launch::sync_policy, std::string const &name, error_code &ec =
throws)

hpx::future<hpx::id_type> resolve_name(std::string const &name)

hpx::future<std::uint32_t> get_num_localities(naming::component_type type = naming::component_invalid)
```

```
std::uint32_t get_num_localities (launch::sync_policy, naming::component_type type, error_code &ec = throws)  
std::uint32_t get_num_localities (launch::sync_policy, error_code &ec = throws)  
std::string get_component_type_name (naming::component_type type, error_code &ec = throws)  
hpx::future<std::vector<std::uint32_t>> get_num_threads ()  
std::vector<std::uint32_t> get_num_threads (launch::sync_policy, error_code &ec = throws)  
hpx::future<std::uint32_t> get_num_overall_threads ()  
std::uint32_t get_num_overall_threads (launch::sync_policy, error_code &ec = throws)  
std::uint32_t get_locality_id (error_code &ec = throws)  
hpx::naming::gid_type get_locality ()  
std::vector<std::uint32_t> get_all_locality_ids (naming::component_type type, error_code &ec = throws)  
std::vector<std::uint32_t> get_all_locality_ids (error_code &ec = throws)  
bool is_local_address_cached (naming::gid_type const &gid, error_code &ec = throws)  
bool is_local_address_cached (naming::gid_type const &gid, naming::address &addr, error_code &ec = throws)  
bool is_local_address_cached (hpx::id_type const &id, error_code &ec = throws)  
bool is_local_address_cached (hpx::id_type const &id, naming::address &addr, error_code &ec = throws)  
void update_cache_entry (naming::gid_type const &gid, naming::address const &addr,  
    std::uint64_t count = 0, std::uint64_t offset = 0, error_code &ec = throws)  
bool is_local_lva_encoded_address (naming::gid_type const &gid)  
bool is_local_lva_encoded_address (hpx::id_type const &id)  
hpx::future<naming::address> resolve (hpx::id_type const &id)  
naming::address resolve (launch::sync_policy, hpx::id_type const &id, error_code &ec = throws)  
bool resolve_local (naming::gid_type const &gid, naming::address &addr, error_code &ec = throws)  
bool resolve_cached (naming::gid_type const &gid, naming::address &addr)  
hpx::future<bool> bind (naming::gid_type const &gid, naming::address const &addr,  
    std::uint32_t locality_id)  
bool bind (launch::sync_policy, naming::gid_type const &gid, naming::address const &addr,  
    std::uint32_t locality_id, error_code &ec = throws)  
hpx::future<bool> bind (naming::gid_type const &gid, naming::address const &addr, naming::gid_type const &locality_)
```

```

bool bind(launch::sync_policy, naming::gid_type const &gid, naming::address const &addr,
           naming::gid_type const &locality_, error_code &ec = throws)

hpx::future<naming::address> unbind(naming::gid_type const &gid, std::uint64_t count = 1)

naming::address unbind(launch::sync_policy, naming::gid_type const &gid, std::uint64_t count
                        = 1, error_code &ec = throws)

bool bind_gid_local(naming::gid_type const &gid, naming::address const &addr, error_code &ec = throws)

void unbind_gid_local(naming::gid_type const &gid, error_code &ec = throws)

bool bind_range_local(naming::gid_type const &gid, std::size_t count, naming::address
                      const &addr, std::size_t offset, error_code &ec = throws)

void unbind_range_local(naming::gid_type const &gid, std::size_t count, error_code &ec
                        = throws)

void garbage_collect_non_blocking(error_code &ec = throws)

void garbage_collect(error_code &ec = throws)

void garbage_collect_non_blocking(hpx::id_type const &id, error_code &ec = throws)
    Invoke an asynchronous garbage collection step on the given target locality.

void garbage_collect(hpx::id_type const &id, error_code &ec = throws)
    Invoke a synchronous garbage collection step on the given target locality.

hpx::id_type get_console_locality(error_code &ec = throws)
    Return an id_type referring to the console locality.

naming::gid_type get_next_id(std::size_t count, error_code &ec = throws)

void decref(naming::gid_type const &id, std::int64_t credits, error_code &ec = throws)

hpx::future<std::int64_t> incref(naming::gid_type const &gid, std::int64_t credits,
                                    hpx::id_type const &keep_alive = hpx::invalid_id)

std::int64_t incref(launch::sync_policy, naming::gid_type const &gid, std::int64_t credits = 1,
                      hpx::id_type const &keep_alive = hpx::invalid_id, error_code &ec = throws)

std::int64_t replenish_credits(naming::gid_type &gid)

hpx::future<hpx::id_type> get_colocation_id(hpx::id_type const &id)

hpx::id_type get_colocation_id(launch::sync_policy, hpx::id_type const &id, error_code
                                &ec = throws)

hpx::future<hpx::id_type> on_symbol_namespace_event(std::string const &name, bool
                                                       call_for_past_events)

hpx::future<std::pair<hpx::id_type, naming::address>> begin_migration(hpx::id_type const
                                                               &id)

bool end_migration(hpx::id_type const &id)

hpx::future<void> mark_as_migrated(naming::gid_type const &gid,
                                         hpx::move_only_function<std::pair<bool,
                                         hpx::future<void>>>
                                         > && bool expect_to_be_marked_as_migrating

```

```
std::pair<bool, components::pinned_ptr> was_object_migrated(naming::gid_type  
                           const &gid,  
                           hpx::move_only_function<components::pinned_ptr>  
> &&f  
  
void unmark_as_migrated(naming::gid_type const &gid)  
  
hpx::future<std::map<std::string, hpx::id_type>> find_symbols(std::string const &pattern =  
                           "*" )  
  
std::map<std::string, hpx::id_type> find_symbols(hpx::launch::sync_policy, std::string const  
                           &pattern = "*")  
  
naming::component_type register_factory(std::uint32_t prefix, std::string const &name,  
                                         error_code &ec = throws)  
  
naming::component_type get_component_id(std::string const &name, error_code &ec =  
                                         throws)  
  
void destroy_component(naming::gid_type const &gid, naming::address const &addr)  
  
naming::address_type get_primary_ns_lva()  
  
naming::address_type get_symbol_ns_lva()  
  
naming::address_type get_runtime_support_lva()
```

hpx/components_base/component_commandline.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_DEFINE_COMPONENT_COMMANDLINE_OPTIONS(add_options_function)
```

```
HPX_REGISTER_COMMANDLINE_MODULE(add_options_function)
```

```
HPX_REGISTER_COMMANDLINE_MODULE_DYNAMIC(add_options_function)
```

```
namespace hpx
```

```
namespace components
```

```
struct component_commandline : public component_commandline_base
```

```
#include <component_commandline.hpp> The component_startup_shutdown provides a minimal im-  
plementation of a component's startup/shutdown function provider.
```

Public Functions

`hpx::program_options::options_description add_commandline_options()`
 Return any additional command line options valid for this component.

Return The module is expected to fill a options_description object with any additional command line options this component will handle.

Note This function will be executed by the runtime system during system startup.

```
namespace cmdline_options_provider
```

Functions

`hpx::program_options::options_description add_commandline_options()`

hpx/components_base/component_startup_shutdown.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_DEFINE_COMPONENT_STARTUP_SHUTDOWN (startup_, shutdown_)

HPX_REGISTER_STARTUP_SHUTDOWN_MODULE_ (startup, shutdown)
HPX_REGISTER_STARTUP_SHUTDOWN_MODULE (startup, shutdown)
HPX_REGISTER_STARTUP_SHUTDOWN_MODULE_DYNAMIC (startup, shutdown)
HPX_REGISTER_STARTUP_MODULE (startup)
HPX_REGISTER_STARTUP_MODULE_DYNAMIC (startup)
HPX_REGISTER_SHUTDOWN_MODULE (shutdown)
HPX_REGISTER_SHUTDOWN_MODULE_DYNAMIC (shutdown)

namespace hpx
```

```
namespace components
```

```
template<bool(*)(startup_function_type &, bool &) Startup, bool(*)(shutdown_function_type &, bool &) Shutdown>
#include <component_startup_shutdown.hpp> The component_startup_shutdown class provides a minimal implementation of a component's startup/shutdown function provider.
```

Public Functions

```
bool get_startup_function (startup_function_type &startup, bool &pre_startup)  
    Return any startup function for this component.
```

Return Returns *true* if the parameter *startup* has been successfully initialized with the startup function.

Parameters

- *startup*: [in, out] The module is expected to fill this function object with a reference to a startup function. This function will be executed by the runtime system during system startup.

```
bool get_shutdown_function (shutdown_function_type &shutdown,  
                           &pre_shutdown)  
    Return any startup function for this component.
```

Return Returns *true* if the parameter *shutdown* has been successfully initialized with the shutdown function.

Parameters

- *shutdown*: [in, out] The module is expected to fill this function object with a reference to a startup function. This function will be executed by the runtime system during system startup.

hpx/components_base/component_type.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_DEFINE_GET_COMPONENT_TYPE (component)  
HPX_DEFINE_GET_COMPONENT_TYPE_TEMPLATE (template_, component)  
HPX_DEFINE_GET_COMPONENT_TYPE_STATIC (component, type)  
HPX_DEFINE_COMPONENT_NAME (...)  
HPX_DEFINE_COMPONENT_NAME_ (...)  
HPX_DEFINE_COMPONENT_NAME_2 (Component, name)  
HPX_DEFINE_COMPONENT_NAME_3 (Component, name, base_name)  
namespace hpx  
  
namespace components
```

Typedefs

```
using component_deleter_type = void (*) (hpx::naming::gid_type const&,  

                                         hpx::naming::address const&)
```

Enums

```
enum component_enum_type  

  Values:  

    component_invalid = naming::address::component_invalid  

    component_runtime_support = 0  

    component_plain_function = 1  

    component_base_lco = 2  

    component_base_lco_with_value_unmanaged = 3  

    component_base_lco_with_value = 4  

    component_latch = ((5 << 10) | component_base_lco_with_value)  

    component_barrier = ((6 << 10) | component_base_lco)  

    component.promise = ((7 << 10) | component_base_lco_with_value)  

    component_agas_locality_namespace = 8  

    component_agas_primary_namespace = 9  

    component_agas_component_namespace = 10  

    component_agas_symbol_namespace = 11  

    component_last  

    component_first_dynamic = component_last  

    component_upper_bound = 0xfffffL  

enum factory_state_enum  

  Values:  

    factory_enabled = 0  

    factory_disabled = 1  

    factory_check = 2
```

Functions

```
bool &enabled(component_type type)  

util::atomic_count &instance_count(component_type type)  

component_deleter_type &deleter(component_type type)  

bool enumerate_instance_counts(hpx::move_only_function<bool> component_type  

  > const &f)
```

```
const std::string get_component_type_name (component_type type)
    Return the string representation for a given component type id.
```

```
constexpr component_type get_base_type (component_type t)
    The lower short word of the component type is the type of the component exposing the actions.
```

```
constexpr component_type get_derived_type (component_type t)
    The upper short word of the component is the actual component type.
```

```
constexpr component_type derived_component_type (component_type derived, component_type base)
A component derived from a base component exposing the actions needs to have a specially formatted component type.
```

```
constexpr bool types_are_compatible (component_type lhs, component_type rhs)
    Verify the two given component types are matching (compatible)
```

```
template<typename Component, typename Enable = void>
constexpr char const *get_component_name ()
```

```
template<typename Component, typename Enable = void>
constexpr const char *get_component_base_name ()
```

```
template<typename Component>
component_type get_component_type ()
```

```
template<typename Component>
void set_component_type (component_type type)
```

```
namespace naming
```

Functions

```
std::ostream &operator<< (std::ostream&, address const&)
```

hpx/components_base/components_base_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

TypeDefs

```
typedef abstract_component_base<Component> instead
```

```
template<typename Component, typename Derived>
class managed_component
```

```
#include <managed_component_base.hpp> The managed_component template is used as a indirection layer for components allowing to gracefully handle the access to non-existing components.
```

```
Additionally it provides memory management capabilities for the wrapping instances, and it integrates the memory management with the AGAS service. Every instance of a managed_component gets
```

assigned a global id. The provided memory management allocates the *managed_component* instances from a special heap, ensuring fast allocation and avoids a full network round trip to the AGAS service for each of the allocated instances.

Template Parameters

- Component:
- Derived:

hpx/components_base/get_lva.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
template<typename Component, typename Enable = void>
struct get_lva
    #include <get_lva.hpp> The get_lva template is a helper structure allowing to convert a local virtual address as stored in a local address (returned from the function resolver_client::resolve) to the address of the component implementing the action.
```

The default implementation uses the template argument *Component* to deduce the type wrapping the component implementing the action. This is used to get the needed address.

Template Parameters

- Component: This is the type of the component implementing the action to execute.

Public Static Functions

```
static constexpr Component *call(naming::address_type lva)
```

hpx/components_base/server/fixed_component_base.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/components_base/server/managed_component_base.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
template<>
struct init<traits::construct_with_back_ptr>
```

Public Static Functions

```
template<typename Component, typename Managed>
static constexpr void call (Component*, Managed*)

template<typename Component, typename Managed, typename ...Ts>
static void call_new (Component *&component, Managed *this_, Ts&&... vs)

template<>
struct init<traits::construct_without_back_ptr>
```

Public Static Functions

```
template<typename Component, typename Managed>
static void call (Component *component, Managed *this_)

template<typename Component, typename Managed, typename ...Ts>
static void call_new (Component *&component, Managed *this_, Ts&&... vs)

template<>
struct destroy_backptr<traits::managed_object_is_lifetime_controlled>
```

Public Static Functions

```
template<typename BackPtr>
static void call (BackPtr *back_ptr)

template<>
struct destroy_backptr<traits::managed_object_controls_lifetime>
```

Public Static Functions

```
template<typename BackPtr>
static constexpr void call (BackPtr*)

template<>
struct manage_lifetime<traits::managed_object_is_lifetime_controlled>
```

Public Static Functions

```
template<typename Component>
static constexpr void call (Component*)

template<typename Component>
static void addref (Component *component)

template<typename Component>
static void release (Component *component)

template<>
struct manage_lifetime<traits::managed_object_controls_lifetime>
```

Public Static Functions

```
template<typename Component>
static void call (Component *component)

template<typename Component>
static constexpr void addref (Component*)

template<typename Component>
static constexpr void release (Component*)

namespace hpx
```

```
namespace components
```

Functions

```
template<typename Component, typename Derived>
void intrusive_ptr_add_ref (managed_component<Component, Derived> *p)

template<typename Component, typename Derived>
void intrusive_ptr_release (managed_component<Component, Derived> *p)

namespace detail_adl_barrier

template<>
struct destroy_backptr<traits::managed_object_controls_lifetime>
```

Public Static Functions

```
template<typename BackPtr>
static constexpr void call (BackPtr*)

template<>
struct destroy_backptr<traits::managed_object_is_lifetime_controlled>
```

Public Static Functions

```
template<typename BackPtr>
static void call (BackPtr *back_ptr)

template<>
struct init<traits::construct_with_back_ptr>
```

Public Static Functions

```
template<typename Component, typename Managed>
static constexpr void call (Component*, Managed*)

template<typename Component, typename Managed, typename ...Ts>
static void call_new (Component *&component, Managed *this_, Ts&&... vs)

template<>
struct init<traits::construct_without_back_ptr>
```

Public Static Functions

```
template<typename Component, typename Managed>
static void call (Component *component, Managed *this_)

template<typename Component, typename Managed, typename ...Ts>
static void call_new (Component *&component, Managed *this_, Ts&&... vs)

template<>
struct manage_lifetime<traits::managed_object_controls_lifetime>
```

Public Static Functions

```
template<typename Component>
static void call (Component *component)

template<typename Component>
static constexpr void addref (Component*)

template<typename Component>
static constexpr void release (Component*)

template<>
struct manage_lifetime<traits::managed_object_is_lifetime_controlled>
```

Public Static Functions

```
template<typename Component>
static constexpr void call (Component*)

template<typename Component>
static void addref (Component *component)

template<typename Component>
static void release (Component *component)
```

compute

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/compute/vector.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace compute
```

Functions

```
template<typename T, typename Allocator>
void swap(vector<T, Allocator> &x, vector<T, Allocator> &y)
    Effects: x.swap(y);

template<typename T, typename Allocator = std::allocator<T>>
class vector
```

Public Types

```
template<>
using value_type = T
    Member types (FIXME: add reference to std.

template<>
using allocator_type = Allocator

template<>
using access_target = typename alloc_traits::access_target

template<>
using size_type = std::size_t

template<>
using difference_type = std::ptrdiff_t

template<>
using reference = typename alloc_traits::reference

template<>
using const_reference = typename alloc_traits::const_reference

template<>
using pointer = typename alloc_traits::pointer

template<>
using const_pointer = typename alloc_traits::const_pointer

template<>
using iterator = detail::iterator<T, Allocator>

template<>
using const_iterator = detail::iterator<T const, Allocator>
```

```
template<>
using reverse_iterator = detail::reverse_iterator<T, Allocator>

template<>
using const_reverse_iterator = detail::const_reverse_iterator<T, Allocator>
```

Public Functions

```
vector (Allocator const &alloc = Allocator())

vector (size_type count, T const &value, Allocator const &alloc = Allocator())

vector (size_type count, Allocator const &alloc = Allocator())

template<typename InIter, typename Enable = typename std::enable_if<hpx::traits::is_input_iterator<InIter>::value
vector (InIter first, InIter last, Allocator const &alloc)
```

vector (*vector const &other*)

vector (*vector const &other*, Allocator **const &alloc**)

vector (*vector &&other*)

vector (*vector &&other*, Allocator **const &alloc**)

vector (*std::initializer_list<T> init*, Allocator **const &alloc**)

~vector ()

vector &operator= (*vector const &other*)

vector &operator= (*vector &&other*)

allocator_type **get_allocator** () **const**
Returns the allocator associated with the container.

reference **operator[]** (size_type *pos*)

const_reference **operator[]** (size_type *pos*) **const**

pointer **data** ()
Returns pointer to the underlying array serving as element storage. The pointer is such that range [*data()*; *data()* + *size()*] is always a valid range, even if the container is empty (*data()* is not dereferenceable in that case).

const_pointer **data** () **const**
Returns pointer to the underlying array serving as element storage. The pointer is such that range [*data()*; *data()* + *size()*] is always a valid range, even if the container is empty (*data()* is not dereferenceable in that case).

T ***device_data** () **const**
Returns a raw pointer corresponding to the address of the data allocated on the device.

std::size_t size () **const**

std::size_t capacity () **const**

bool **empty** () **const**
Returns: *size()* == 0.

```
void resize(size_type)
```

Effects: If size <= size(), equivalent to calling pop_back() size() - size times. If size() < size, appends size - size() default-inserted elements to the sequence.

Requires: T shall be MoveInsertable and DefaultInsertable into *this.

Remarks: If an exception is thrown other than by the move constructor of a non-CopyInsertable T there are no effects.

```
void resize(size_type, T const&)
```

Effects: If size <= size(), equivalent to calling pop_back() size() - size times. If size() < size, appends size - size() copies of val to the sequence.

Requires: T shall be CopyInsertable into *this.

Remarks: If an exception is thrown there are no effects.

```
iterator begin()
```

```
iterator end()
```

```
const_iterator cbegin() const
```

```
const_iterator cend() const
```

```
const_iterator begin() const
```

```
const_iterator end() const
```

```
void swap(vector &other)
```

Effects: Exchanges the contents and capacity() of *this with that of x.

Complexity: Constant time.

```
void clear()
```

Effects: Erases all elements in the range [begin(),end()). Destroys all elements in a. Invalidates all references, pointers, and iterators referring to the elements of a and may invalidate the past-the-end iterator.

Post: a.empty() returns true.

Complexity: Linear.

Private Types

```
typedef traits::allocator_traits<Allocator> alloc_traits
```

Private Members

```
size_type size_
```

```
size_type capacity_
```

```
allocator_type alloc_
```

```
pointer data_
```

hpx/compute/host/block_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
template<typename Executor>
struct executor_execution_category<compute::host::block_executor<Executor>>
```

Public Types

```
typedef hpx::execution::parallel_execution_tag type
namespace hpx
```

```
namespace compute
```

```
namespace host
```

```
template<typename Executor = hpx::parallel::execution::restricted_thread_pool_executor>
struct block_executor
#include <block_executor.hpp> The block executor can be used to build NUMA aware programs.  
It will distribute work evenly across the passed targets
```

Template Parameters

- **Executor**: The underlying executor to use

Public Types

```
template<>
using executor_parameters_type = hpx::execution::static_chunk_size
```

Public Functions

```
block_executor(std::vector<host::target> const &targets, threads::thread_priority  
priority = threads::thread_priority::high, threads::thread_stacksize  
stacksize = threads::thread_stacksize::default_,  
threads::thread_schedule_hint schedulehint = {})

block_executor(std::vector<host::target> &&targets)

block_executor(block_executor const &other)

block_executor(block_executor &&other)

block_executor &operator=(block_executor const &other)

block_executor &operator=(block_executor &&other)

template<typename F, typename ...Ts>
void post(F &&f, Ts&&... ts)

template<typename F, typename ...Ts>
```

```

hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> async_execute(F
&&f,
Ts&&...
ts)

template<typename F, typename ...Ts>
hpx::util::detail::invoke_deferred_result<F, Ts...>::type sync_execute(F &&f, Ts&&...
ts)

template<typename F, typename Shape, typename ...Ts>
std::vector<hpx::future<typename parallel::execution::detail::bulk_function_result<F, Shape, Ts...>::type>> bulk_sync_execute(F
&&f,
Shape
const
&shape,
Ts&&...
ts)

std::vector<host::target> const &targets() const

```

Private Functions

```
void init_executors()
```

Private Members

```

std::vector<host::target> targets_
std::atomic<std::size_t> current_
std::vector<Executor> executors_
threads::thread_priority priority_ = threads::thread_priority::high
threads::thread_stacksize stacksize_ = threads::thread_stacksize::default_
threads::thread_schedule_hint schedulehint_ = {}

namespace parallel

namespace execution

template<typename Executor>
struct executor_execution_category<compute::host::block_executor<Executor>>

```

Public Types

```
typedef hpx::execution::parallel_execution_tag type
```

hpx/compute/host/target_distribution_policy.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

distribution_policies

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/distribution_policies/binpacking_distribution_policy.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

Variables

```
constexpr char const *const default_binpacking_counter_name = "/runtime{locality/total}/count/components/binpacking"
const binpacking_distribution_policy binpacked = {}
```

A predefined instance of the binpacking *distribution_policy*. It will represent the local locality and will place all items to create here.

```
struct binpacking_distribution_policy
```

```
#include <binpacking_distribution_policy.hpp> This class specifies the parameters for a binpacking distribution policy to use for creating a given number of items on a given set of localities. The binpacking policy will distribute the new objects in a way such that each of the localities will equalize the number of overall objects of this type based on a given criteria (by default this criteria is the overall number of objects of this type).
```

Public Functions

```
binpacking_distribution_policy()
```

Default-construct a new instance of a *binpacking_distribution_policy*. This policy will represent one locality (the local locality).

```
binpacking_distribution_policy operator()(std::vector<id_type> const &locs,
                                         char const *perf_counter_name = default_binpacking_counter_name) const
```

Create a new *default_distribution* policy representing the given set of localities.

Parameters

- `locs`: [in] The list of localities the new instance should represent

- `perf_counter_name`: [in] The name of the performance counter which should be used as the distribution criteria (by default the overall number of existing instances of the given component type will be used).

```
binpacking_distribution_policy operator() (std::vector<id_type>      &&locs,      char
                                         const      *perf_counter_name = de-
                                         fault_binpacking_counter_name) const
```

Create a new *default_distribution* policy representing the given set of localities.

Parameters

- `locs`: [in] The list of localities the new instance should represent
- `perf_counter_name`: [in] The name of the performance counter which should be used as the distribution criteria (by default the overall number of existing instances of the given component type will be used).

```
binpacking_distribution_policy operator() (id_type      const      &loc,      char
                                         const      *perf_counter_name = de-
                                         fault_binpacking_counter_name) const
```

Create a new *default_distribution* policy representing the given locality

Parameters

- `loc`: [in] The locality the new instance should represent
- `perf_counter_name`: [in] The name of the performance counter that should be used as the distribution criteria (by default the overall number of existing instances of the given component type will be used).

```
template<typename Component, typename ...Ts>
hpx::future<hpx::id_type> create(Ts&&... vs) const
```

Create one object on one of the localities associated by this policy instance

Return A future holding the global address which represents the newly created object

Parameters

- `vs`: [in] The arguments which will be forwarded to the constructor of the new object.

```
template<typename Component, typename ...Ts>
hpx::future<std::vector<bulk_locality_result>> bulk_create (std::size_t count, Ts&&... vs)
                                                               const
```

Create multiple objects on the localities associated by this policy instance

Return A future holding the list of global addresses which represent the newly created objects

Parameters

- `count`: [in] The number of objects to create
- `vs`: [in] The arguments which will be forwarded to the constructors of the new objects.

```
std::string const &get_counter_name () const
```

Returns the name of the performance counter associated with this policy instance.

```
std::size_t get_num_localities () const
```

Returns the number of associated localities for this distribution policy

Note This function is part of the creation policy implemented by this class

hpx/distribution_policies/colocating_distribution_policy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace components

Variables

const colocating_distribution_policy colocated = {}

A predefined instance of the co-locating *distribution_policy*. It will represent the local locality and will place all items to create here.

struct colocating_distribution_policy

`#include <colocating_distribution_policy.hpp>` This class specifies the parameters for a distribution policy to use for creating a given number of items on the locality where a given object is currently placed.

Public Functions

colocating_distribution_policy()

Default-construct a new instance of a *colocating_distribution_policy*. This policy will represent the local locality.

colocating_distribution_policy operator() (id_type const &id) const

Create a new *colocating_distribution_policy* representing the locality where the given object is current located

Parameters

- `id`: [in] The global address of the object with which the new instances should be colocated on

`template<typename Client, typename Stub>`

`colocating_distribution_policy operator() (client_base<Client, Stub> const &client)`

const

Create a new *colocating_distribution_policy* representing the locality where the given object is current located

Parameters

- `client`: [in] The client side representation of the object with which the new instances should be colocated on

`template<typename Component, typename ...Ts>`

`hpx::future<hpx::id_type> create (Ts&&... vs) const`

Create one object on the locality of the object this distribution policy instance is associated with

Note This function is part of the placement policy implemented by this class

Return A future holding the global address which represents the newly created object

Parameters

- `vs`: [in] The arguments which will be forwarded to the constructor of the new object.

```
template<typename Component, typename ...Ts>
hpx::future<std::vector<bulk_locality_result>> bulk_create(std::size_t count, Ts&&... vs)
    const
Create multiple objects colocated with the object represented by this policy instance
```

Note This function is part of the placement policy implemented by this class

Return A future holding the list of global addresses which represent the newly created objects
Parameters

- count: [in] The number of objects to create
- vs: [in] The arguments which will be forwarded to the constructors of the new objects.

```
template<typename Action, typename ...Ts>
async_result<Action>::type async(launch policy, Ts&&... vs) const
```

```
template<typename Action, typename Callback, typename ...Ts>
async_result<Action>::type async_cb(launch policy, Callback &&cb, Ts&&... vs) const
Note This function is part of the invocation policy implemented by this class
```

```
template<typename Action, typename Continuation, typename ...Ts>
bool apply(Continuation &&c, threads::thread_priority priority, Ts&&... vs) const
```

Note This function is part of the invocation policy implemented by this class

```
template<typename Action, typename ...Ts>
bool apply(threads::thread_priority priority, Ts&&... vs) const
```

```
template<typename Action, typename Continuation, typename Callback, typename ...Ts>
bool apply_cb(Continuation &&c, threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
Note This function is part of the invocation policy implemented by this class
```

```
template<typename Action, typename Callback, typename ...Ts>
bool apply_cb(threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
```

```
std::size_t get_num_localities() const
```

Returns the number of associated localities for this distribution policy

Note This function is part of the creation policy implemented by this class

```
hpx::id_type get_next_target() const
```

Returns the locality which is anticipated to be used for the next async operation

```
template<typename Action>
struct async_result
#include <colocating_distribution_policy.hpp>
```

Note This function is part of the invocation policy implemented by this class

Public Types

```
template<>
using type = hpx::future<typename traits::promise_local_result<typename hpx::traits::extract_action<Act>
```

hpx/distribution_policies/target_distribution_policy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

Variables

```
const target_distribution_policy target = {}
```

A predefined instance of the `target_distribution_policy`. It will represent the local locality and will place all items to create here.

```
struct target_distribution_policy
```

#include <target_distribution_policy.hpp> This class specifies the parameters for a simple distribution policy to use for creating (and evenly distributing) a given number of items on a given set of localities.

Public Functions

```
target_distribution_policy()
```

Default-construct a new instance of a `target_distribution_policy`. This policy will represent one locality (the local locality).

```
target_distribution_policy operator()(id_type const &id) const
```

Create a new `target_distribution_policy` representing the given locality

Parameters

- `loc`: [in] The locality the new instance should represent

```
template<typename Component, typename ...Ts>
```

```
hpx::future<hpx::id_type> create(Ts&&... vs) const
```

Create one object on one of the localities associated by this policy instance

Note This function is part of the placement policy implemented by this class

Return A future holding the global address which represents the newly created object

Parameters

- `vs`: [in] The arguments which will be forwarded to the constructor of the new object.

```
template<typename Component, typename ...Ts>
```

```
hpx::future<std::vector<bulk_locality_result>> bulk_create(std::size_t count, Ts&&... vs)
```

```
const
```

Create multiple objects on the localities associated by this policy instance

Note This function is part of the placement policy implemented by this class

Return A future holding the list of global addresses which represent the newly created objects
Parameters

- `count`: [in] The number of objects to create
- `vs`: [in] The arguments which will be forwarded to the constructors of the new objects.

```
template<typename Action, typename ...Ts>
async_result<Action>::type async (launch policy, Ts&&... vs) const

template<typename Action, typename Callback, typename ...Ts>
async_result<Action>::type async_cb (launch policy, Callback &&cb, Ts&&... vs) const
```

Note This function is part of the invocation policy implemented by this class

```
template<typename Action, typename Continuation, typename ...Ts>
bool apply (Continuation &&c, threads::thread_priority priority, Ts&&... vs) const
```

Note This function is part of the invocation policy implemented by this class

```
template<typename Action, typename ...Ts>
bool apply (threads::thread_priority priority, Ts&&... vs) const
```

```
template<typename Action, typename Continuation, typename Callback, typename ...Ts>
bool apply_cb (Continuation &&c, threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
```

Note This function is part of the invocation policy implemented by this class

```
template<typename Action, typename Callback, typename ...Ts>
bool apply_cb (threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
```

```
std::size_t get_num_localities () const
```

Returns the number of associated localities for this distribution policy

Note This function is part of the creation policy implemented by this class

```
hpx::id_type get_next_target () const
```

Returns the locality which is anticipated to be used for the next async operation

```
template<typename Action>
struct async_result
#include <target_distribution_policy.hpp>
```

Note This function is part of the invocation policy implemented by this class

Public Types

```
template<>
using type = hpx::future<typename traits::promise_local_result<typename hpx::traits::extract_action<Act
```

hpx/distribution_policies/unwrapping_result_policy.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

```
struct unwrapping_result_policy
```

#include <unwrapping_result_policy.hpp> This class is a distribution policy that can be using with actions that return futures. For those actions it is possible to apply certain optimizations if the action is invoked synchronously.

Public Functions

```
unwrapping_result_policy(id_type const &id)
```

```
template<typename Client, typename Stub>
unwrapping_result_policy(client_base<Client, Stub> const &client)
```

```
template<typename Action, typename ...Ts>
async_result<Action>::type async(launch policy, Ts&&... vs) const
```

```
template<typename Action, typename ...Ts>
async_result<Action>::type async(launch::sync_policy, Ts&&... vs) const
```

```
template<typename Action, typename Callback, typename ...Ts>
async_result<Action>::type async_cb(launch policy, Callback &&cb, Ts&&... vs) const
```

```
template<typename Action, typename Continuation, typename ...Ts>
bool apply(Continuation &&c, threads::thread_priority priority, Ts&&... vs) const
```

Note This function is part of the invocation policy implemented by this class

```
template<typename Action, typename ...Ts>
bool apply(threads::thread_priority priority, Ts&&... vs) const
```

```
template<typename Action, typename Continuation, typename Callback, typename ...Ts>
bool apply_cb(Continuation &&c, threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
```

Note This function is part of the invocation policy implemented by this class

```
template<typename Action, typename Callback, typename ...Ts>
bool apply_cb(threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
```

```
hpx::id_type const &get_next_target() const
```

```
template<typename Action>
struct async_result
```

Public Types

```
template<>
using type = typename traits::promise_local_result::type
```

executors_distributed

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/executors_distributed/distribution_policy_executor.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace parallel
```

```
namespace execution
```

Functions

```
template<typename DistPolicy>
distribution_policy_executor<typename std::decay<DistPolicy>::type> make_distribution_policy_executor
```

Create a new `distribution_policy_executor` from the given distribution_policy.

Parameters

- `policy`: The distribution_policy to create an executor from

```
template<typename DistPolicy>
class distribution_policy_executor
```

`#include <distribution_policy_executor.hpp>` A `distribution_policy_executor` creates groups of parallel execution agents which execute in threads implicitly created by the executor and placed on any of the associated localities.

Template Parameters

- `DistPolicy`: The distribution policy type for which an executor should be created. The expression `hpx::traits::is_distribution_policy<DistPolicy>::value` must evaluate to true.

Public Functions

```
template<typename DistPolicy_, typename Enable = typename std::enable_if<!std::is_same<distribution_>>::value>
distribution_policy_executor(DistPolicy_ &&policy)
```

Create a new distribution_policy executor from the given distribution policy

Parameters

- `policy`: The distribution_policy to create an executor from

Private Members

DistPolicy **policy_**

init_runtime

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/hpx_finalize.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

int finalize(double shutdown_timeout, double localwait = -1.0, error_code &ec = throws)

Main function to gracefully terminate the HPX runtime system.

The function *hpx::finalize* is the main way to (gracefully) exit any HPX application. It should be called from one locality only (usually the console) and it will notify all connected localities to finish execution. Only after all other localities have exited this function will return, allowing to exit the console locality as well.

During the execution of this function the runtime system will invoke all registered shutdown functions (see *hpx::init*) on all localities.

The default value (-1.0) will try to find a globally set timeout value (can be set as the configuration parameter *hpx.shutdown_timeout*), and if that is not set or -1.0 as well, it will disable any timeout, each connected locality will wait for all existing HPX-threads to terminate.

Parameters

- *shutdown_timeout*: This parameter allows to specify a timeout (in microseconds), specifying how long any of the connected localities should wait for pending tasks to be executed. After this timeout, all suspended HPX-threads will be aborted. Note, that this function will not abort any running HPX-threads. In any case the shutdown will not proceed as long as there is at least one pending/running HPX-thread.

The default value (-1.0) will try to find a globally set wait time value (can be set as the configuration parameter “*hpx.finalize_wait_time*”), and if this is not set or -1.0 as well, it will disable any addition local wait time before proceeding.

Parameters

- *localwait*: This parameter allows to specify a local wait time (in microseconds) before the connected localities will be notified and the overall shutdown process starts.

This function will block and wait for all connected localities to exit before returning to the caller. It should be the last HPX-function called by any application.

Return This function will always return zero.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Using this function is an alternative to *hpx::disconnect*, these functions do not need to be called both.

```
int finalize (error_code &ec = throws)
```

Main function to gracefully terminate the HPX runtime system.

The function *hpx::finalize* is the main way to (gracefully) exit any HPX application. It should be called from one locality only (usually the console) and it will notify all connected localities to finish execution. Only after all other localities have exited this function will return, allowing to exit the console locality as well.

During the execution of this function the runtime system will invoke all registered shutdown functions (see *hpx::init*) on all localities.

This function will block and wait for all connected localities to exit before returning to the caller. It should be the last HPX-function called by any application.

Return This function will always return zero.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

Using this function is an alternative to *hpx::disconnect*, these functions do not need to be called both.

```
void terminate ()
```

Terminate any application non-gracefully.

The function *hpx::terminate* is the non-graceful way to exit any application immediately. It can be called from any locality and will terminate all localities currently used by the application.

Note This function will cause HPX to call *std::terminate()* on all localities associated with this application. If the function is called not from an HPX thread it will fail and return an error using the argument *ec*.

```
int disconnect (double shutdown_timeout, double localwait = -1.0, error_code &ec = throws)
```

Disconnect this locality from the application.

The function *hpx::disconnect* can be used to disconnect a locality from a running HPX application.

During the execution of this function the runtime system will invoke all registered shutdown functions (see *hpx::init*) on this locality. The default value (-1.0) will try to find a globally set timeout value (can be set as the configuration parameter "hpx.shutdown_timeout"), and if that is not set or -1.0 as well, it will disable any timeout, each connected locality will wait for all existing HPX-threads to terminate.

Parameters

- *shutdown_timeout*: This parameter allows to specify a timeout (in microseconds), specifying how long this locality should wait for pending tasks to be executed. After this timeout, all

suspended HPX-threads will be aborted. Note, that this function will not abort any running HPX-threads. In any case the shutdown will not proceed as long as there is at least one pending/running HPX-thread.

The default value (`-1.0`) will try to find a globally set wait time value (can be set as the configuration parameter `hpx.finalize_wait_time`), and if this is not set or `-1.0` as well, it will disable any addition local wait time before proceeding.

Parameters

- `localwait`: This parameter allows to specify a local wait time (in microseconds) before the connected localities will be notified and the overall shutdown process starts.

This function will block and wait for this locality to finish executing before returning to the caller. It should be the last HPX-function called by any locality being disconnected.

Return This function will always return zero.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`int disconnect(error_code &ec = throws)`

Disconnect this locality from the application.

The function `hpx::disconnect` can be used to disconnect a locality from a running HPX application.

During the execution of this function the runtime system will invoke all registered shutdown functions (see `hpx::init`) on this locality.

This function will block and wait for this locality to finish executing before returning to the caller. It should be the last HPX-function called by any locality being disconnected.

Return This function will always return zero.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`int stop(error_code &ec = throws)`

Stop the runtime system.

This function will block and wait for this locality to finish executing before returning to the caller. It should be the last HPX-function called on every locality. This function should be used only if the runtime system was started using `hpx::start`.

Return The function returns the value, which has been returned from the user supplied main HPX function (usually `hpx_main`).

hpx/hpx_init.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/hpx_init_impl.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

```
int init (std::function<int> hpx::program_options::variables_map&
    >f, int argc, char **argv, init_params const &params = init_params())Main entry point for launching
the HPX runtime system.
```

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. This overload will not call `hpx_main`.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread.

Return The function returns the value, which has been returned from the user supplied `f`.

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the `parametermode`.

Parameters

- `f`: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.
- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `params`: [in] The parameters to the `hpx::init` function (See documentation of `hpx::init_params`)

```
int init (std::function<int> int, char **
    >f, int argc, char **argv, init_params const &params = init_params())Main entry point for launching
the HPX runtime system.
```

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. This overload will not call `hpx_main`.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread.

Return The function returns the value, which has been returned from the user supplied `f`.

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the `parametermode`.

Parameters

- `f`: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.
- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `params`: [in] The parameters to the `hpx::init` function (See documentation of `hpx::init_params`)

```
int init (int argc, char **argv, init_params const &params = init_params())
```

Main entry point for launching the HPX runtime system.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. This overload will not call `hpx_main`.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread.

Return The function returns the value, which has been returned from the user supplied `f`.

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the `parametermode`.

Parameters

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `params`: [in] The parameters to the `hpx::init` function (See documentation of `hpx::init_params`)

```
int init (std::nullptr_t f, int argc, char **argv, init_params const &params = init_params())
```

Main entry point for launching the HPX runtime system.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. This overload will not call `hpx_main`.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread.

Return The function returns the value, which has been returned from the user supplied `f`.

Note If the parameter mode is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parametermode.

Parameters

- f: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.
- argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- params: [in] The parameters to the `hpx::init` function (See documentation of `hpx::init_params`)

```
int init(init_params const &params = init_params())
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

Return The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. If no command line arguments are passed, console mode is assumed.

Note If no command line arguments are passed the HPX runtime system will not support any of the default command line options as described in the section ‘HPX Command Line Options’.

Parameters

- params: [in] The parameters to the `hpx::init` function (See documentation of `hpx::init_params`)

hpx/hpx_init_params.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
struct init_params
```

```
#include <hpx_init_params.hpp> Parameters used to initialize the HPX runtime through hpx::init and hpx::start.
```

Public Members

```
std::reference_wrapper<hpx::program_options::options_description const> desc_cmdline = detail::default_desc
std::vector<std::string> cfg
startup_function_type startup
shutdown_function_type shutdown
hpx::runtime_mode mode = ::hpx::runtime_mode::default_
hpx::resource::partitioner_mode rp_mode = ::hpx::resource::mode_default
hpx::resource::rp_callback_type rp_callback
```

hpx/hpx_start.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/hpx_start_impl.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
bool start (std::function<int> hpx::program_options::variables_map&
> f, int argc, char **argv, init_params const &params = init_params())Main non-blocking entry point
for launching the HPX runtime system.
```

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution. This overload will not call `hpx_main`.

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as an HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution.

Return The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.

Parameters

- `f`: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `params`: [in] The parameters to the `hpx::start` function (See documentation of `hpx::init_params`)

```
bool start (std::function<int> int, char**> f, int argc, char **argv, init_params const &params = init_params())Main non-blocking entry point for launching the HPX runtime system.
```

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution. This overload will not call `hpx_main`.

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as an HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution.

Return The function returns *true* if command line processing succeeded and the runtime system was started successfully. It will return *false* otherwise.

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.

Parameters

- `f`: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.
- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `params`: [in] The parameters to the `hpx::start` function (See documentation of `hpx::init_params`)

```
bool start (int argc, char **argv, init_params const &params = init_params())Main non-blocking entry point for launching the HPX runtime system.
```

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution. This overload will not call `hpx_main`.

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as an HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution.

Return The function returns *true* if command line processing succeeded and the runtime system was started successfully. It will return *false* otherwise.

Note If the parameter *mode* is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in *argc/argv*. Otherwise it will be executed as specified by the *parametermode*.

Parameters

- *argc*: [in] The number of command line arguments passed in *argv*. This is usually the unchanged value as passed by the operating system (to *main()*).
- *argv*: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to *main()*).
- *params*: [in] The parameters to the *hpx::start* function (See documentation of [hpx::init_params](#))

bool **start** (*std::nullptr_t f*, int *argc*, char ***argv*, *init_params const ¶ms = init_params()*)
Main non-blocking entry point for launching the HPX runtime system.

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users *main()* function. It will set up the HPX runtime environment and schedule the function given by *f* as a HPX thread. It will return immediately after that. Use *hpx::wait* and *hpx::stop* to synchronize with the runtime system's execution. This overload will not call *hpx_main*.

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users *main()* function. It will set up the HPX runtime environment and schedule the function given by *f* as an HPX thread. It will return immediately after that. Use *hpx::wait* and *hpx::stop* to synchronize with the runtime system's execution.

Return The function returns *true* if command line processing succeeded and the runtime system was started successfully. It will return *false* otherwise.

Note If the parameter *mode* is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in *argc/argv*. Otherwise it will be executed as specified by the *parametermode*.

Parameters

- *f*: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If *f* is *nullptr* the HPX runtime environment will be started without invoking *f*.
- *argc*: [in] The number of command line arguments passed in *argv*. This is usually the unchanged value as passed by the operating system (to *main()*).
- *argv*: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to *main()*).
- *params*: [in] The parameters to the *hpx::start* function (See documentation of [hpx::init_params](#))

bool **start** (*init_params const ¶ms = init_params()*)
Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use *hpx::wait* and *hpx::stop* to synchronize with the runtime system's execution.

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as an HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system's execution.

Return The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. If no command line arguments are passed, console mode is assumed.

Note If no command line arguments are passed the HPX runtime system will not support any of the default command line options as described in the section ‘HPX Command Line Options’.

Parameters

- `params`: [in] The parameters to the `hpx::start` function (See documentation of `hpx::init_params`)

`hpx/hpx_suspend.hpp`

See [Public API](#) for a list of names and headers that are part of the public *HPX API*.

namespace hpx

Functions

int **suspend** (*error_code &ec = throws*)

Suspend the runtime system.

The function `hpx::suspend` is used to suspend the HPX runtime system. It can only be used when running HPX on a single locality. It will block waiting for all thread pools to be empty. This function only be called when the runtime is running, or already suspended in which case this function will do nothing.

Return This function will always return zero.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

int **resume** (*error_code &ec = throws*)

Resume the HPX runtime system.

The function `hpx::resume` is used to resume the HPX runtime system. It can only be used when running HPX on a single locality. It will block waiting for all thread pools to be resumed. This function only be called when the runtime suspended, or already running in which case this function will do nothing.

Return This function will always return zero.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

naming_base

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/naming_base/unmanaged.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace naming`

Functions

`hpx::id_type unmanaged(hpx::id_type const &id)`

The helper function `hpx::unmanaged` can be used to generate a global identifier which does not participate in the automatic garbage collection.

Return This function returns a new global id referencing the same object as the parameter `id`. The only difference is that the returned global identifier does not participate in the automatic garbage collection.

Note This function allows to apply certain optimizations to the process of memory management in HPX. It however requires the user to take full responsibility for keeping the referenced objects alive long enough.

Parameters

- `id`: [in] The id to generated the unmanaged global id from This parameter can be itself a managed or a unmanaged global id.

parcelset

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parcelset/connection_cache.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parcelset/message_handler_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parcelset/parcelhandler.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parcelset/parcelset_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

parcelset_base

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parcelset_base/parcelport.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parcelset_base/parcelset_base_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace parcelset

Typedefs

```
using parcel_write_handler_type = hpx::function<void (std::error_code const&,
```

```
          parcelset::parcel const&) >
```

The type of a function that can be registered as a parcel write handler using the function `hpx::set_parcel_write_handler`.

Note A parcel write handler is a function which is called by the parcel layer whenever a parcel has been sent by the underlying networking library and if no explicit parcel handler function was specified for the parcel.

Enums

```
enum parcelport_background_mode
```

Type of background work to perform.

Values:

```
parcelport_background_mode_flush_buffers = 0x01
```

perform buffer flush operations

```
parcelport_background_mode_send = 0x03
```

perform send operations (includes buffer flush)

```
parcelport_background_mode_receive = 0x04
```

perform receive operations

```
parcelport_background_mode_all = 0x07
```

perform all operations

hpx/parcelset_base/set_parcel_write_handler.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

performance_counters

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/performance_counters/counter_creators.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace performance_counters
```

Functions

```
bool default_counter_discoverer(counter_info const&, discover_counter_func const&, discover_counters_mode, error_code&)
```

Default discovery function for performance counters; to be registered with the counter types. It will pass the `counter_info` and the `error_code` to the supplied function.

```
bool locality_counter_discoverer(counter_info const&, discover_counter_func const&, discover_counters_mode, error_code&)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

/<objectname>(locality#<locality_id>/total)/<instancename>

```
bool locality_pool_counter_discoverer(counter_info const&, cover_counter_func const&, discover_counters_mode, error_code&)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

```
<objectname>(locality#<locality_id>/pool#<pool_name>/total)/<instancename>
```

```
bool locality0_counter_discoverer(counter_info const&, discover_counter_func const&, discover_counters_mode, error_code&)
```

Default discoverer function for AGAS performance counters; to be registered with the counter types.
It is suitable to be used for all counters following the naming scheme:

```
/<objectname>{locality#0/total}/<instancename>
```

```
bool locality_thread_counter_discoverer(counter_info const&, cover_counter_func const&, discover_counters_mode, error_code&)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

```
/<objectname>(locality#<locality_id>/worker-thread#<threadnum>)/<instancename>
```

```
bool locality_pool_thread_counter_discoverer(counter_info const &info, discover_counter_func const &f, discover_counters_mode mode, error_code &ec)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

```
/<objectname>{locality#<locality_id>/pool#<poolname>/thread#<threadnum>}/<instancename>
```

```
bool locality_pool_thread_no_total_counter_discoverer(counter_info const &info, discover_counter_func const &f, discover_counters_mode mode, error_code &ec)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

```
/<objectname>{locality#<locality_id>/pool#<poolname>/thread#<threadnum>}/<instancename>
```

This is essentially the same as above just that locality#*/total is not supported.

```
bool locality_numa_counter_discoverer(counter_info const&, cover_counter_func const&, discover_counters_mode, error_code&)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

```
/<objectname>(locality#<locality_id>/numa-node#<threadnum>)/<instancename>
```

```
naming::gid_type locality_raw_counter_creator(counter_info const&, hpx::function<std::int64_t> bool
```

> **const&**, **error_code&**Creation function for raw counters. The passed function is encapsulating the actual value to monitor. This function checks the validity of the supplied counter name, it has to follow the scheme:

```
/<objectname>(locality#<locality_id>/total)/<instancename>
```

```
naming::gid_type locality_raw_values_counter_creator(counter_info const&, hpx::function<std::vector<std::int64_t>> bool
```

> **const&**, **error_code&**

```
naming::gid_type agas_raw_counter_creator(counter_info const&, error_code&, char const*const)
```

Creation function for raw counters. The passed function is encapsulating the actual value to monitor. This function checks the validity of the supplied counter name, it has to follow the scheme:

```
/agas(<objectinstance>/total)/<instancename>
```

```
bool agas_counter_discoverer(counter_info const&, discover_counter_func const&, discover_counters_mode, error_code&)
```

Default discoverer function for performance counters; to be registered with the counter types. It is suitable to be used for all counters following the naming scheme:

```
/agas(<objectinstance>/total)/<instancename>
```

```
naming::gid_type local_action_invocation_counter_creator(counter_info const&, error_code&)
```

```
bool local_action_invocation_counter_discoverer(counter_info const&, discover_counter_func const&, discover_counters_mode, error_code&)
```

hpx/performance_counters/counters.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace performance_counters
```

Typedefs

```
typedef hpx::function<naming::gid_type(counter_info const&, error_code&) > create_counter_func
```

This declares the type of a function, which will be called by HPX whenever a new performance counter instance of a particular type needs to be created.

```
typedef hpx::function<bool(counter_info const&, error_code&) > discover_counter_func
```

This declares a type of a function, which will be passed to a *discover_counters_func* in order to be called for each discovered performance counter instance.

```
typedef hpx::function<bool(counter_info const&, discover_counter_func const&, discover_counters_mode, error_code&) > discover_counters_func
```

This declares the type of a function, which will be called by HPX whenever it needs to discover all performance counter instances of a particular type.

Enums

enum counter_type

Values:

text

text shows a variable-length text string. It does not deliver calculated values.

Formula: None Average: None Type: Text

raw

raw shows the last observed value only. It does not deliver an average.

Formula: None. Shows raw data as collected. Average: None Type: Instantaneous

monotonically_increasing

monotonically_increasing shows the cumulatively accumulated observed value. It does not deliver an average.

Formula: None. Shows cumulatively accumulated data as collected. Average: None Type: Instantaneous

average_base

average_base is used as the base data (denominator) in the computation of time or count averages for the `counter_type::average_count` and `counter_type::average_timer` counter types. This counter type collects the last observed value only.

Formula: None. This counter uses raw data in factorial calculations without delivering an output. Average: $\text{SUM}(N) / x$ Type: Instantaneous

average_count

average_count shows how many items are processed, on average, during an operation. Counters of this type display a ratio of the items processed (such as bytes sent) to the number of operations completed. The ratio is calculated by comparing the number of items processed during the last interval to the number of operations completed during the last interval.

Formula: $(N_1 - N_0) / (D_1 - D_0)$, where the numerator (N) represents the number of items processed during the last sample interval, and the denominator (D) represents the number of operations completed during the last two sample intervals. Average: $(N_x - N_0) / (D_x - D_0)$ Type: Average

aggregating

aggregating applies a function to an embedded counter instance. The embedded counter is usually evaluated repeatedly after a fixed (but configurable) time interval.

Formula: $F(N_x)$

average_timer

average_timer measures the average time it takes to complete a process or operation. Counters of this type display a ratio of the total elapsed time of the sample interval to the number of processes or operations completed during that time. This counter type measures time in ticks of the system clock. The variable F represents the number of ticks per second. The value of F is factored into the equation so that the result is displayed in seconds.

Formula: $((N_1 - N_0) / F) / (D_1 - D_0)$, where the numerator (N) represents the number of ticks counted during the last sample interval, the variable F represents the frequency of the ticks, and the denominator (D) represents the number of operations completed during the last sample interval. Average: $((N_x - N_0) / F) / (D_x - D_0)$ Type: Average

elapsed_time

elapsed_time shows the total time between when the component or process started and the time

when this value is calculated. The variable F represents the number of time units that elapse in one second. The value of F is factored into the equation so that the result is displayed in seconds.

Formula: $(D_0 - N_0) / F$, where the nominator (D) represents the current time, the numerator (N) represents the time the object was started, and the variable F represents the number of time units that elapse in one second. Average: $(D_x - N_0) / F$ Type: Difference

histogram

histogram exposes a histogram of the measured values instead of a single value as many of the other counter types. Counters of this type expose a *counter_value_array* instead of a *counter_value*. Those will also not implement the *get_counter_value()* functionality. The results are exposed through a separate *get_counter_values_array()* function.

The first three values in the returned array represent the lower and upper boundaries, and the size of the histogram buckets. All remaining values in the returned array represent the number of measurements for each of the buckets in the histogram.

raw_values

raw_values exposes an array of measured values instead of a single value as many of the other counter types. Counters of this type expose a *counter_value_array* instead of a *counter_value*. Those will also not implement the *get_counter_value()* functionality. The results are exposed through a separate *get_counter_values_array()* function.

text**raw****monotonically_increasing****average_base****average_count****aggregating****average_timer****elapsed_time****histogram****raw_values**

raw_values counter exposes an array of measured values instead of a single value as many of the other counter types. Counters of this type expose a *counter_value_array* instead of a *counter_value*. Those will also not implement the *get_counter_value()* functionality. The results are exposed through a separate *get_counter_values_array()* function.

enum counter_status

Status and error codes used by the functions related to performance counters.

Values:

status_valid_data

No error occurred, data is valid.

status_new_data

Data is valid and different from last call.

status_invalid_data

Some error occurred, data is not valid.

status_already_defined

The type or instance already has been defined.

```

status_counter_unknown
    The counter instance is unknown.

status_counter_type_unknown
    The counter type is unknown.

status_generic_error
    A unknown error occurred.

status_valid_data
    No error occurred, data is valid.

status_new_data
    Data is valid and different from last call.

status_invalid_data
    Some error occurred, data is not value.

status_already_defined
    The type or instance already has been defined.

status_counter_unknown
    The counter instance is unknown.

status_counter_type_unknown
    The counter type is unknown.

status_generic_error
    A unknown error occurred.

```

Functions

```

std::string &ensure_counter_prefix (std::string &name)

std::string ensure_counter_prefix (std::string const &counter)

std::string &remove_counter_prefix (std::string &name)

std::string remove_counter_prefix (std::string const &counter)

char const *get_counter_type_name (counter_type state)
    Return the readable name of a given counter type.

bool status_is_valid (counter_status s)

counter_status add_counter_type (counter_info const &info, error_code &ec)

hpx::id_type get_counter (std::string const &name, error_code &ec)

hpx::id_type get_counter (counter_info const &info, error_code &ec)

```

Variables

```
constexpr const char counter_prefix[] = "/counters"
constexpr std::size_t counter_prefix_len = (sizeof(counter_prefix) / sizeof(counter_prefix[0])) - 1

struct counter_info
```

Public Functions

```
counter_info(counter_type type = counter_type::raw)

counter_info(std::string const &name)

counter_info(counter_type type, std::string const &name, std::string const &help-
text = "", std::uint32_t version = HPX_PERFORMANCE_COUNTER_V1,
std::string const &uom = "")
```

Public Members

counter_type **type_**
The type of the described counter.

std::uint32_t **version_**
The version of the described counter using the 0xMMmmSSSS scheme

counter_status **status_**
The status of the counter object.

std::string **fullname_**
The full name of this counter.

std::string **helptext_**
The full descriptive text for this counter.

std::string **unit_of_measure_**
The unit of measure for this counter.

Private Functions

```
void serialize(serialization::output_archive &ar, const unsigned int)
void serialize(serialization::input_archive &ar, const unsigned int)
```

Friends

```
friend hpx::performance_counters::hpx::serialization::access

struct counter_path_elements : public hpx::performance_counters::counter_type_path_elements
#include <counters.hpp> A counter_path_elements holds the elements of a full name for a counter
instance. Generally, a full name of a counter instance has the structure:
/objectname{parentinstancename::parentindex/instancename#instanceindex}           /counter-
name#parameters
i.e. /queue{localityprefix/thread#2}/length
```

Public Types

```
typedef counter_type_path_elements base_type
```

Public Functions

```
counter_path_elements()

counter_path_elements(std::string const &objectname, std::string const &countername, std::string const &parameters, std::string const &parentname, std::string const &instancename, std::int64_t parentindex = -1, std::int64_t instanceindex = -1, bool parentinstance_is_basename = false)

counter_path_elements(std::string const &objectname, std::string const &countername, std::string const &parameters, std::string const &parentname, std::string const &subinstancename, std::string const &subinstancename, std::int64_t parentindex = -1, std::int64_t instanceindex = -1, std::int64_t subinstanceindex = -1, bool parentinstance_is_basename = false)
```

Public Members

std::string parentinstancename_
 the name of the parent instance

std::string instancename_
 the name of the object instance

std::string subinstancename_
 the name of the object sub-instance

std::int64_t parentinstanceindex_
 the parent instance index

std::int64_t instanceindex_
 the instance index

std::int64_t subinstanceindex_
 the sub-instance index

bool parentinstance_is_basename_
 the parentinstancename_

Private Functions

```
void serialize(serialization::output_archive &ar, const unsigned int)
void serialize(serialization::input_archive &ar, const unsigned int)
```

Friends

```
friend hpx::performance_counters::hpx::serialization::access
    member holds a base counter name

struct counter_type_path_elements
    #include <counters.hpp> A counter_type_path_elements holds the elements of a full name for a
    counter type. Generally, a full name of a counter type has the structure:
        /objectname/countername
        i.e. /queue/length
    Subclassed by hpx::performance_counters::counter_path_elements
```

Public Functions

```
counter_type_path_elements()

counter_type_path_elements(std::string const &objectname, std::string const &countername, std::string const &parameters)
```

Public Members

```
std::string objectname_
    the name of the performance object

std::string countername_
    contains the counter name

std::string parameters_
    optional parameters for the counter instance
```

Protected Functions

```
void serialize(serialization::output_archive &ar, const unsigned int)
void serialize(serialization::input_archive &ar, const unsigned int)
```

Friends

```
friend hpx::performance_counters::hpx::serialization::access
```

hpx/performance_counters/counters_fwd.hpp

See *Public API* for a list of names and headers that are part of the public *HPX* API.

Defines

```
HPX_COUNTER_TYPE_UNSCOPED_ENUM_DEPRECATED_MSG
HPX_PERFORMANCE_COUNTER_V1
```

namespace hpx

namespace performance_counters

Enums

enum counter_type

Values:

text

text shows a variable-length text string. It does not deliver calculated values.

Formula: None Average: None Type: Text

raw

raw shows the last observed value only. It does not deliver an average.

Formula: None. Shows raw data as collected. Average: None Type: Instantaneous

monotonically_increasing

monotonically_increasing shows the cumulatively accumulated observed value. It does not deliver an average.

Formula: None. Shows cumulatively accumulated data as collected. Average: None Type: Instantaneous

average_base

average_base is used as the base data (denominator) in the computation of time or count averages for the *counter_type::average_count* and *counter_type::average_timer* counter types. This counter type collects the last observed value only.

Formula: None. This counter uses raw data in factorial calculations without delivering an output.
Average: SUM (N) / x Type: Instantaneous

average_count

average_count shows how many items are processed, on average, during an operation. Counters of this type display a ratio of the items processed (such as bytes sent) to the number of operations completed. The ratio is calculated by comparing the number of items processed during the last interval to the number of operations completed during the last interval.

Formula: $(N_1 - N_0) / (D_1 - D_0)$, where the numerator (N) represents the number of items processed during the last sample interval, and the denominator (D) represents the number of operations completed during the last two sample intervals. Average: $(N_x - N_0) / (D_x - D_0)$ Type: Average

aggregating

aggregating applies a function to an embedded counter instance. The embedded counter is usually evaluated repeatedly after a fixed (but configurable) time interval.

Formula: F(Nx)

average_timer

average_timer measures the average time it takes to complete a process or operation. Counters of this type display a ratio of the total elapsed time of the sample interval to the number of processes

or operations completed during that time. This counter type measures time in ticks of the system clock. The variable F represents the number of ticks per second. The value of F is factored into the equation so that the result is displayed in seconds.

Formula: $((N_1 - N_0) / F) / (D_1 - D_0)$, where the numerator (N) represents the number of ticks counted during the last sample interval, the variable F represents the frequency of the ticks, and the denominator (D) represents the number of operations completed during the last sample interval.
Average: $((N_x - N_0) / F) / (D_x - D_0)$ Type: Average

elapsed_time

elapsed_time shows the total time between when the component or process started and the time when this value is calculated. The variable F represents the number of time units that elapse in one second. The value of F is factored into the equation so that the result is displayed in seconds.

Formula: $(D_0 - N_0) / F$, where the nominator (D) represents the current time, the numerator (N) represents the time the object was started, and the variable F represents the number of time units that elapse in one second. Average: $(D_x - N_0) / F$ Type: Difference

histogram

histogram exposes a histogram of the measured values instead of a single value as many of the other counter types. Counters of this type expose a *counter_value_array* instead of a *counter_value*. Those will also not implement the *get_counter_value()* functionality. The results are exposed through a separate *get_counter_values_array()* function.

The first three values in the returned array represent the lower and upper boundaries, and the size of the histogram buckets. All remaining values in the returned array represent the number of measurements for each of the buckets in the histogram.

raw_values

raw_values exposes an array of measured values instead of a single value as many of the other counter types. Counters of this type expose a *counter_value_array* instead of a *counter_value*. Those will also not implement the *get_counter_value()* functionality. The results are exposed through a separate *get_counter_values_array()* function.

text**raw****monotonically_increasing****average_base****average_count****aggregating****average_timer****elapsed_time****histogram****raw_values**

raw_values counter exposes an array of measured values instead of a single value as many of the other counter types. Counters of this type expose a *counter_value_array* instead of a *counter_value*. Those will also not implement the *get_counter_value()* functionality. The results are exposed through a separate *get_counter_values_array()* function.

enum counter_status

Values:

status_valid_data

No error occurred, data is valid.

```

status_new_data
    Data is valid and different from last call.

status_invalid_data
    Some error occurred, data is not value.

status_already_defined
    The type or instance already has been defined.

status_counter_unknown
    The counter instance is unknown.

status_counter_type_unknown
    The counter type is unknown.

status_generic_error
    A unknown error occurred.

status_valid_data
    No error occurred, data is valid.

status_new_data
    Data is valid and different from last call.

status_invalid_data
    Some error occurred, data is not value.

status_already_defined
    The type or instance already has been defined.

status_counter_unknown
    The counter instance is unknown.

status_counter_type_unknown
    The counter type is unknown.

status_generic_error
    A unknown error occurred.

enum discover_counters_mode
    Values:
        discover_counters_minimal
        discover_counters_full

```

Functions

```

constexpr bool operator< (counter_type lhs, counter_type rhs)
constexpr bool operator> (counter_type lhs, counter_type rhs)
counter_status get_counter_type_name (counter_type_path_elements const &path,
                                         std::string &result, error_code &ec = throws)
Create a full name of a counter type from the contents of the given counter_type_path_elements
instance. The generated counter type name will not contain any parameters.

counter_status get_full_counter_type_name (counter_type_path_elements const &path,
                                         std::string &result, error_code &ec =
                                         throws)
Create a full name of a counter type from the contents of the given counter_type_path_elements
instance. The generated counter type name will contain all parameters.

```

```
counter_status get_counter_name(counter_path_elements const &path, std::string &result,
                                error_code &ec = throws)
```

Create a full name of a counter from the contents of the given `counter_path_elements` instance.

```
counter_status get_counter_instance_name(counter_path_elements const &path,
                                         std::string &result, error_code &ec = throws)
```

Create a name of a counter instance from the contents of the given `counter_path_elements` instance.

```
counter_status get_counter_type_path_elements(std::string const &name,
                                              counter_type_path_elements &path,
                                              error_code &ec = throws)
```

Fill the given `counter_type_path_elements` instance from the given full name of a counter type.

```
counter_status get_counter_path_elements(std::string const &name,
                                         counter_path_elements &path, error_code
                                         &ec = throws)
```

Fill the given `counter_path_elements` instance from the given full name of a counter.

```
counter_status get_counter_name(std::string const &name, std::string &countername, error_code &ec = throws)
```

Return the canonical counter instance name from a given full instance name.

```
counter_status get_counter_type_name(std::string const &name, std::string &type_name,
                                      error_code &ec = throws)
```

Return the canonical counter type name from a given (full) instance name.

```
counter_status complement_counter_info(counter_info &info, counter_info const &type_info, error_code &ec = throws)
```

Complement the counter info if parent instance name is missing.

```
counter_status complement_counter_info(counter_info &info, error_code &ec = throws)
```

```
counter_status add_counter_type(counter_info const &info, create_counter_func const &create_counter, discover_counters_func const &discover_counters, error_code &ec = throws)
```

```
counter_status discover_counter_types(discover_counter_func const &discover_counter, discover_counters_mode mode = discover_counters_minimal, error_code &ec = throws)
```

Call the supplied function for each registered counter type.

```
counter_status discover_counter_types(std::vector<counter_info> &counters, discover_counters_mode mode = discover_counters_minimal, error_code &ec = throws)
```

Return a list of all available counter descriptions.

```
counter_status discover_counter_type(std::string const &name, discover_counter_func const &discover_counter, discover_counters_mode mode = discover_counters_minimal, error_code &ec = throws)
```

Call the supplied function for the given registered counter type.

```
counter_status discover_counter_type(counter_info const &info, discover_counter_func const &discover_counter, discover_counters_mode mode = discover_counters_minimal, error_code &ec = throws)
```

```
counter_status discover_counter_type (std::string const &name,
                                      std::vector<counter_info> &counters,
                                      discover_counters_mode mode = discover_counters_minimal,
                                      error_code &ec = throws)
```

Return a list of matching counter descriptions for the given registered counter type.

```
counter_status discover_counter_type (counter_info const &info,
                                      std::vector<counter_info> &counters,
                                      discover_counters_mode mode = discover_counters_minimal,
                                      error_code &ec = throws)
```

```
bool expand_counter_info (counter_info const&, discover_counter_func const&, error_code&)
```

call the supplied function will all expanded versions of the supplied counter info.

This function expands all locality## and worker-thread## wild cards only.

```
counter_status remove_counter_type (counter_info const &info, error_code &ec = throws)
```

Remove an existing counter type from the (local) registry.

Note This doesn't remove existing counters of this type, it just inhibits defining new counters using this type.

```
counter_status get_counter_type (std::string const &name, counter_info &info, error_code &ec = throws)
```

Retrieve the counter type for the given counter name from the (local) registry.

```
hpx::future<hpx::id_type> get_counter_async (std::string name, error_code &ec = throws)
```

Get the global id of an existing performance counter, if the counter does not exist yet, the function attempts to create the counter based on the given counter name.

```
hpx::future<hpx::id_type> get_counter_async (counter_info const &info, error_code &ec = throws)
```

Get the global id of an existing performance counter, if the counter does not exist yet, the function attempts to create the counter based on the given counter info.

```
void get_counter_infos (counter_info const &info, counter_type &type, std::string &help_text,
                        std::uint32_t &version, error_code &ec = throws)
```

Retrieve the meta data specific for the given counter instance.

```
void get_counter_infos (std::string name, counter_type &type, std::string &help_text,
                        std::uint32_t &version, error_code &ec = throws)
```

Retrieve the meta data specific for the given counter instance.

Variables

```
constexpr counter_type counter_text = counter_type::text
constexpr counter_type counter_raw = counter_type::raw
constexpr counter_type counter_monotonically_increasing = counter_type::monotonically_increasing
constexpr counter_type counter_average_base = counter_type::average_base
constexpr counter_type counter_average_count = counter_type::average_count
constexpr counter_type counter_aggregating = counter_type::aggregating
```

```
constexpr counter_type counter_average_timer = counter_type::average_timer
constexpr counter_type counter_elapsed_time = counter_type::elapsed_time
constexpr counter_type counter_raw_values = counter_type::raw_values
constexpr counter_type counter_histogram = counter_type::histogram

struct counter_value
```

Public Functions

`counter_value (std::int64_t value = 0, std::int64_t scaling = 1, bool scale_inverse = false)`

template<typename T>
`T get_value (error_code &ec = throws) const`

Retrieve the ‘real’ value of the `counter_value`, converted to the requested type `T`.

Public Members

`std::uint64_t time_`

The local time when data was collected.

`std::uint64_t count_`

The invocation counter for the data.

`std::int64_t value_`

The current counter value.

`std::int64_t scaling_`

The scaling of the current counter value.

`counter_status status_`

The status of the counter value.

`bool scale_inverse_`

If true, `value_` needs to be divided by `scaling_`, otherwise it has to be multiplied.

Private Functions

`void serialize (serialization::output_archive &ar, const unsigned int)`

`void serialize (serialization::input_archive &ar, const unsigned int)`

Friends

```
friend hpx::performance_counters::hpx::serialization::access
struct counter_values_array
```

Public Functions

```
counter_values_array(std::int64_t scaling = 1, bool scale_inverse = false)
counter_values_array(std::vector<std::int64_t> &&values, std::int64_t scaling = 1, bool
scale_inverse = false)
counter_values_array(std::vector<std::int64_t> const &values, std::int64_t scaling =
1, bool scale_inverse = false)

template<typename T>
T get_value(std::size_t index, error_code &ec = throws) const
    Retrieve the ‘real’ value of the counter_value, converted to the requested type T.
```

Public Members

```
std::uint64_t time_
    The local time when data was collected.

std::uint64_t count_
    The invocation counter for the data.

std::vector<std::int64_t> values_
    The current counter values.

std::int64_t scaling_
    The scaling of the current counter values.

counter_status status_
    The status of the counter value.

bool scale_inverse_
    If true, value_ needs to be divided by scaling_, otherwise it has to be multiplied.
```

Private Functions

```
void serialize(serialization::output_archive &ar, const unsigned int)
void serialize(serialization::input_archive &ar, const unsigned int)
```

Friends

```
friend hpx::performance_counters::hpx::serialization::access
```

hpx/performance_counters/manage_counter_type.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace performance_counters
```

Functions

```
counter_status install_counter_type (std::string const &name,  
                                     hpx::function<std::int64_t> bool  
> const &counter_value, std::string const &helptext = "", std::string const &uom = "",  
                                     counter_type type = counter_type::raw, error_code &ec = throws)Install a new generic performance  
counter type in a way, which will uninstall it automatically during shutdown.
```

The function *install_counter_type* will register a new generic counter type based on the provided function. The counter type will be automatically unregistered during system shutdown. Any consumer querying any instance of this counter type will cause the provided function to be called and the returned value to be exposed as the counter value.

The counter type is registered such that there can be one counter instance per locality. The expected naming scheme for the counter instances is: '/objectname{locality#<*>/total}/countername' where '<*>' is a zero based integer identifying the locality the counter is created on.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Return If successful, this function returns *status_valid_data*, otherwise it will either throw an exception or return an *error_code* from the enum *counter_status* (also, see note related to parameter *ec*).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Parameters

- *name*: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- *counter_value*: [in] The function to call whenever the counter value is requested by a consumer.
- *helptext*: [in, optional] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
- *uom*: [in] The unit of measure for the new performance counter type.
- *type*: [in] Type for the new performance counter type.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
counter_status install_counter_type (std::string const &name,  
                                     hpx::function<std::vector<std::int64_t>> bool  
> const &counter_value, std::string const &helptext = "", std::string const &uom = "", error_code &ec = throws)Install a new generic performance counter type returning an array of values in a way, that will uninstall it automatically during shutdown.
```

The function *install_counter_type* will register a new generic counter type that returns an array of values based on the provided function. The counter type will be automatically unregistered during system shutdown. Any consumer querying any instance of this counter type will cause the provided function to be called and the returned array value to be exposed as the counter value.

The counter type is registered such that there can be one counter instance per locality. The expected naming scheme for the counter instances is: '/objectname{locality#<*>/total}/countername' where '<*>' is a zero based integer identifying the locality the counter is created on.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Return If successful, this function returns *status_valid_data*, otherwise it will either throw an exception or return an error_code from the enum *counter_status* (also, see note related to parameter *ec*).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Parameters

- *name*: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- *counter_value*: [in] The function to call whenever the counter value (array of values) is requested by a consumer.
- *helptext*: [in, optional] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
- *uom*: [in] The unit of measure for the new performance counter type.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
void install_counter_type(std::string const &name, counter_type type, error_code &ec = throws)
```

Install a new performance counter type in a way, which will uninstall it automatically during shutdown.

The function *install_counter_type* will register a new counter type based on the provided *counter_type_info*. The counter type will be automatically unregistered during system shutdown.

Return If successful, this function returns *status_valid_data*, otherwise it will either throw an exception or return an error_code from the enum *counter_status* (also, see note related to parameter *ec*).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *name*: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- *type*: [in] The type of the counters of this counter_type.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
counter_status install_counter_type(std::string const &name, counter_type type, std::string const &helptext, std::string const &uom = "", std::uint32_t version = HPX_PERFORMANCE_COUNTER_V1, error_code &ec = throws)
```

Install a new performance counter type in a way, which will uninstall it automatically during shutdown.

The function *install_counter_type* will register a new counter type based on the provided *counter_type_info*. The counter type will be automatically unregistered during system shutdown.

Return If successful, this function returns *status_valid_data*, otherwise it will either throw an exception or return an error_code from the enum *counter_status* (also, see note related to parameter *ec*).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Parameters

- *name*: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- *type*: [in] The type of the counters of this counter_type.
- *helptext*: [in] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
- *uom*: [in] The unit of measure for the new performance counter type.
- *version*: [in] The version of the counter type. This is currently expected to be set to HPX_PERFORMANCE_COUNTER_V1.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

```
counter_status install_counter_type(std::string const &name, counter_type type,  
          std::string const &helptext, create_counter_func  
          const &create_counter, discover_counters_func  
          const &discover_counters, std::uint32_t version =  
          HPX_PERFORMANCE_COUNTER_V1, std::string  
          const &uom = "", error_code &ec = throws)
```

Install a new generic performance counter type in a way, which will uninstall it automatically during shutdown.

The function *install_counter_type* will register a new generic counter type based on the provided *counter_type_info*. The counter type will be automatically unregistered during system shutdown.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Return If successful, this function returns *status_valid_data*, otherwise it will either throw an exception or return an *error_code* from the enum *counter_status* (also, see note related to parameter *ec*).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Parameters

- *name*: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- *type*: [in] The type of the counters of this counter_type.
- *helptext*: [in] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
- *version*: [in] The version of the counter type. This is currently expected to be set to HPX_PERFORMANCE_COUNTER_V1.
- *create_counter*: [in] The function which will be called to create a new instance of this counter type.
- *discover_counters*: [in] The function will be called to discover counter instances which can be created.
- *uom*: [in] The unit of measure of the counter type (default: "")
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

hpx/performance_counters/registry.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace performance_counters

class registry
```

Public Functions

`registry()`

`void clear()`

Reset registry by deleting all stored counter types.

`counter_status add_counter_type(counter_info const &info, create_counter_func const &create_counter, discover_counters_func const &discover_counters, error_code &ec = throws)`

Add a new performance counter type to the (local) registry.

`counter_status discover_counter_types(discover_counter_func discover_counter, discover_counters_mode mode, error_code &ec = throws)`

Call the supplied function for all registered counter types.

`counter_status discover_counter_type(std::string const &fullname, discover_counter_func discover_counter, discover_counters_mode mode, error_code &ec = throws)`

Call the supplied function for the given registered counter type.

`counter_status discover_counter_type(counter_info const &info, discover_counter_func const &f, discover_counters_mode mode, error_code &ec = throws)`

`counter_status get_counter_create_function(counter_info const &info, create_counter_func &create_counter, error_code &ec = throws) const`

Retrieve the counter creation function which is associated with a given counter type.

`counter_status get_counter_discovery_function(counter_info const &info, discover_counters_func &func, error_code &ec) const`

Retrieve the counter discovery function which is associated with a given counter type.

`counter_status remove_counter_type(counter_info const &info, error_code &ec = throws)`

Remove an existing counter type from the (local) registry.

Note This doesn't remove existing counters of this type, it just inhibits defining new counters using this type.

```
counter_status create_raw_counter_value(counter_info const &info, std::int64_t *countervalue, naming::gid_type &id, error_code &ec = throws)
```

Create a new performance counter instance of type raw_counter based on given counter value.

```
counter_status create_raw_counter(counter_info const &info,  
                                hpx::function<std::int64_t>)
```

> **const &f**, *naming::gid_type &id*, *error_code &ec = throws* Create a new performance counter instance of type raw_counter based on given function returning the counter value.

```
counter_status create_raw_counter(counter_info const &info,  
                                hpx::function<std::int64_t> bool)
```

> **const &f**, *naming::gid_type &id*, *error_code &ec = throws* Create a new performance counter instance of type raw_counter based on given function returning the counter value.

```
counter_status create_raw_counter(counter_info const &info,  
                                hpx::function<std::vector<std::int64_t>>)
```

> **const &f**, *naming::gid_type &id*, *error_code &ec = throws* Create a new performance counter instance of type raw_counter based on given function returning the counter value.

```
counter_status create_raw_counter(counter_info const &info,  
                                hpx::function<std::vector<std::int64_t>> bool)
```

> **const &f**, *naming::gid_type &id*, *error_code &ec = throws* Create a new performance counter instance of type raw_counter based on given function returning the counter value.

```
counter_status create_counter(counter_info const &info, naming::gid_type &id, error_code &ec = throws)
```

Create a new performance counter instance based on given counter info.

```
counter_status create_statistics_counter(counter_info const &info, std::string const &base_counter_name,  
                                         std::vector<std::size_t> const &parameters, naming::gid_type &id, error_code &ec = throws)
```

Create a new statistics performance counter instance based on given base counter name and given base time interval (milliseconds).

```
counter_status create_arithmetics_counter(counter_info const &info,  
                                         std::vector<std::string> const &base_counter_names, naming::gid_type &id, error_code &ec = throws)
```

Create a new arithmetics performance counter instance based on given base counter names.

```
counter_status create_arithmetics_counter_extended(counter_info const &info,  
                                         std::vector<std::string> const &base_counter_names, naming::gid_type &id, error_code &ec = throws)
```

Create a new extended arithmetics performance counter instance based on given base counter names.

```
counter_status add_counter(hpx::id_type const &id, counter_info const &info, error_code &ec = throws)
```

Add an existing performance counter instance to the registry.

`counter_status remove_counter (counter_info const &info, hpx::id_type const &id, error_code &ec = throws)`
remove the existing performance counter from the registry

`counter_status get_counter_type (std::string const &name, counter_info &info, error_code &ec = throws)`
Retrieve counter type information for given counter name.

Public Static Functions

`static registry &instance ()`

Protected Functions

`counter_type_map_type::iterator locate_counter_type (std::string const &type_name)`

`counter_type_map_type::const_iterator locate_counter_type (std::string const &type_name) const`

Private Types

`typedef std::map<std::string, counter_data> counter_type_map_type`

Private Members

`counter_type_map_type countertypes_`
`struct counter_data`

Public Functions

`counter_data (counter_info const &info, create_counter_func const &create_counter, discover_counters_func const &discover_counters)`

Public Members

`counter_info info_`
`create_counter_func create_counter_`
`discover_counters_func discover_counters_`

plugin_factories

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/plugin_factories/binary_filter_factory.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_BINARY_FILTER_FACTORY (*BinaryFilter, pluginname*)

This macro is used to register a minimal component factory with `Hpx.Plugin`.

```
namespace hpx
```

```
namespace plugins
```

```
template<typename BinaryFilter>
struct binary_filter_factory : public binary_filter_factory_base
    #include <binary_filter_factory.hpp> The message_handler_factory provides a minimal implementation
    of a message handler's factory. If no additional functionality is required this type can be used to
    implement the full set of minimally required functions to be exposed by a message handler's factory
    instance.
```

Template Parameters

- `BinaryFilter`: The message handler type this factory should be responsible for.

Public Functions

```
binary_filter_factory(util::section const *global, util::section const *local, bool
    isenabled)
```

Construct a new factory instance.

Note The contents of both sections has to be cloned in order to save the configuration setting for later use.

Parameters

- `global`: [in] The pointer to a `hpx::util::section` instance referencing the settings read from the [settings] section of the global configuration file (hpx.ini) This pointer may be `nullptr` if no such section has been found.
- `local`: [in] The pointer to a `hpx::util::section` instance referencing the settings read from the section describing this component type: [hpx.components.<name>], where <name> is the instance name of the component as given in the configuration files.

```
~binary_filter_factory()
```

```
serialization::binary_filter *create(bool compress, serialization::binary_filter *next_filter =
    nullptr)
```

Create a new instance of a message handler

return Returns the newly created instance of the message handler supported by this factory

Protected Attributes

```
util::section global_settings_
util::section local_settings_
    bool isenabled_
```

hpx/plugin_factories/message_handler_factory.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/plugin_factories/parcelport_factory.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/plugin_factories/plugin_registry.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

```
HPX_REGISTER_PLUGIN_REGISTRY(...)
```

This macro is used to create and register a minimal plugin registry with `Hpx.Plugin`.

```
HPX_REGISTER_PLUGIN_REGISTRY_(...)
```

```
HPX_REGISTER_PLUGIN_REGISTRY_2(PluginType, pluginname)
```

```
HPX_REGISTER_PLUGIN_REGISTRY_4(PluginType, pluginname, pluginsection, pluginsuffix)
```

```
HPX_REGISTER_PLUGIN_REGISTRY_5(PluginType, pluginname, pluginstring, pluginsection, pluginsuffix)
```

```
namespace hpx
```

```
namespace plugins
```

```
template<typename Plugin, char const *const Name, char const *const Section, char const *const Suffix>
```

```
struct plugin_registry : public plugin_registry_base
```

#include <`plugin_registry.hpp`> The `plugin_registry` provides a minimal implementation of a plugin's registry. If no additional functionality is required this type can be used to implement the full set of minimally required functions to be exposed by a plugin's registry instance.

Template Parameters

- **Plugin:** The plugin type this registry should be responsible for.

Public Functions

`bool get_plugin_info (std::vector<std::string> &fillini)`

Return the ini-information for all contained components.

Return Returns *true* if the parameter *fillini* has been successfully initialized with the registry data of all implemented in this module.

Parameters

- *fillini*: [in] The module is expected to fill this vector with the ini-information (one line per vector element) for all components implemented in this module.

runtimme_components

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/runtimme_components/component_factory.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_COMPONENT (*type, name, mode*)

Define a component factory for a component type.

This macro is used create and to register a minimal component factory for a component type which allows it to be remotely created using the *hpx::new_<>* function.

This macro can be invoked with one, two or three arguments

Parameters

- *type*: The *type* parameter is a (fully decorated) type of the component type for which a factory should be defined.
- *name*: The *name* parameter specifies the name to use to register the factory. This should uniquely (system-wide) identify the component type. The *name* parameter must conform to the C++ identifier rules (without any namespace). If this parameter is not given, the first parameter is used.
- *mode*: The *mode* parameter has to be one of the defined enumeration values of the enumeration *hpx::components::factory_state_enum*. The default for this parameter is *hpx::components::factory_enabled*.

hpx/runtime_components/component_registry.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

Defines

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY(...)

This macro is used to create and register a minimal component registry with `Hpx.Plugin`.

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_(...)

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_2(*ComponentType*, *componentname*)

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_3(*ComponentType*, *componentname*, *state*)

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_DYNAMIC(...)

HPX REGISTER MINIMAL COMPONENT REGISTRY DYNAMIC (...)

HPX REGISTER MINIMAL COMPONENT REGISTRY DYNAMIC 2 (*ComponentType, componentname*)

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_DYNAMIC_3(*ComponentType*, *componentname*, *state*)

namespace hpx

Namespace Components

```
template<typename Component, factory_state_enum state>
struct component_registry : public component_registry_base
{
    #include "component_registry.h"  
    The component_registry provides

```

`#include <component_registry.hpp>` The `component_registry` provides a minimal implementation of a component's registry. If no additional functionality is required this type can be used to implement the full set of minimally required functions to be exposed by a component's registry instance.

Template Parameters

- Component: The component type this registry should be responsible for.

Public Functions

```
bool get_component_info(std::vector<std::string> &fillini, std::string const &filepath,  
                      bool is_static = false)
```

Return the ini-information for all contained components.

Return Returns *true* if the parameter *fillini* has been successfully initialized with the registry data of all implemented in this module.

Parameters

- **fillini**: [in] The module is expected to fill this vector with the ini-information (one line per vector element) for all components implemented in this module.

```
void register_component_type()
```

Return the unique identifier of the component type this factory is responsible for.

Return Returns the unique identifier of the component type this factory instance is responsible for. This function throws on any error.

Parameters

- locality: [in] The id of the locality this factory is responsible for.
- agas_client: [in] The AGAS client to use for component id registration (if needed).

hpx/runtime_components/components_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

```
components::server::runtime_support *get_runtime_support_ptr()
```

hpx/runtime_components/default_distribution_policy.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

Variables

```
const default_distribution_policy default_layout = {}
```

A predefined instance of the default *distribution_policy*. It will represent the local locality and will place all items to create here.

```
struct default_distribution_policy
```

#include <default_distribution_policy.hpp> This class specifies the parameters for a simple distribution policy to use for creating (and evenly distributing) a given number of items on a given set of localities.

Public Functions

```
constexpr default_distribution_policy()
```

Default-construct a new instance of a *default_distribution_policy*. This policy will represent one locality (the local locality).

```
default_distribution_policy operator()(std::vector<id_type> const &locs) const
```

Create a new *default_distribution* policy representing the given set of localities.

Parameters

- locs: [in] The list of localities the new instance should represent

```
default_distribution_policy operator()(std::vector<id_type> &&locs) const
```

Create a new *default_distribution* policy representing the given set of localities.

Parameters

- `locs`: [in] The list of localities the new instance should represent

`default_distribution_policy operator() (id_type const &loc) const`
 Create a new `default_distribution` policy representing the given locality

Parameters

- `loc`: [in] The locality the new instance should represent

`template<typename Component, typename ...Ts>`
`hpx::future<hpx::id_type> create (Ts&&... vs) const`
 Create one object on one of the localities associated by this policy instance

Note This function is part of the placement policy implemented by this class

Return A future holding the global address which represents the newly created object

Parameters

- `vs`: [in] The arguments which will be forwarded to the constructor of the new object.

`template<typename Component, typename ...Ts>`
`hpx::future<std::vector<bulk_locality_result>> bulk_create (std::size_t count, Ts&&... vs)`
`const`
 Create multiple objects on the localities associated by this policy instance

Note This function is part of the placement policy implemented by this class

Return A future holding the list of global addresses which represent the newly created objects

Parameters

- `count`: [in] The number of objects to create
- `vs`: [in] The arguments which will be forwarded to the constructors of the new objects.

`template<typename Action, typename ...Ts>`
`async_result<Action>::type async (launch policy, Ts&&... vs) const`

`template<typename Action, typename Callback, typename ...Ts>`
`async_result<Action>::type async_cb (launch policy, Callback &&cb, Ts&&... vs) const`

Note This function is part of the invocation policy implemented by this class

`template<typename Action, typename Continuation, typename ...Ts>`
`bool apply (Continuation &&c, threads::thread_priority priority, Ts&&... vs) const`
Note This function is part of the invocation policy implemented by this class

`template<typename Action, typename ...Ts>`
`bool apply (threads::thread_priority priority, Ts&&... vs) const`

`template<typename Action, typename Continuation, typename Callback, typename ...Ts>`
`bool apply_cb (Continuation &&c, threads::thread_priority priority, Callback &&cb, Ts&&... vs) const`
Note This function is part of the invocation policy implemented by this class

`template<typename Action, typename Callback, typename ...Ts>`
`bool apply_cb (threads::thread_priority priority, Callback &&cb, Ts&&... vs) const`

`std::size_t get_num_localities () const`
 Returns the number of associated localities for this distribution policy

Note This function is part of the creation policy implemented by this class

`hpx::id_type get_next_target() const`

Returns the locality which is anticipated to be used for the next async operation

```
template<typename Action>
struct async_result
#include <default_distribution_policy.hpp>
```

Note This function is part of the invocation policy implemented by this class

Public Types

```
template<>
```

```
using type = hpx::future<typename traits::promise_local_result<typename hpx::traits::extract_action<Act
```

hpx/runtime_components/derived_component_factory.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

Defines

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY(...)`

This macro is used to create and register a minimal component factory with Hpx.Plugin. This macro may be used if the registered component factory is the only factory to be exposed from a particular module. If more than one factory needs to be exposed the `HPX_REGISTER_COMPONENT_FACTORY` and `HPX_REGISTER_COMPONENT_MODULE` macros should be used instead.

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_(...)`

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_3(ComponentType, componentname, basecomponentname)`

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_4(ComponentType, componentname, basecomponentname, state)`

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DYNAMIC(...)`

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DYNAMIC_(...)`

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DYNAMIC_3(ComponentType, componentname, basecomponentname)`

`HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DYNAMIC_4(ComponentType, componentname, basecomponentname, state)`

hpx/runtime_components/new.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

Functions

template<typename Component, typename... Ts><unspecified> hpx::new_(id_type const & loc)

Create one or more new instances of the given Component type on the specified locality.

This function creates one or more new instances of the given Component type on the specified locality and returns a future object for the global address which can be used to reference the new component instance.

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```
hpx::future<hpx::id_type> f =
    hpx::new_<some_component>(hpx::find_here(), ...);
hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument *Component* represents a component type (`traits::is_component<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which can be used to retrieve the global address of the newly created component.
- If the explicit template argument *Component* represents a client side object (`traits::is_client<Component>::value` evaluates to true), the function will return a new instance of that type which can be used to refer to the newly created component instance.

Parameters

- `locality`: [in] The global address of the locality where the new instance should be created on.
- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

template<typename Component, typename... Ts><unspecified> hpx::local_new(Ts &&... vs)

Create one new instance of the given Component type on the current locality.

This function creates one new instance of the given Component type on the current locality and returns a future object for the global address which can be used to reference the new component instance.

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```
hpx::future<hpx::id_type> f =
    hpx::local_new<some_component>(...);
hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument *Component* represents a component type (`traits::is_component<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which can be used to retrieve the global address of the newly created component. If the first argument is `hpx::launch::sync` the function will directly return an `hpx::id_type`.
- If the explicit template argument *Component* represents a client side object (`traits::is_client<Component>::value` evaluates to true), the function will

return a new instance of that type which can be used to refer to the newly created component instance.

Note The difference of this function to `hpx::new_` is that it can be used in cases where the supplied arguments are non-copyable and non-movable. All operations are guaranteed to be local only.

Parameters

- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

```
template<typename Component, typename... Ts><unspecified> hpx::new_(id_type const & loc)
```

Create multiple new instances of the given Component type on the specified locality.

This function creates multiple new instances of the given Component type on the specified locality and returns a future object for the global address which can be used to reference the new component instance.

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```
hpx::future<std::vector<hpx::id_type>> f =  
    hpx::new_<some_component[]>(hpx::find_here(), 10, ...);  
hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument `Component` represents an array of a component type (i.e. `Component[]`, where `traits::is_component<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which holds a `std::vector<hpx::id_type>`, where each of the items in this vector is a global address of one of the newly created components.
- If the explicit template argument `Component` represents an array of a client side object type (i.e. `Component[]`, where `traits::is_client<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which holds a `std::vector<hpx::id_type>`, where each of the items in this vector is a client side instance of the given type, each representing one of the newly created components.

Parameters

- `locality`: [in] The global address of the locality where the new instance should be created on.
- `count`: [in] The number of component instances to create
- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

```
template<typename Component, typename DistPolicy, typename... Ts><unspecified> hpx::new_
```

Create one or more new instances of the given Component type based on the given distribution policy.

This function creates one or more new instances of the given Component type on the localities defined by the given distribution policy and returns a future object for global address which can be used to reference the new component instance(s).

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```
hpx::future<hpx::id_type> f =  
    hpx::new_<some_component>(hpx::default_layout, ...);  
hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument *Component* represents a component type (`traits::is_component<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which can be used to retrieve the global address of the newly created component.
- If the explicit template argument *Component* represents a client side object (`traits::is_client<Component>::value` evaluates to true), the function will return a new instance of that type which can be used to refer to the newly created component instance.

Parameters

- `policy`: [in] The distribution policy used to decide where to place the newly created.
- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

template<typename Component, typename DistPolicy, typename... Ts><unspecified> hpx::new_<Component>(DistPolicy, Ts...)

Create multiple new instances of the given Component type on the localities as defined by the given distribution policy.

This function creates multiple new instances of the given Component type on the localities defined by the given distribution policy and returns a future object for the global address which can be used to reference the new component instance.

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```
hpx::future<std::vector<hpx::id_type> > f =
    hpx::new_<some_component[]>(hpx::default_layout, 10, ...);
hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument *Component* represents an array of a component type (i.e. `Component[]`, where `traits::is_component<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which holds a `std::vector<hpx::id_type>`, where each of the items in this vector is a global address of one of the newly created components.
- If the explicit template argument *Component* represents an array of a client side object type (i.e. `Component[]`, where `traits::is_client<Component>::value` evaluates to true), the function will return an `hpx::future` object instance which holds a `std::vector<hpx::id_type>`, where each of the items in this vector is a client side instance of the given type, each representing one of the newly created components.

Parameters

- `policy`: [in] The distribution policy used to decide where to place the newly created.
- `count`: [in] The number of component instances to create
- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

runtime_distributed

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/runtime_distributed.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

class runtime_distributed : public runtime

#include <runtime_distributed.hpp> The *runtime* class encapsulates the HPX runtime system in a simple to use way. It makes sure all required parts of the HPX runtime system are properly initialized.

Public Functions

hpx::runtime_distributed::runtime_distributed(util::runtime_configuration & rtcfg,
Construct a new HPX runtime instance

Parameters

- `locality_mode`: [in] This is the mode the given runtime instance should be executed in.

~runtime_distributed()

The destructor makes sure all HPX runtime services are properly shut down before exiting.

int start (hpx::function<hpx_main_function_type> const &func, bool blocking = false)
Start the runtime system.

Return If a blocking is a true, this function will return the value as returned as the result of the invocation of the function object given by the parameter `func`. Otherwise it will return zero.

Parameters

- `func`: [in] This is the main function of an HPX application. It will be scheduled for execution by the thread manager as soon as the runtime has been initialized. This function is expected to expose an interface as defined by the typedef `hpx_main_function_type`.
- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been stopped. If this parameter is *true* the function `runtime::start` will call `runtime::wait` internally.

int start (bool blocking = false)
Start the runtime system.

Return If a blocking is a true, this function will return the value as returned as the result of the invocation of the function object given by the parameter `func`. Otherwise it will return zero.

Parameters

- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been stopped. If this parameter is *true* the function `runtime::start` will call `runtime::wait` internally .

int wait ()
Wait for the shutdown action to be executed.

Return This function will return the value as returned as the result of the invocation of the function object given by the parameter `func`.

```
void stop (bool blocking = true)
    Initiate termination of the runtime system.
```

Parameters

- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been fully stopped. If this parameter is *false* then this call will initiate the stop action but will return immediately. Use a second call to stop with this parameter set to *true* to wait for all internal work to be completed.

```
int finalize (double shutdown_timeout)
```

```
void stop_helper (bool blocking, std::condition_variable &cond, std::mutex &mtx)
    Stop the runtime system, wait for termination.
```

Parameters

- `blocking`: [in] This allows to control whether this call blocks until the runtime system has been fully stopped. If this parameter is *false* then this call will initiate the stop action but will return immediately. Use a second call to stop with this parameter set to *true* to wait for all internal work to be completed.

```
int suspend ()
    Suspend the runtime system.
```

```
int resume ()
    Resume the runtime system.
```

```
bool report_error (std::size_t num_thread, std::exception_ptr const &e, bool terminate_all =
    true)
    Report a non-recoverable error to the runtime system.
```

Parameters

- `num_thread`: [in] The number of the operating system thread the error has been detected in.
- `e`: [in] This is an instance encapsulating an exception which lead to this function call.
- `terminate_all`: [in] Kill all localities attached to the currently running application (default: *true*)

```
bool report_error (std::exception_ptr const &e, bool terminate_all = true)
    Report a non-recoverable error to the runtime system.
```

Note This function will retrieve the number of the current shepherd thread and forward to the `report_error` function above.

Parameters

- `e`: [in] This is an instance encapsulating an exception which lead to this function call.
- `terminate_all`: [in] Kill all localities attached to the currently running application (default: *true*)

```
int run (hpx::function<hpx_main_function_type> const &func)
```

Run the HPX runtime system, use the given function for the main *thread* and block waiting for all threads to finish.

Note The parameter *func* is optional. If no function is supplied, the runtime system will simply wait for the shutdown action without explicitly executing any main thread.

Return This function will return the value as returned as the result of the invocation of the function object given by the parameter *func*.

Parameters

- *func*: [in] This is the main function of an HPX application. It will be scheduled for execution by the thread manager as soon as the runtime has been initialized. This function is expected to expose an interface as defined by the typedef *hpx_main_function_type*. This parameter is optional and defaults to none main thread function, in which case all threads have to be scheduled explicitly.

`int run ()`

Run the HPX runtime system, initially use the given number of (OS) threads in the thread-manager and block waiting for all threads to finish.

Return This function will always return 0 (zero).

`bool is_networking_enabled ()`

`template<typename F>`

`components::server::console_error_dispatcher::sink_type set_error_sink (F && sink)`

`performance_counters::registry &get_counter_registry ()`

Allow access to the registry counter registry instance used by the HPX runtime.

`performance_counters::registry const &get_counter_registry () const`

Allow access to the registry counter registry instance used by the HPX runtime.

`void register_counter_types ()`

Install all performance counters related to this runtime instance.

`void register_query_counters (std::shared_ptr<util::query_counters> const &active_counters)`

`void start_active_counters (error_code &ec = throws)`

`void stop_active_counters (error_code &ec = throws)`

`void reset_active_counters (error_code &ec = throws)`

`void reinit_active_counters (bool reset = true, error_code &ec = throws)`

`void evaluate_active_counters (bool reset = false, char const *description = nullptr, error_code &ec = throws)`

`void stop_evaluating_counters (bool terminate = false)`

`naming::resolver_client &get_agas_client ()`

Allow access to the AGAS client instance used by the HPX runtime.

`hpx::threads::threadmanager &get_thread_manager ()`

Allow access to the thread manager instance used by the HPX runtime.

`applier::applier &get_applier ()`

Allow access to the applier instance used by the HPX runtime.

`std::string here () const`

Returns a string of the locality endpoints (usable in debug output)

```

naming::address_type get_runtime_support_lva() const
naming::gid_type get_next_id(std::size_t count = 1)

void init_id_pool_range()

util::unique_id_ranges &get_id_pool()

void initialize_agas()
    Initialize AGAS operation.

void add_pre_startup_function(startup_function_type f)
    Add a function to be executed inside a HPX thread before hpx_main but guaranteed to be executed before any startup function registered with add_startup_function.

```

Note The difference to a startup function is that all pre-startup functions will be (system-wide) executed before any startup function.

Parameters

- f: The function ‘f’ will be called from inside a HPX thread before hpx_main is executed. This is very useful to setup the runtime environment of the application (install performance counters, etc.)

```

void add_startup_function(startup_function_type f)
    Add a function to be executed inside a HPX thread before hpx_main

```

Parameters

- f: The function ‘f’ will be called from inside a HPX thread before hpx_main is executed. This is very useful to setup the runtime environment of the application (install performance counters, etc.)

```

void add_pre_shutdown_function(shutdown_function_type f)
    Add a function to be executed inside a HPX thread during hpx::finalize, but guaranteed before any of the shutdown functions is executed.

```

Note The difference to a shutdown function is that all pre-shutdown functions will be (system-wide) executed before any shutdown function.

Parameters

- f: The function ‘f’ will be called from inside a HPX thread while hpx::finalize is executed. This is very useful to tear down the runtime environment of the application (uninstall performance counters, etc.)

```

void add_shutdown_function(shutdown_function_type f)
    Add a function to be executed inside a HPX thread during hpx::finalize

```

Parameters

- f: The function ‘f’ will be called from inside a HPX thread while hpx::finalize is executed. This is very useful to tear down the runtime environment of the application (uninstall performance counters, etc.)

```

hpx::util::io_service_pool *get_thread_pool(char const *name)
    Access one of the internal thread pools (io_service instances) HPX is using to perform specific tasks. The three possible values for the argument name are “main_pool”, “io_pool”, “parcel_pool”, and “timer_pool”. For any other argument value the function will return zero.

```

```
bool register_thread(char const *name, std::size_t num = 0, bool service_thread = true,  
                      error_code &ec = throws)  
    Register an external OS-thread with HPX.  
  
notification_policy_type get_notification_policy(char const *prefix, runtime_local::os_thread_type type)  
    Generate a new notification policy instance for the given thread name prefix  
  
std::uint32_t get_locality_id(error_code &ec) const  
  
std::size_t get_num_worker_threads() const  
  
std::uint32_t get_num_localities(hpx::launch::sync_policy, error_code &ec) const  
  
std::uint32_t get_initial_num_localities() const  
  
hpx::future<std::uint32_t> get_num_localities() const  
  
std::string get_locality_name() const  
  
std::uint32_t get_num_localities(hpx::launch::sync_policy, components::component_type type,  
                                error_code &ec) const  
  
hpx::future<std::uint32_t> get_num_localities(components::component_type type) const  
  
std::uint32_t assign_cores(std::string const &locality_basename, std::uint32_t num_threads)  
  
std::uint32_t assign_cores()
```

Private Types

```
using used_cores_map_type = std::map<std::string, std::uint32_t>
```

Private Functions

```
threads::thread_result_type run_helper(hpx::function<runtime::hpx_main_function_type>  
                                         const &func, int &result)  
  
void init_global_data()  
  
void deinit_global_data()  
  
void wait_helper(std::mutex &mtx, std::condition_variable &cond, bool &running)  
  
void init_tss_helper(char const *context, runtime_local::os_thread_type type, std::size_t  
                     local_thread_num, std::size_t global_thread_num, char const  
                     *pool_name, char const *postfix, bool service_thread)  
  
void deinit_tss_helper(char const *context, std::size_t num)  
  
void init_tss_ex(std::string const &locality, char const *context, runtime_local::os_thread_type type,  
                  std::size_t local_thread_num, std::size_t global_thread_num, char const *pool_name,  
                  char const *postfix, bool service_thread, error_code &ec)
```

Private Members

```
runtime_mode mode_
util::unique_id_ranges id_pool_
naming::resolver_client agas_client_
applier::applier applier_
used_cores_map_type used_cores_map_
std::unique_ptr<components::server::runtime_support> runtime_support_
std::shared_ptr<util::query_counters> active_counters_
int (*pre_main_)(runtime_mode)
void (*post_main_)()
```

Private Static Functions

```
static void default_errorsink(std::string const&)
```

hpx/runtime_distributed/applier.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace applier
```

```
class applier
```

```
#include <applier.hpp> The applier class is used to decide whether a particular action has to be issued on a local or a remote resource. If the target component is local a new thread will be created, if the target is remote a parcel will be sent.
```

Public Functions

```
HPX_NON_COPYABLE(applier)
```

```
applier()
```

```
void init(threads::threadmanager &tm)
```

```
~applier()
```

```
void initialize(std::uint64_t rts)
```

```
threads::threadmanager &get_thread_manager()
```

Access the *thread-manager* instance associated with this *applier*.

This function returns a reference to the thread manager this *applier* instance has been created with.

```
naming::gid_type const &get_raw_locality(error_code &ec = throws) const
Allow access to the locality of the locality this applier instance is associated with.
```

This function returns a reference to the locality this applier instance is associated with.

```
std::uint32_t get_locality_id(error_code &ec = throws) const
```

Allow access to the id of the locality this applier instance is associated with.

This function returns a reference to the id of the locality this applier instance is associated with.

```
bool get_raw_remote_localities(std::vector<naming::gid_type> &locality_ids,
                               components::component_type type = components::component_invalid,
                               error_code &ec = throws) const
```

Return list of localities of all remote localities registered with the AGAS service for a specific component type.

This function returns a list of all remote localities (all localities known to AGAS except the local one) supporting the given component type.

Return The function returns *true* if there is at least one remote locality known to the AGASService (!prefixes.empty()).

Parameters

- `locality_ids`: [out] The reference to a vector of id_types filled by the function.
- `type`: [in] The type of the component which needs to exist on the returned localities.

```
bool get_remote_localities(std::vector<hpx::id_type> &locality_ids,
                           components::component_type type = components::component_invalid,
                           error_code &ec = throws) const
```

```
bool get_raw_localities(std::vector<naming::gid_type> &locality_ids,
                        components::component_type type = components::component_invalid) const
```

Return list of locality_ids of all localities registered with the AGAS service for a specific component type.

This function returns a list of all localities (all localities known to AGAS except the local one) supporting the given component type.

Return The function returns *true* if there is at least one remote locality known to the AGASService (!prefixes.empty()).

Parameters

- `locality_ids`: [out] The reference to a vector of id_types filled by the function.
- `type`: [in] The type of the component which needs to exist on the returned localities.

```
bool get_localities(std::vector<hpx::id_type> &locality_ids, error_code &ec = throws) const
```

```
bool get_localities(std::vector<hpx::id_type> &locality_ids, components::component_type type, error_code &ec = throws) const
```

```
naming::gid_type const &get_runtime_support_raw_gid() const
```

By convention the runtime_support has a gid identical to the prefix of the locality the runtime_support is responsible for

```
hpx::id_type const &get_runtime_support_gid() const
```

By convention the runtime_support has a gid identical to the prefix of the locality the runtime_support is responsible for

Private Members

```
threads::threadmanager *thread_manager_
hpx::id_type runtime_support_id_
```

hpx/runtime_distributed/applier_fwd.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace applier
```

Functions

```
applier &get_applier()
```

The function `get_applier` returns a reference to the (thread specific) applier instance.

```
applier *get_applier_ptr()
```

The function `get_applier` returns a pointer to the (thread specific) applier instance. The returned pointer is NULL if the current thread is not known to HPX or if the runtime system is not active.

```
namespace applier
```

The namespace `applier` contains all definitions needed for the class `hpx::applier::applier` and its related functionality. This namespace is part of the HPX core module.

hpx/runtime_distributed/server/copy_component.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

```
namespace server
```

Functions

```
template<typename Component>
```

```
future<hpx::id_type> copy_component_here(hpx::id_type const &to_copy)
```

```
template<typename Component>
```

```
future<hpx::id_type> copy_component(hpx::id_type const &to_copy, hpx::id_type
const &target_locality)
```

hpx/runtime_distributed/find_all_localities.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

hpx::id_type find_root_locality(error_code &ec = throws)

Return the global id representing the root locality.

The function [find_root_locality\(\)](#) can be used to retrieve the global id usable to refer to the root locality. The root locality is the locality where the main AGAS service is hosted.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global id representing the root locality for this application.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note This function will return meaningful results only if called from an HPX-thread. It will return *hpx::invalid_id* otherwise.

See [hpx::find_all_localities\(\)](#), [hpx::find_locality\(\)](#)

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

std::vector<hpx::id_type> find_all_localities(error_code &ec = throws)

Return the list of global ids representing all localities available to this application.

The function [find_all_localities\(\)](#) can be used to retrieve the global ids of all localities currently available to this application.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the localities currently available to this application.

Note As long as *ec* is not pre-initialized to *hpx::throws* this function doesn't throw but returns the result code using the parameter *ec*. Otherwise it throws an instance of *hpx::exception*.

Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See [hpx::find_here\(\)](#), [hpx::find_locality\(\)](#)

Parameters

- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

`std::vector<hpx::id_type> find_remote_localities(error_code &ec = throws)`

Return the list of locality ids of remote localities supporting the given component type. By default this function will return the list of all remote localities (all but the current locality).

The function `find_remote_localities()` can be used to retrieve the global ids of all remote localities currently available to this application (i.e. all localities except the current one).

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the remote localities currently available to this application.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See `hpx::find_here()`, `hpx::find_locality()`

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

hpx/runtime_distributed/find_here.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`hpx::id_type find_here(error_code &ec = throws)`

Return the global id representing this locality.

The function `find_here()` can be used to retrieve the global id usable to refer to the current locality.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global id representing the locality this function has been called on.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note This function will return meaningful results only if called from an HPX-thread. It will return `hpx::invalid_id` otherwise.

See `hpx::find_all_localities()`, `hpx::find_locality()`

Parameters

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

hpx/runtime_distributed/find_localities.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`std::vector<hpx::id_type> find_all_localities(components::component_type type, error_code &ec = throws)`

Return the list of global ids representing all localities available to this application which support the given component type.

The function `find_all_localities()` can be used to retrieve the global ids of all localities currently available to this application which support the creation of instances of the given component type.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the localities currently available to this application which support the creation of instances of the given component type. If no localities supporting the given component type are currently available, this function will return an empty vector.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See [hpx::find_here\(\)](#), [hpx::find_locality\(\)](#)

Parameters

- `type`: [in] The type of the components for which the function should return the available localities.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`std::vector<hpx::id_type> find_remote_localities(components::component_type type, error_code &ec = throws)`

Return the list of locality ids of remote localities supporting the given component type. By default this function will return the list of all remote localities (all but the current locality).

The function `find_remote_localities()` can be used to retrieve the global ids of all remote localities currently available to this application (i.e. all localities except the current one) which support the creation of instances of the given component type.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the remote localities currently available to this application.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See [hpx::find_here\(\)](#), [hpx::find_locality\(\)](#)

Parameters

- `type`: [in] The type of the components for which the function should return the available remote localities.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`hpx::id_type find_locality(components::component_type type, error_code &ec = throws)`

Return the global id representing an arbitrary locality which supports the given component type.

The function `find_locality()` can be used to retrieve the global id of an arbitrary locality currently available to this application which supports the creation of instances of the given component type.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global id representing an arbitrary locality currently available to this application which supports the creation of instances of the given component type. If no locality supporting the given component type is currently available, this function will return `hpx::invalid_id`.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn't throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note This function will return meaningful results only if called from an HPX-thread. It will return `hpx::invalid_id` otherwise.

See [hpx::find_here\(\)](#), [hpx::find_all_localities\(\)](#)

Parameters

- `type`: [in] The type of the components for which the function should return any available locality.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

hpx/runtime_distributed/get_locality_name.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

Functions

`future<std::string> get_locality_name(hpx::id_type const &id)`

Return the name of the referenced locality.

This function returns a future referring to the name for the locality of the given id.

Return This function returns the name for the locality of the given id. The name is retrieved from the underlying networking layer and may be different for different parcel ports.

See `std::string get_locality_name()`

Parameters

- `id`: [in] The global id of the locality for which the name should be retrieved

hpx/runtime_distributed/get_num_localities.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

Functions

`hpx::future<std::uint32_t> get_num_localities(components::component_type t)`

Asynchronously return the number of localities which are currently registered for the running application.

The function *get_num_localities* asynchronously returns the number of localities currently connected to the console which support the creation of the given component type. The returned future represents the actual result.

Note This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

See *hpx::find_all_localities, hpx::get_num_localities*

Parameters

- *t*: The component type for which the number of connected localities should be retrieved.

`std::uint32_t get_num_localities(launch::sync_policy, components::component_type t, error_code &ec = throws)`

Synchronously return the number of localities which are currently registered for the running application.

The function *get_num_localities* returns the number of localities currently connected to the console which support the creation of the given component type. The returned future represents the actual result.

Note This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

See *hpx::find_all_localities, hpx::get_num_localities*

Parameters

- *t*: The component type for which the number of connected localities should be retrieved.
- *ec*: [in,out] this represents the error status on exit, if this is pre-initialized to *hpx::throws* the function will throw on error instead.

hpx/runtime_distributed/migrate_component.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

namespace hpx

namespace components

Functions

```
template<typename Component, typename DistPolicy>
future<hpx::id_type> migrate (hpx::id_type const &to_migrate, DistPolicy const &policy)
Migrate the given component to the specified target locality
```

The function `migrate<Component>` will migrate the component referenced by `to_migrate` to the locality specified with `target_locality`. It returns a future referring to the migrated component instance.

Return A future representing the global id of the migrated component instance. This should be the same as `migrate_to`.

Parameters

- `to_migrate`: [in] The client side representation of the component to migrate.
- `policy`: [in] A distribution policy which will be used to determine the locality to migrate this object to.

Template Parameters

- `Component`: Specifies the component type of the component to migrate.
- `DistPolicy`: Specifies the distribution policy to use to determine the destination locality.

```
template<typename Derived, typename Stub, typename DistPolicy>
Derived migrate (client_base<Derived, Stub> const &to_migrate, DistPolicy const &policy)
Migrate the given component to the specified target locality
```

The function `migrate<Component>` will migrate the component referenced by `to_migrate` to the locality specified with `target_locality`. It returns a future referring to the migrated component instance.

Return A future representing the global id of the migrated component instance. This should be the same as `migrate_to`.

Parameters

- `to_migrate`: [in] The client side representation of the component to migrate.
- `policy`: [in] A distribution policy which will be used to determine the locality to migrate this object to.

Template Parameters

- `Derived`: Specifies the component type of the component to migrate.
- `DistPolicy`: Specifies the distribution policy to use to determine the destination locality.

```
template<typename Component>
future<hpx::id_type> migrate (hpx::id_type const &to_migrate, hpx::id_type const &target_locality)
Migrate the component with the given id to the specified target locality
```

The function `migrate<Component>` will migrate the component referenced by `to_migrate` to the locality specified with `target_locality`. It returns a future referring to the migrated component instance.

Return A future representing the global id of the migrated component instance. This should be the same as `migrate_to`.

Parameters

- `to_migrate`: [in] The global id of the component to migrate.
- `target_locality`: [in] The locality where the component should be migrated to.

Template Parameters

- `Component`: Specifies the component type of the component to migrate.

```
template<typename Derived, typename Stub>
```

```
Derived migrate(client_base<Derived, Stub> const &to_migrate, hpx::id_type const &target_locality)
```

Migrate the given component to the specified target locality

The function *migrate*<*Component*> will migrate the component referenced by *to_migrate* to the locality specified with *target_locality*. It returns a future referring to the migrated component instance.

Return A client side representation of representing of the migrated component instance. This should be the same as *migrate_to*.

Parameters

- *to_migrate*: [in] The client side representation of the component to migrate.
- *target_locality*: [in] The id of the locality to migrate this object to.

Template Parameters

- *Derived*: Specifies the component type of the component to migrate.

hpx/runtime_distributed/runtime_fwd.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

hpx/runtime_distributed/stubs/runtime_support.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace components
```

```
namespace stubs
```

struct runtime_support

Subclassed by hpx::components::runtime_support

Public Static Functions

```
template<typename Component, typename ...Ts>
```

```
static hpx::future<hpx::id_type> create_component_async(hpx::id_type const &gid, Ts&&... vs)
```

Create a new component *type* using the *runtime_support* with the given *targetgid*. This is a non-blocking call. The caller needs to call *future::get* on the result of this function to obtain the global id of the newly created object.

```
template<typename Component, typename ...Ts>
```

```
static hpx::id_type create_component(hpx::id_type const &gid, Ts&&... vs)
```

Create a new component *type* using the *runtime_support* with the given *targetgid*. Block for the creation to finish.

```
template<typename Component, typename ...Ts>
```

```
static hpx::future<std::vector<hpx::id_type>> bulk_create_component_colocated_async(hpx::id_type const &gid, std::size_t count, Ts&&... vs)
```

Create multiple new components *type* using the *runtime_support* colocated with the given *targetgid*. This is a non-blocking call.

```
template<typename Component, typename ...Ts>
static std::vector<hpx::id_type> bulk_create_component_colocated(hpx::id_type const &gid, std::size_t count, Ts&&... vs)
```

Create multiple new components *type* using the *runtime_support* colocated with the given *targetgid*. Block for the creation to finish.

```
template<typename Component, typename ...Ts>
static hpx::future<std::vector<hpx::id_type>> bulk_create_component_async(hpx::id_type const &gid, std::size_t count, Ts&&... vs)
```

Create multiple new components *type* using the *runtime_support* on the given locality. This is a non-blocking call.

```
template<typename Component, typename ...Ts>
static std::vector<hpx::id_type> bulk_create_component(hpx::id_type const &gid, std::size_t count, Ts&&... vs)
```

Create multiple new components *type* using the *runtime_support* on the given locality. Block for the creation to finish.

```
template<typename Component, typename ...Ts>
static hpx::future<hpx::id_type> create_component_colocated_async(hpx::id_type const &gid, Ts&&... vs)
```

Create a new component *type* using the *runtime_support* with the given *targetgid*. This is a non-blocking call. The caller needs to call *future::get* on the result of this function to obtain the global id of the newly created object.

```
template<typename Component, typename ...Ts>
static hpx::id_type create_component_colocated(hpx::id_type const &gid, Ts&&... vs)
```

Create a new component *type* using the *runtime_support* with the given *targetgid*. Block for the creation to finish.

```
template<typename Component>
```

```
static hpx::future<hpx::id_type> copy_create_component_async(hpx::id_type
                                                               const &gid,
                                                               std::shared_ptr<Component>
                                                               const &p,
                                                               bool local_op)

template<typename Component>
static hpx::id_type copy_create_component(hpx::id_type      const      &gid,
                                         std::shared_ptr<Component> const
                                         &p, bool local_op)

template<typename Component>
static hpx::future<hpx::id_type> migrate_component_async(hpx::id_type const
                                                       &target_locality,
                                                       std::shared_ptr<Component>
                                                       const      &p,
                                                       hpx::id_type const
                                                       &to_migrate)

template<typename Component, typename DistPolicy>
static hpx::future<hpx::id_type> migrate_component_async(DistPolicy
                                                       const      &policy,
                                                       std::shared_ptr<Component>
                                                       const      &p,
                                                       hpx::id_type const
                                                       &to_migrate)

template<typename Component, typename Target>
static hpx::id_type migrate_component(Target const &target, hpx::id_type const
                                       &to_migrate, std::shared_ptr<Component>
                                       const &p)

static hpx::future<int> load_components_async(hpx::id_type const &gid)

static int load_components(hpx::id_type const &gid)

static hpx::future<void> call_startup_functions_async(hpx::id_type
                                                       const      &gid,    bool
                                                       pre_startup)

static void call_startup_functions(hpx::id_type      const      &gid,    bool
                                   pre_startup)

static hpx::future<void> shutdown_async(hpx::id_type const &targetgid, double
                                         timeout = -1)
    Shutdown the given runtime system.

static void shutdown(hpx::id_type const &targetgid, double timeout = -1)

static void shutdown_all(hpx::id_type const &targetgid, double timeout = -1)
    Shutdown the runtime systems of all localities.

static void shutdown_all(double timeout = -1)

static hpx::future<void> terminate_async(hpx::id_type const &targetgid)
    Retrieve configuration information.

    Terminate the given runtime system

static void terminate(hpx::id_type const &targetgid)
```

```
static void terminate_all (hpx::id_type const &targetgid)
    Terminate the runtime systems of all localities.

static void terminate_all ()

static void garbage_collect_non_blocking (hpx::id_type const &targetgid)

static hpx::future<void> garbage_collect_async (hpx::id_type const &target-
    gid)

static void garbage_collect (hpx::id_type const &targetgid)

static hpx::future<hpx::id_type> create_performance_counter_async (hpx::id_type
    target-
    gid,
    perfor-
    mance_counters::counter_in-
    const
    &info)

static hpx::id_type create_performance_counter (hpx::id_type targetgid, perfor-
    mance_counters::counter_info
    const &info, error_code &ec
    = throws)

static hpx::future<util::section> get_config_async (hpx::id_type const &target-
    gid)
    Retrieve configuration information.

static void get_config (hpx::id_type const &targetgid, util::section &ini)

static void remove_from_connection_cache_async (hpx::id_type const &tar-
    get, naming::gid_type
    const &gid,
    parcelset::endpoints_type
    const &endpoints)
```

segmented_algorithms

See *Public API* for a list of names and headers that are part of the public HPX API.

`hpx/parallel/segmented_algorithms/adjacent_difference.hpp`

See *Public API* for a list of names and headers that are part of the public HPX API.

`namespace hpx`

```
namespace segmented
```

Functions

```
template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op>
hpx::parallel::util::detail::algorithm_result_t<ExPolicy, FwdIter2> tag_invoke(hpx::adjacent_difference_t,
    ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, Op &&op)

template<typename InIter1, typename InIter2, typename Op>
InIter2 tag_invoke(hpx::adjacent_difference_t, InIter1 first, InIter1 last, InIter2 dest, Op &&op)
```

hpx/parallel/segmented_algorithms/adjacent_find.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename InIter, typename Pred>
InIter tag_invoke(hpx::adjacent_find_t, InIter first, InIter last, Pred &&pred = Pred())

template<typename ExPolicy, typename SegIter, typename Pred>
hpx::parallel::util::detail::algorithm_result<ExPolicy, SegIter>::type tag_invoke(hpx::adjacent_find_t,
    ExPolicy &&policy,
    SegIter first,
    SegIter last,
    Pred &&pred)
```

hpx/parallel/segmented_algorithms/all_any_none.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```

template<typename InIter, typename F>
bool tag_invoke(hpx::none_of_t, InIter first, InIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type tag_invoke(hpx::none_of_t,
    ExPolicy &&policy, SegIter first,
    SegIter last, F
    &&f)

template<typename InIter, typename F>
bool tag_invoke(hpx::any_of_t, InIter first, InIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type tag_invoke(hpx::any_of_t,
    ExPolicy &&policy, SegIter first,
    SegIter last, F
    &&f)

template<typename InIter, typename F>
bool tag_invoke(hpx::all_of_t, InIter first, InIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, bool>::type tag_invoke(hpx::all_of_t,
    ExPolicy &&policy, SegIter first,
    SegIter last, F
    &&f)

```

`hpx/parallel/segmented_algorithms/count.hpp`

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace segmented`

Functions

```

template<typename InIter, typename T>
std::iterator_traits<InIter>::difference_type tag_invoke(hpx::count_t, InIter first, InIter last, T
    const &value)

template<typename ExPolicy, typename SegIter, typename T>

```

```
hpx::parallel::util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<SegIter>::difference_type>::type tag
```

```
template<typename InIter, typename F>
std::iterator_traits<InIter>::difference_type tag_invoke(hpx::count_if_t, InIter first, InIter last, F
&&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<SegIter>::difference_type>::type tag
```

[hpx/parallel/segmented_algorithms/exclusive_scan.hpp](#)

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename InIter, typename OutIter, typename T, typename Op = std::plus<T>>
OutIter tag_invoke(hpx::exclusive_scan_t, InIter first, InIter last, OutIter dest, T init, Op &&op
= Op())

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Op = std::plus<T>>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter2>::type tag_invoke(hpx::exclusive_scan_t,
ExPolicy &&policy, FwdIter1 first,
FwdIter1 last,
FwdIter2 dest, T
init, Op &&op =
Op())
```

hpx/parallel/segmented_algorithms/fill.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

hpx/parallel/segmented_algorithms/for_each.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

`namespace hpx`

`namespace segmented`

Functions

```
template<typename InIter, typename F>
InIter tag_invoke(hpx::for_each_t, InIter first, InIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, SegIter>::type tag_invoke(hpx::for_each_t,
ExPolicy
&&policy,
SegIter first,
SegIter last, F
&&f)

template<typename InIter, typename Size, typename F>
InIter tag_invoke(hpx::for_each_n_t, InIter first, Size count, F &&f)

template<typename ExPolicy, typename SegIter, typename Size, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, SegIter>::type tag_invoke(hpx::for_each_n_t,
ExPolicy
&&policy,
SegIter first,
Size count, F
&&f)
```

hpx/parallel/segmented_algorithms/generate.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename SegIter, typename F>
SegIter tag_invoke(hpx::generate_t, SegIter first, SegIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
parallel::util::detail::algorithm_result<ExPolicy, SegIter>::type tag_invoke(hpx::generate_t,
ExPolicy &&policy,
SegIter first, SegIter
last, F &&f)
```

hpx/parallel/segmented_algorithms/inclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename InIter, typename OutIter, typename Op = std::plus<typename std::iterator_traits<InIter>::value_type>
OutIter tag_invoke(hpx::inclusive_scan_t, InIter first, InIter last, OutIter dest, Op &&op = Op())

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op = std::plus<typename std::iterator_traits<FwdIter1>::value_type>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter2>::type tag_invoke(hpx::inclusive_scan_t,
ExPolicy &&policy,
FwdIter1 first,
FwdIter1 last,
FwdIter2 dest, Op
&&op = Op())

template<typename InIter, typename OutIter, typename Op, typename T>
OutIter tag_invoke(hpx::inclusive_scan_t, InIter first, InIter last, OutIter dest, Op &&op, T
&&init)

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op, typename T>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter2>::type tag_invoke(hpx::inclusive_scan_t,
ExPolicy &&policy,
FwdIter1 first,
FwdIter1 last,
FwdIter2 dest, Op
&&op, T &&init)
```

hpx/parallel/segmented_algorithms/minmax.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
    namespace parallel
```

Typedefs

```
template<typename T>
using minmax_element_result = hpx::parallel::util::min_max_result<T>
```

```
namespace segmented
```

Typedefs

```
template<typename T>
using minmax_element_result = hpx::parallel::util::min_max_result<T>
```

Functions

```
template<typename SegIter, typename F>
SegIter tag_invoke(hpx::min_element_t, SegIter first, SegIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result_t<ExPolicy, SegIter> tag_invoke(hpx::min_element_t,
    ExPolicy &&policy, SegIter first,
    SegIter last, F
    &&f)

template<typename SegIter, typename F>
SegIter tag_invoke(hpx::max_element_t, SegIter first, SegIter last, F &&f)

template<typename ExPolicy, typename SegIter, typename F>
hpx::parallel::util::detail::algorithm_result_t<ExPolicy, SegIter> tag_invoke(hpx::max_element_t,
    ExPolicy &&policy, SegIter first,
    SegIter last, F
    &&f)

template<typename SegIter, typename F>
minmax_element_result<SegIter> tag_invoke(hpx::minmax_element_t, SegIter first, SegIter last,
    F &&f)

template<typename ExPolicy, typename SegIter, typename F>
```

```
hpx::parallel::util::detail::algorithm_result_t<ExPolicy, minmax_element_result<SegIter>> tag_invoke(hpx::minmax_e  
Ex-  
Pol-  
icy  
&&pol-  
icy,  
Se-  
gIter  
first,  
Se-  
gIter  
last,  
F  
&&f)
```

hpx/parallel/segmented_algorithms/reduce.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename InIterB, typename InIterE, typename T, typename F>  
T tag_invoke(hpx::reduce_t, InIterB first, InIterE last, T init, F &&f)  
  
template<typename ExPolicy, typename InIterB, typename InIterE, typename T, typename F>  
parallel::util::detail::algorithm_result<ExPolicy, T>::type tag_invoke(hpx::reduce_t, ExPolicy  
&&policy, InIterB first,  
InIterE last, T init, F  
&&f)
```

hpx/parallel/segmented_algorithms/transform.hpp

See *Public API* for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename SegIter, typename OutIter, typename F>
hpx::parallel::util::in_out_result<SegIter, OutIter> tag_invoke(hpx::transform_t, SegIter first,
SegIter last, OutIter dest, F
&&f)
```

```
template<typename ExPolicy, typename SegIter, typename OutIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::parallel::util::in_out_result<SegIter, OutIter>>::type tag_invoke
```

```
template<typename InIter1, typename InIter2, typename OutIter, typename F>
hpx::parallel::util::in_in_out_result<InIter1, InIter2, OutIter> tag_invoke(hpx::transform_t,
InIter1 first1, InIter1
last1, InIter2 first2,
OutIter dest, F &&f)
```

```
template<typename ExPolicy, typename InIter1, typename InIter2, typename OutIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::parallel::util::in_in_out_result<InIter1, InIter2, OutIter>>::type tag_invoke
```

```
template<typename InIter1, typename InIter2, typename OutIter, typename F>
```

```
hpx::parallel::util::in_in_out_result<InIter1, InIter2, OutIter> tag_invoke(hpx::transform_t,
    InIter1 first1, InIter1 last1, InIter2 first2,
    InIter2 last2, OutIter dest, F &&f)

template<typename ExPolicy, typename InIter1, typename InIter2, typename OutIter, typename F>
hpx::parallel::util::detail::algorithm_result<ExPolicy, hpx::parallel::util::in_in_out_result<InIter1, InIter2, OutIter>>::type
```

hpx/parallel/segmented_algorithms/transform_exclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename InIter, typename OutIter, typename T, typename Op, typename Conv>
OutIter tag_invoke(hpx::transform_exclusive_scan_t, InIter first, InIter last, OutIter dest, T init,
    Op &&op, Conv &&conv)

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Op, typename Conv>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter2>::type tag_invoke(hpx::transform_exclusive_scan_t,
    ExPolicy &&policy, FwdIter1 first,
    FwdIter1 last,
    FwdIter2 dest, T
    init, Op &&op,
    Conv &&conv)
```

hpx/parallel/segmented_algorithms/transform_inclusive_scan.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename InIter, typename OutIter, typename Op, typename Conv>
OutIter tag_invoke(hpx::transform_inclusive_scan_t, InIter first, InIter last, OutIter dest, Op
&&op, Conv &&conv)

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op, typename Conv>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter2>::type tag_invoke(hpx::transform_inclusive_scan_t,
ExPolicy &&policy, FwdIter1 first,
FwdIter1 last,
FwdIter2 dest,
Op &&op, Conv
&&conv)

template<typename InIter, typename OutIter, typename T, typename Op, typename Conv>
OutIter tag_invoke(hpx::transform_inclusive_scan_t, InIter first, InIter last, OutIter dest, Op
&&op, Conv &&conv, T init)

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Op, typename Conv>
parallel::util::detail::algorithm_result<ExPolicy, FwdIter2>::type tag_invoke(hpx::transform_inclusive_scan_t,
ExPolicy &&policy, FwdIter1 first,
FwdIter1 last,
FwdIter2 dest,
Op &&op, Conv
&&conv, T init)
```

hpx/parallel/segmented_algorithms/transform_reduce.hpp

See [Public API](#) for a list of names and headers that are part of the public HPX API.

```
namespace hpx
```

```
namespace segmented
```

Functions

```
template<typename SegIter, typename T, typename Reduce, typename Convert>
std::decay<T> tag_invoke (hpx::transform_reduce_t, SegIter first, SegIter last, T &&init, Reduce
&&red_op, Convert &&conv_op)

template<typename ExPolicy, typename SegIter, typename T, typename Reduce, typename Convert>
parallel::util::detail::algorithm_result<ExPolicy, typename std::decay<T>::type>::type tag_invoke (hpx::transform_re-
Ex-
Pol-
icy
&&pol-
icy,
Se-
gIter
first,
Se-
gIter
last,
T
&&init,
Re-
duce
&&red_op,
Con-
vert
&&conv_op)

template<typename FwdIter1, typename FwdIter2, typename T, typename Reduce, typename Convert>
T tag_invoke (hpx::transform_reduce_t, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, T init,
Reduce &&red_op, Convert &&conv_op)

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Reduce, typename Convert>
parallel::util::detail::algorithm_result<ExPolicy, T>::type tag_invoke (hpx::transform_reduce_t,
ExPolicy &&policy,
FwdIter1 first1, FwdIter1
last1, FwdIter2 first2, T
init, Reduce &&red_op,
Convert &&conv_op)
```

2.9 Contributing to HPX

HPX development happens on Github. The following sections are a collection of useful information related to *HPX* development.

2.9.1 Contributing to HPX

The main source of information to understand the process of how to contribute to HPX can be found in [this document⁴⁷⁰](#). This is a living document that is constantly updated with relevant information.

2.9.2 HPX governance model

The *HPX* project is a meritocratic, consensus-based community project. Anyone with an interest in the project can join the community, contribute to the project design and participate in the decision making process. [This document⁴⁷¹](#) describes how that participation takes place and how to set about earning merit within the project community.

2.9.3 Release procedure for HPX

Below is a step by step procedure for making an *HPX* release. We aim to produce two releases per year: one in March-April, and one in September-October.

This is a living document and may not be totally current or accurate. It is an attempt to capture current practices in making an *HPX* release. Please update it as appropriate.

One way to use this procedure is to print a copy and check off the lines as they are completed to avoid confusion.

1. Notify developers that a release is imminent.
2. For minor and major releases: create and check out a new branch at an appropriate point on `master` with the name `release-major.minor.X.major` and `minor` should be the major and minor versions of the release. For patch releases: check out the corresponding `release-major.minor.X` branch.
3. Write release notes in `docs/sphinx/releases/whats_new_$.VERSION.rst`. Keep adding merged PRs and closed issues to this until just before the release is made. Use `tools/generate_pr_issue_list.sh` to generate the lists. Add the new release notes to the table of contents in `docs/sphinx/releases.rst`.
4. Build the docs, and proof-read them. Update any documentation that may have changed, and correct any typos. Pay special attention to:
 - `$HPX_SOURCE/README.rst`
 - Update grant information
 - `docs/sphinx/releases/whats_new_$.VERSION.rst`
 - `docs/sphinx/about_hpx/people.rst`
 - Update collaborators
 - Update grant information
5. This step does not apply to patch releases. For both APEX and libCDS:

⁴⁷⁰ <https://github.com/STELLAR-GROUP/hpx/blob/master/.github/CONTRIBUTING.md>

⁴⁷¹ <http://hpx.stellar-group.org/documents/governance/>

- Change the release branch to be the most current release tag available in the APEX/libCDS git_external section in the main CMakeLists.txt. Please contact the maintainers of the respective packages to generate a new release to synchronize with the HPX release (APEX⁴⁷², libCDS⁴⁷³).
6. Make sure HPX_VERSION_MAJOR/MINOR/SUBMINOR in CMakeLists.txt contain the correct values. Change them if needed.
 7. Change version references in CITATION.cff. There are two occurrences.
 8. This step does not apply to patch releases. Remove features which have been deprecated for at least 2 releases. This involves removing build options which enable those features from the main CMakeLists.txt and also deleting all related code and tests from the main source tree.
- The general deprecation policy involves a three-step process we have to go through in order to introduce a breaking change:
- a. First release cycle: add a build option that allows for explicitly disabling any old (now deprecated) code.
 - b. Second release cycle: turn this build option OFF by default.
 - c. Third release cycle: completely remove the old code.
- The main CMakeLists.txt contains a comment indicating for which version the breaking change was introduced first. In the case of deprecated features which don't have a replacement yet, we keep them around in case (like Vc for example).
9. Update the minimum required versions if necessary (compilers, dependencies, etc.) in building_hpx.rst.
 10. Verify that the Jenkins setups for the release branch on Rostam and Piz Daint are running and do not display any errors.
 11. Repeat the following steps until satisfied with the release.
 1. Change HPX_VERSION_TAG in CMakeLists.txt to -rcN, where N is the current iteration of this step. Start with -rc1.
 2. Create a pre-release on GitHub using the script tools/roll_release.sh. This script automatically tag with the corresponding release number. The script requires that you have the STELLAR Group signing key.
 3. This step is not necessary for patch releases. Notify hpx-users@stellar-group.org and stellar@cct.lsu.edu of the availability of the release candidate. Ask users to test the candidate by checking out the release candidate tag.
 4. Allow at least a week for testing of the release candidate.
 - Use git merge when possible, and fall back to git cherry-pick when needed. For patch releases git cherry-pick is most likely your only choice if there have been significant unrelated changes on master since the previous release.
 - Go back to the first step when enough patches have been added.
 - If there are no more patches, continue to make the final release.
 12. Update any occurrences of the latest stable release to refer to the version about to be released. For example, quickstart.rst contains instructions to check out the latest stable tag. Make sure that refers to the new version.
 13. Add a new entry to the RPM changelog (cmake/packaging/rpm/Changelog.txt) with the new version number and a link to the corresponding changelog.
 14. Change HPX_VERSION_TAG in CMakeLists.txt to an empty string.

⁴⁷² <http://github.com/UO-OACISS/xpress-apex>

⁴⁷³ <https://github.com/STELLAR-GROUP/libcds>

15. Add the release date to the caption of the current “What’s New” section in the docs, and change the value of `HPX_VERSION_DATE` in `CMakeLists.txt`.
16. Create a release on GitHub using the script `tools/roll_release.sh`. This script automatically tag the with the corresponding release number. The script requires that you have the STELLAR Group signing key.
17. Update the websites (hpx.stellar-group.org⁴⁷⁴, stellar-group.org⁴⁷⁵ and stellar.cct.lsu.edu⁴⁷⁶). You can login on wordpress through *this page* <<https://hpx.stellar-group.org/wp-login.php>>. You can update the pages with the following:
 - Update links on the downloads page. Link to the release on GitHub.
 - Documentation links on the docs page (link to generated documentation on GitHub Pages). Follow the style of previous releases.
 - A new blog post announcing the release, which links to downloads and the “What’s New” section in the documentation (see previous releases for examples).
18. Merge release branch into master.
19. Post-release cleanup. Create a new pull request against master with the following changes:
 1. Modify the release procedure if necessary.
 2. Change `HPX_VERSION_TAG` in `CMakeLists.txt` back to `-trunk`.
 3. Increment `HPX_VERSION_MINOR` in `CMakeLists.txt`.
20. Update Vcpkg (<https://github.com/Microsoft/vcpkg>) to pull from latest release.
 - Update version number in `CONTROL`
 - Update tag and SHA512 to that of the new release
21. Update spack (<https://github.com/spack/spack>) with the latest HPX package.
 - Update version number in `hpx/package.py` and SHA256 to that of the new release
22. Announce the release on `hpx-users@stellar-group.org`, `stellar@cct.lsu.edu`, `allcct@cct.lsu.edu`, `faculty@csc.lsu.edu`, `faculty@ece.lsu.edu`, `xpress@crest.iu.edu`, the *HPX* Slack channel, the IRC channel, our list of external collaborators, isocpp.org, reddit.com, HPC Wire, Inside HPC, Heise Online, and a CCT press release.
23. Beer and pizza.

2.9.4 Testing HPX

To ensure correctness of *HPX*, we ship a large variety of unit and regression tests. The tests are driven by the CTest⁴⁷⁷ tool and are executed automatically on each commit to the *HPX* Github⁴⁷⁸ repository. In addition, it is encouraged to run the test suite manually to ensure proper operation on your target system. If a test fails for your platform, we highly recommend submitting an issue on our *HPX* Issues⁴⁷⁹ tracker with detailed information about the target system.

⁴⁷⁴ <https://hpx.stellar-group.org>

⁴⁷⁵ <https://stellar-group.org>

⁴⁷⁶ <https://stellar.cct.lsu.edu>

⁴⁷⁷ <https://gitlab.kitware.com/cmake/community/wikis/doc/ctest/Testing-With-CTest>

⁴⁷⁸ <https://github.com/STELLAR-GROUP/hpx/>

⁴⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues>

Running tests manually

Running the tests manually is as easy as typing `make tests && make test`. This will build all tests and run them once the tests are built successfully. After the tests have been built, you can invoke separate tests with the help of the `ctest` command. You can list all available test targets using `make help | grep tests`. Please see the CTest Documentation⁴⁸⁰ for further details.

Running performance tests

We run performance tests on Piz Daint for each pull request using Jenkins. To run those performance tests locally or on Piz Daint, a script is provided under `tools/perftests_ci/local_run.sh` (to be run in the build directory specifying the *HPX* source directory as the argument to the script, default is `$HOME/projects/hpx_perftests_ci`.

Adding new performance tests

To add a new performance test, you need to wrap the portion of code to benchmark with `hpx::util::perftests_report`, passing the test name, the executor name and the function to time (can be a lambda). This facility is used to output the time results in a json format (format needed to compare the results and plot them). To effectively print them at the end of your test, call `hpx::util::perftests_print_times`. To see an example of use, see `future_overhead_report.cpp`. Finally, you can add the test to the CI report editing the `hpx_targets` variable for the executable name and the `hpx_test_options` variable for the corresponding options to use for the run in the performance test script `.jenkins/cscs-perftests/launch_perftests.sh`. And then run the `tools/perftests_ci/local_run.sh` script to get a reference json run (use the name of the test) to be added in the `tools/perftests_ci/perftest/references/daint_default` directory.

Issue tracker

If you stumble over a bug or missing feature in *HPX*, please submit an issue to our [HPX Issues⁴⁸¹](#) page. For more information on how to submit support requests or other means of getting in contact with the developers, please see the [Support Website⁴⁸²](#) page.

Continuous testing

In addition to manual testing, we run automated tests on various platforms. We also run tests on all pull requests using both [CircleCI⁴⁸³](#) and a combination of [CDash⁴⁸⁴](#) and [pycicle⁴⁸⁵](#). You can see the dashboards here: [CircleCI HPX dashboard⁴⁸⁶](#) and [CDash HPX dashboard⁴⁸⁷](#).

⁴⁸⁰ <https://www.cmake.org/cmake/help/latest/manual/ctest.1.html>

⁴⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues>

⁴⁸² <https://stellar.cct.lsu.edu/support/>

⁴⁸³ <https://circleci.com>

⁴⁸⁴ <https://www.kitware.com/cdash/project/about.html>

⁴⁸⁵ <https://github.com/biddisco/pycicle/>

⁴⁸⁶ <https://circleci.com/gh/STELLAR-GROUP/hpx>

⁴⁸⁷ <https://cdash.cscs.ch/index.php?project=HPX>

2.9.5 Using docker for development

Although it can often be useful to set up a local development environment with system-provided or self-built dependencies, Docker⁴⁸⁸ provides a convenient alternative to quickly get all the dependencies needed to start development of *HPX*. Our testing setup on CircleCI⁴⁸⁹ uses a docker image to run all tests.

To get started you need to install Docker⁴⁹⁰ using whatever means is most convenient on your system. Once you have Docker⁴⁹¹ installed, you can pull or directly run the docker image. The image is based on Debian and Clang, and can be found on Docker Hub⁴⁹². To start a container using the *HPX* build environment, run:

```
$ docker run --interactive --tty stellargroup/build_env:latest bash
```

You are now in an environment where all the *HPX* build and runtime dependencies are present. You can install additional packages according to your own needs. Please see the Docker Documentation⁴⁹³ for more information on using Docker⁴⁹⁴.

Warning: All changes made within the container are lost when the container is closed. If you want files to persist (e.g., the *HPX* source tree) after closing the container, you can bind directories from the host system into the container (see Docker Documentation (Bind mounts)⁴⁹⁵).

2.9.6 Documentation

This documentation is built using Sphinx⁴⁹⁶, and an automatically generated API reference using Doxygen⁴⁹⁷ and Breathe⁴⁹⁸.

We always welcome suggestions on how to improve our documentation, as well as pull requests with corrections and additions.

Prerequisites

To build the *HPX* documentation, you need recent versions of the following packages:

- python3
- sphinx 3.5.4 (Python package)
- sphinx_rtd_theme (Python package)
- breathe 4.16.0 (Python package)
- doxygen

If the Python⁴⁹⁹ dependencies are not available through your system package manager, you can install them using the Python package manager pip:

⁴⁸⁸ <https://www.docker.com>

⁴⁸⁹ <https://circleci.com>

⁴⁹⁰ <https://www.docker.com>

⁴⁹¹ <https://www.docker.com>

⁴⁹² https://hub.docker.com/r/stellargroup/build_env/

⁴⁹³ <https://docs.docker.com/>

⁴⁹⁴ <https://www.docker.com>

⁴⁹⁵ <https://docs.docker.com/storage/bind-mounts/>

⁴⁹⁶ <http://www.sphinx-doc.org>

⁴⁹⁷ <https://www.doxygen.org>

⁴⁹⁸ <https://breathe.readthedocs.io/en/latest>

⁴⁹⁹ <https://www.python.org>

```
pip install --user sphinx sphinx_rtd_theme breathe
```

You may need to set the following CMake variables to make sure CMake can find the required dependencies.

DOXYGEN_ROOT:PATH

Specifies where to look for the installation of the Doxygen⁵⁰⁰ tool.

SPHINX_ROOT:PATH

Specifies where to look for the installation of the Sphinx⁵⁰¹ tool.

BREATHE_APIDOC_ROOT:PATH

Specifies where to look for the installation of the Breathe⁵⁰² tool.

Building documentation

Enable building of the documentation by setting `HPX_WITH_DOCUMENTATION=ON` during CMake⁵⁰³ configuration. To build the documentation, build the `docs` target using your build tool. The default output format is HTML documentation. You can choose alternative output formats (single-page HTML, PDF, and man) with the `HPX_WITH_DOCUMENTATION_OUTPUT_FORMATS` CMake option.

Note: If you add new source files to the Sphinx documentation, you have to run CMake again to have the files included in the build.

Style guide

The documentation is written using reStructuredText. These are the conventions used for formatting the documentation:

- Use, at most, 80 characters per line.
- Top-level headings use over- and underlines with =.
- Sub-headings use only underlines with characters in decreasing level of importance: =, – and ..
- Use sentence case in headings.
- Refer to common terminology using :term:`Component`.
- Indent content of directives (.. directive::) by three spaces.
- For C++ code samples at the end of paragraphs, use :: and indent the code sample by 4 spaces.
 - For other languages (or if you don't want a colon at the end of the paragraph), use .. code-block:: language and indent by three spaces as with other directives.
- Use .. list-table:: to wrap tables with a lot of text in cells.

⁵⁰⁰ <https://www.doxygen.org>

⁵⁰¹ <http://www.sphinx-doc.org>

⁵⁰² <https://breathe.readthedocs.io/en/latest>

⁵⁰³ <https://www.cmake.org>

API documentation

The source code is documented using Doxygen. If you add new API documentation either to existing or new source files, make sure that you add the documented source files to the `doxygen_dependencies` variable in `docs/CMakeLists.txt`.

2.9.7 Module structure

This section explains the structure of an *HPX* module.

The tool `create_library_skeleton.py`⁵⁰⁴ can be used to generate a basic skeleton. To create a library skeleton, run the tool in the `libs` subdirectory with the module name as an argument:

```
$ ./create_library_skeleton <lib_name>
```

This creates a skeleton with the necessary files for an *HPX* module. It will not create any actual source files. The structure of this skeleton is as follows:

- <lib_name>/
 - README.rst
 - CMakeLists.txt
 - cmake
 - docs/
 - * index.rst
 - examples/
 - * CMakeLists.txt
 - include/
 - * hpx/
 - <lib_name>
 - src/
 - * CMakeLists.txt
 - tests/
 - * CMakeLists.txt
 - * unit/
 - CMakeLists.txt
 - * regressions/
 - CMakeLists.txt
 - * performance/
 - CMakeLists.txt

⁵⁰⁴ https://github.com/STELLAR-GROUP/hpx/blob/master/libs/create_library_skeleton.py

A README.rst should be always included which explains the basic purpose of the library and a link to the generated documentation.

A main CMakeLists.txt is created in the root directory of the module. By default it contains a call to add_hpx_module which takes care of most of the boilerplate required for a module. You only need to fill in the source and header files in most cases.

add_hpx_module requires a module name. Optional flags are:

Optional single-value arguments are:

- INSTALL_BINARIES: Install the resulting library.

Optional multi-value arguments-are:

- SOURCES: List of source files.
- HEADERS: List of header files.
- COMPAT_HEADERS: List of compatibility header files.
- DEPENDENCIES: Libraries that this module depends on, such as other modules.
- CMAKE_SUBDIRS: List of subdirectories to add to the module.

The include directory should contain only headers that other libraries need. For each of those headers, an automatic header test to check for self containment will be generated. Private headers should be placed under the src directory. This allows for clear separation. The cmake subdirectory may include additional CMake⁵⁰⁵ scripts needed to generate the respective build configurations.

Compatibility headers (forwarding headers for headers whose location is changed when creating a module, if moving them from the main library) should be placed in an include_compatibility directory. This directory is not created by default.

Documentation is placed in the docs folder. A empty skeleton for the index is created, which is picked up by the main build system and will be part of the generated documentation. Each header inside the include directory will automatically be processed by Doxygen and included into the documentation.

Tests are placed in suitable subdirectories of tests.

When in doubt, consult existing modules for examples on how to structure the module.

Finding circular dependencies

Our CI will perform a check to see if there are circular dependencies between modules. In cases where it's not clear what is causing the circular dependency, running the cpp-dependencies⁵⁰⁶ tool manually can be helpful. It can give you detailed information on exactly which files are causing the circular dependency. If you do not have the cpp-dependencies tool already installed, one way of obtaining it is by using our docker image. This way you will have exactly the same environment as on the CI. See [Using docker for development](#) for details on how to use the docker image.

To produce the graph produced by CI run the following command (HPX_SOURCE is assumed to hold the path to the HPX source directory):

```
$ cpp-dependencies --dir $HPX_SOURCE/libs --graph-cycles circular_dependencies.dot
```

This will produce a dot file in the current directory. You can inspect this manually with a text editor. You can also convert this to an image if you have graphviz installed:

⁵⁰⁵ <https://www.cmake.org>

⁵⁰⁶ <https://github.com/tomtom-international/cpp-dependencies>

```
$ dot circular_dependencies.dot -Tsvg -o circular_dependencies.svg
```

This produces an `svg` file in the current directory which shows the circular dependencies. Note that if there are no cycles the image will be empty.

You can use `cpp-dependencies` to print the include paths between two modules.

```
$ cpp-dependencies --dir $HPX_SOURCE/libs --shortest <from> <to>
```

prints all possible paths from the module `<from>` to the module `<to>`. For example, as most modules depend on `config`, the following should give you a long list of paths from `algorithms` to `config`:

```
$ cpp-dependencies --dir $HPX_SOURCE/libs --shortest algorithms config
```

The following should report that it can't find a path between the two modules:

```
$ cpp-dependencies --dir $HPX_SOURCE/libs --shortest config algorithms
```

2.10 Releases

2.10.1 HPX V1.8.0 (May 18, 2022)

With HPX parallel algorithms been fully adapted to C++20 the new release achieves full conformance with C++20 concurrency and parallelism facilities. HPX now supports all of the algorithms as specified by C++20. We have added support for vectorization to more of our algorithms. Much work has been done towards implementing P2300 (“`std::execution`”) and implementing the underlying senders/receivers facilities. Finally, The new release comes with a brand new documentation interface!

General changes

- The new documentation can now be found on our webpage: <https://hpx-docs.stellar-group.org>. This includes a completely new and user-friendly interface environment along with restructuring of certain components. The content in the “Quick start”, “Manual” and “Examples” was improved, while the “Build system” page was adapted to include necessary information for newcomers.
- With the vectorization support available in modern hardware architectures HPX now provides new data-parallel vector execution policies `hpx::execution::simd` and `hpx::execution::par_simd` that enable significant speed-up of our parallel algorithm implementations. The following algorithms now support SIMD execution:
 - `copy`, `copy_n`
 - `generate`
 - `adjacent_difference`, `adjacent_find`
 - `all_of`, `any_of`, `none_of`
 - `equal`, `mismatch`,
 - `inner_product`
 - `count`, `count_if`
 - `fill`, `fill_n`
 - `find`, `find_end`, `find_first_of`, `find_if`, `find_if_not`

- `for_each`, `for_each_n`
 - `generate`, `generate_n`.
- Based on top of P2300 the HPX parallel algorithms now support the pipeline syntax towards an effort to unify their usage along with senders/receivers. The HPX parallel algorithms can now bind with senders/receivers using the pipeline operator.
- Several changes took place on the executors provided by HPX:
- The executors now support the `num_cores` options in order for the user to be able to specify the desired number of cores to be used in the correspodning execution.
- The `scheduler` executor was implemented on top of senders/receivers and can be used with all HPX facilities that schedule new work, such as parallel algorithms, `hpx::async`, `hpx::dataflow`, etc.
- The performance of `fork_join_executor` was improved.
- The following algorithms have been added/adapted to be C++20 conformant:
 - `min_element`
 - `max_element`
 - `minmax_element`
 - `starts_with`
 - `ends_with`
 - `swap_ranges`
 - `unique`
 - `unique_copy`
 - `rotate`
 - `rotate_copy`
 - `sort`
 - `shift_left`
 - `shift_right`
 - `stable_sort`
 - `partition`
 - `partition_copy`
 - `stable_partition`
 - `adjacent_difference`
 - `nth_element`
 - `partial_sort`
 - `partial_sort_copy`.
- `HPX_FORWARD`/`HPX_MOVE` macros were introduced that replaced the `std::move` and `std::forward` facilities that in the library code.
- Hangs on distributed barrier were fixed.
- The performance of `scan_partitioner` was improved.
- Support was added for `thread_priority` to the `parallel_execution_policy`

- Regarding senders/receivers and the P2300 proposal various actions took place. `stop_token` was adapted to the recent proposal version (`in_place_stop_token` was introduced). Also `hint`, `annotation`, `priority` and `stacksize` properties were added to the scheduler executor. Stop support was added to `when_all`. Support for completion signatures was added. The following schedulers and algorithms were added:

- `get_completion_scheduler`
- `any_sender` and `unique_any_sender`
- `split sender`
- `transform_mpi sender`
- `transfer sender`
- `let_error`, `let_stopped`
- `get_env` and related environment queries
- `schedule`, `set_value`, `set_error`, `set_done`, `start` and `connect` are now proper customization points as defined in P2300.

- Several namespaces were altered towards conformance with C++20. Compatibility layers have been added and the old versions will be removed in next releases. The namespace changes are the following:

- `hpx::parallel::induction/reduction` were moved into namespace `hpx::experimental`
- `for_loop` and friends were moved into namespace `hpx::experimental`.
- `hpx::util::optional` and friends were moved into namespace `hpx`.
- `hpx::lcos::barrier` has been moved into the `hpx::distributed` namespace and `hpx::lcos::local::cpp20_barrier` has been renamed to `barrier` and moved into the `hpx` namespace.
- `hpx::lcos::latch` has been moved into the `hpx::distributed` namespace and `lcos::local::latch` has been moved into the `hpx` namespace. The `count_down_and_wait()` functionality of `latch` has been renamed to `arrive_and_wait()`.
- `hpx::util::unique_function_nonserv` has been renamed to `hpx::move_only_function`.
- `hpx::util::unique_function` has been renamed to `hpx::distributed::move_only_function`.
- `hpx::util::function` has been renamed to `hpx::distributed::function`.
- `hpx::util::function_nonserv` has been renamed to `hpx::function`.
- `hpx::util::function_ref` have been moved to namespace `hpx`.
- `hpx::lcos::split_future` changed namespace and is now used as `hpx::split_future`.
- `hpx::lcos::local::counting_semaphore` has been deprecated and `hpx::lcos::local::cpp20_counting_semaphore` has been renamed to `hpx::counting_semaphore`.
- `hpx::lcos::local::cpp20_binary_semaphore` has been renamed to `hpx::binary_semaphore`.
- `hpx::lcos::local::sliding_semaphore` has been renamed to `hpx::sliding_semaphore` and
- `hpx::lcos::local::sliding_semaphore_var` has been renamed to `hpx::sliding_semaphore_var`.
- `hpx::lcos::local::spinlock` has been renamed to `hpx::spinlock`.

- `hpx::lcos::local::mutex` has been renamed to `hpx::mutex`.
 - `hpx::lcos::local::timed_mutex` has been renamed to `hpx::timed_mutex`.
 - `hpx::lcos::local::no_mutex` has been renamed to `hpx::no_mutex`.
 - `hpx::lcos::local::recursive_mutex` has been renamed to `hpx::recursive_mutex`.
 - `hpx::lcos::local::shared_mutex` has been renamed to `hpx::shared_mutex`.
 - `hpx::lcos::local::upgrade_lock` has been renamed to `hpx::upgrade_lock`.
 - `hpx::lcos::local::upgrade_to_unique_lock` has been renamed to `hpx::upgrade_to_unique_lock`.
 - `hpx::lcos::local::condition_variable` has been renamed to `hpx::condition_variable`. `hpx::lcos::local::condition_variable_var` has been renamed to `hpx::condition_variable_var`.
 - `hpx::lcos::local::once_flag` has been renamed to `hpx::once_flag`, and `hpx::lcos::local::call_once` has been renamed to `hpx::call_once`.
- The new LCI (Lightweight Communication Interface) parcelport was added that supports irregular and asynchronous applications like graph analysis, sparse linear algebra, modern parallel architectures etc. Major features include:
 - Support for advanced communication primitives like two sided send/recv and one sided remote put.
 - Better multi-threaded performance.
 - Explicit user control of communication resource.
 - Flexible signaling mechanisms (synchronizer, completion queue, active message handler).
 - The following CMake flags were added, mostly to support using HPX as a backend for SHAD (<https://github.com/pnml/SHAD>). Please note that these options enable questionable functionalities, partially they even enable undefined behavior. Please only use any of them if you know what you're doing:
 - `HPX_SERIALIZATION_WITH_ALLOW_RAW_POINTER_SERIALIZATION`
 - `HPX_SERIALIZATION_WITH_ALL_TYPES_ARE_BITWISE_SERIALIZABLE`
 - `HPX_SERIALIZATION_WITH_ALLOW_CONST_TUPLE_MEMBERS`

Breaking changes

- Minimum required C++ standard library is C++17.
- Support for GCC 7 and Clang 8.0.0 and below has been removed.
- CUDA version required updated to 11.4.
- CMake version required updated to 3.18.
- The default version of Asio used was updated to 1.20.0.
- The default version of APEX used was updated to 2.5.1.
- `tagged_pair` and `tagged_tuple` were removed.
- `tag_dispatch` was renamed to `tag_invoke`.
- `hpx.max_backgroud_threads` was renamed to `hpx.parcel.max_background_threads`.
- The following CMake flags were removed after being deprecated for at least two releases:
 - `HPX_SCHEDULER_MAX_TERMINATED_THREADS`

- HPX_WITH_GOOGLE_PERFTOOLS
- HPX_WITH_INIT_START_OVERLOADS_COMPATIBILITY
- HPX_HAVE_{COROUTINE, PLUGIN}_GCC_HIDDEN_VISIBILITY
- HPX_TOP_LEVEL
- HPX_WITH_COMPUTE_CUDA
- HPX_WITH_ASYNC_CUDA
- `annotate_function` was renamed to `scoped_annotation`.
- `execution::transform` was renamed to `execution::then`.
- `execution::detach` was renamed to `execution::start_detached`.
- `execution::on_sender` was renamed to `execution::schedule_on`.
- `execution::just_on` was renamed to `execution::just_transfer`.
- `execution::set_done` was renamed to `execution::set_stopped`.

Closed issues

- Issue #5871⁵⁰⁷ - `distributed::channel.register_as` terminates the active task.
- Issue #5856⁵⁰⁸ - Performance counters do not compile
- Issue #5828⁵⁰⁹ - `hpx::distributed::barrier` errors
- Issue #5812⁵¹⁰ - OctoTiger does not compile with HPX master and CUDA 11.5
- Issue #5784⁵¹¹ - HPX failing with `co_await` and `hpx::when_all(futures)`
- Issue #5774⁵¹² - CMake can't find HPXCacheVariables.cmake
- Issue #5764⁵¹³ - Fix HIP problem
- Issue #5724⁵¹⁴ - Missing binary filter compression header
- Issue #5721⁵¹⁵ - Cleanup after repository split
- Issue #5701⁵¹⁶ - It seems that the tcp parcelport is running, and the MPI parcelport is ignored
- Issue #5692⁵¹⁷ - Kokkos compilation fails when using both HPX and CUDA execution spaces with gcc 9.3.0
- Issue #5686⁵¹⁸ - Rename `annotate_function`
- Issue #5668⁵¹⁹ - HPX does not detect the C++ 20 standard using gcc 11.2
- Issue #5666⁵²⁰ - Compilation error using boost 1.76 and gcc 11.2.1

⁵⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5871>

⁵⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5856>

⁵⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/5828>

⁵¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/5812>

⁵¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/5784>

⁵¹² <https://github.com/STELLAR-GROUP/hpx/issues/5774>

⁵¹³ <https://github.com/STELLAR-GROUP/hpx/issues/5764>

⁵¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5724>

⁵¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5721>

⁵¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/5701>

⁵¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5692>

⁵¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5686>

⁵¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/5668>

⁵²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/5666>

- Issue #5653⁵²¹ - Implement P2248 for our algorithms
- Issue #5647⁵²² - [User input needed] Remove (CUDA) compute functionality?
- Issue #5590⁵²³ - hello_world_distributed fails on startup with HPX stable, MPICH 3.3.2, on Deep Bayou
- Issue #5570⁵²⁴ - Rename tag_dispatch to tag_invoke
- Issue #5566⁵²⁵ - can't build simple example: "Cannot use the dummy implementation of future_then_dispatch"
- Issue #5565⁵²⁶ - build failure: hpx::string_util::trim()
- Issue #5553⁵²⁷ - Github action to validate the cff file refs #5471
- Issue #5504⁵²⁸ - CMake does not work for HPX 1.7.0 on Piz Daint
- Issue #5503⁵²⁹ - Use contiguous index queue in bulk execution to reduce number of spawned tasks
- Issue #5502⁵³⁰ - C++20 std::coroutine cmake detection
- Issue #5478⁵³¹ - hpx.dll built with vcpkg got functions pointing to the same location
- Issue #5472⁵³² - Compilation error with cuda/11.3
- Issue #5469⁵³³ - Compiler warning about HPX_NODISCARD when building with APEX
- Issue #5463⁵³⁴ - Address minor comments of the C++17 PR bump
- Issue #5456⁵³⁵ - Use *std::ranges::iter_swap* where available
- Issue #5404⁵³⁶ - Build fails with error "Cannot open include file asio/io_context.hpp"
- Issue #5381⁵³⁷ - Add starts_with and ends_with algorithms
- Issue #5344⁵³⁸ - Further simplify tag_invoke helpers
- Issue #5269⁵³⁹ - Allow setting a label on executors/policies
- Issue #5219⁵⁴⁰ - (Re-)Implement executor API on top of sender/receiver infrastructure
- Issue #5216⁵⁴¹ - Performance counter module not loading
- Issue #5162⁵⁴² - Require C++17 support
- Issue #5156⁵⁴³ - Disentangle segmented algorithms

⁵²¹ <https://github.com/STELLAR-GROUP/hpx/issues/5653>

⁵²² <https://github.com/STELLAR-GROUP/hpx/issues/5647>

⁵²³ <https://github.com/STELLAR-GROUP/hpx/issues/5590>

⁵²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5570>

⁵²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5566>

⁵²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/5565>

⁵²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5553>

⁵²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5504>

⁵²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/5503>

⁵³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/5502>

⁵³¹ <https://github.com/STELLAR-GROUP/hpx/issues/5478>

⁵³² <https://github.com/STELLAR-GROUP/hpx/issues/5472>

⁵³³ <https://github.com/STELLAR-GROUP/hpx/issues/5469>

⁵³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5463>

⁵³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5456>

⁵³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/5404>

⁵³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5381>

⁵³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5344>

⁵³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/5269>

⁵⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/5219>

⁵⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/5216>

⁵⁴² <https://github.com/STELLAR-GROUP/hpx/issues/5162>

⁵⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/5156>

- Issue #5118⁵⁴⁴ - Lock held while suspending
- Issue #5111⁵⁴⁵ - Tests fail to build with binary_filter plugins enabled
- Issue #5110⁵⁴⁶ - Tests don't get built
- Issue #5105⁵⁴⁷ - PAPI performance counters not available
- Issue #5002⁵⁴⁸ - hpx::lcos::barrier() results in deadlock
- Issue #4992⁵⁴⁹ - Clang-format the rest of the files
- Issue #4987⁵⁵⁰ - Use std::function in public APIs
- Issue #4871⁵⁵¹ - HEP: conformance to C++20
- Issue #4822⁵⁵² - Adapt parallel algorithms to C++20
- Issue #4736⁵⁵³ - Deprecate hpx::flush and hpx::endl
- Issue #4558⁵⁵⁴ - Prevent work-stealing from stalling
- Issue #4495⁵⁵⁵ - Add anchor links to table rows in documentation
- Issue #4469⁵⁵⁶ - New thread state: *pending_low*
- Issue #4321⁵⁵⁷ - After the modularization the libfabric parcelport does not compile
- Issue #4308⁵⁵⁸ - Using APEX on multinode jobs when HPX_WITH_NETWORKING = OFF
- Issue #3995⁵⁵⁹ - Use C++20 std::source_location where available, adapt ours to conform
- Issue #3861⁵⁶⁰ - Selected processor does not support 'yield' in ARM mode
- Issue #3706⁵⁶¹ - Add shift_left and shift_right algorithms
- Issue #3646⁵⁶² - Parallel algorithms should accept iterator/sentinel pairs
- Issue #3636⁵⁶³ - HPX Modularization
- Issue #3546⁵⁶⁴ - Modularization of HPX
- Issue #3474⁵⁶⁵ - Modernize CMake used in HPX
- Issue #1836⁵⁶⁶ - hpx::parallel does not have a sort implementation

⁵⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5118>

⁵⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5111>

⁵⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/5110>

⁵⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5105>

⁵⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5002>

⁵⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4992>

⁵⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4987>

⁵⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/4871>

⁵⁵² <https://github.com/STELLAR-GROUP/hpx/issues/4822>

⁵⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/4736>

554 <https://github.com/STELLAR-GROUP/hpx/issues/4558>555 <https://github.com/STELLAR-GROUP/hpx/issues/4495>556 <https://github.com/STELLAR-GROUP/hpx/issues/4469>557 <https://github.com/STELLAR-GROUP/hpx/issues/4321>558 <https://github.com/STELLAR-GROUP/hpx/issues/4308>559 <https://github.com/STELLAR-GROUP/hpx/issues/3995>560 <https://github.com/STELLAR-GROUP/hpx/issues/3861>561 <https://github.com/STELLAR-GROUP/hpx/issues/3706>562 <https://github.com/STELLAR-GROUP/hpx/issues/3646>563 <https://github.com/STELLAR-GROUP/hpx/issues/3636>564 <https://github.com/STELLAR-GROUP/hpx/issues/3546>565 <https://github.com/STELLAR-GROUP/hpx/issues/3474>566 <https://github.com/STELLAR-GROUP/hpx/issues/1836>

- Issue #1668⁵⁶⁷ - Adapt all parallel algorithms to Ranges TS
- Issue #1141⁵⁶⁸ - Implement N4409 on top of HPX

Closed pull requests

- PR #5885⁵⁶⁹ - Testing newer ASIO version
- PR #5884⁵⁷⁰ - Fix miscellaneous doc sections
- PR #5882⁵⁷¹ - Fixing OctoTiger incompatibility introduced recently
- PR #5881⁵⁷² - Fixing recent patch that disables ATOMIC_FLAG_INIT for C++20 and up
- PR #5880⁵⁷³ - refactor: convert *counter_status* enum to enum class
- PR #5878⁵⁷⁴ - Docs: Replaced non-existent create_reducer function with create_communicator
- PR #5877⁵⁷⁵ - Doc updates hpx runtime and resources
- PR #5876⁵⁷⁶ - Updates to documentation; grammar edits.
- PR #5875⁵⁷⁷ - Doc updates starting the hpx runtime
- PR #5874⁵⁷⁸ - Doc updates launching configuring
- PR #5873⁵⁷⁹ - Prevent certain generated files from being deleted on reconfigure
- PR #5870⁵⁸⁰ - Adding support for the PJM batch environment
- PR #5867⁵⁸¹ - Update CMakeLists.txt
- PR #5866⁵⁸² - add cmake option HPX_WITH_PARCELPORT_COUNTERS
- PR #5864⁵⁸³ - ATOMIC_INIT_FLAG is deprecated starting C++20
- PR #5863⁵⁸⁴ - Adding llvm 14.0.0 with boost 1.79.0 to Jenkins
- PR #5861⁵⁸⁵ - Let install step proceed on CircleCI even if the segmented algorithms fail
- PR #5860⁵⁸⁶ - Updating APEX tag
- PR #5859⁵⁸⁷ - Splitting documentation generation steps on CircleCI
- PR #5854⁵⁸⁸ - Fixing left-overs from changing counter_type to enum class

⁵⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1668>

⁵⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1141>

⁵⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5885>

⁵⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5884>

571 <https://github.com/STELLAR-GROUP/hpx/pull/5882>572 <https://github.com/STELLAR-GROUP/hpx/pull/5881>573 <https://github.com/STELLAR-GROUP/hpx/pull/5880>574 <https://github.com/STELLAR-GROUP/hpx/pull/5878>575 <https://github.com/STELLAR-GROUP/hpx/pull/5877>576 <https://github.com/STELLAR-GROUP/hpx/pull/5876>577 <https://github.com/STELLAR-GROUP/hpx/pull/5875>578 <https://github.com/STELLAR-GROUP/hpx/pull/5874>579 <https://github.com/STELLAR-GROUP/hpx/pull/5873>580 <https://github.com/STELLAR-GROUP/hpx/pull/5870>581 <https://github.com/STELLAR-GROUP/hpx/pull/5867>582 <https://github.com/STELLAR-GROUP/hpx/pull/5866>583 <https://github.com/STELLAR-GROUP/hpx/pull/5864>584 <https://github.com/STELLAR-GROUP/hpx/pull/5863>585 <https://github.com/STELLAR-GROUP/hpx/pull/5861>586 <https://github.com/STELLAR-GROUP/hpx/pull/5860>587 <https://github.com/STELLAR-GROUP/hpx/pull/5859>588 <https://github.com/STELLAR-GROUP/hpx/pull/5854>

- PR #5853⁵⁸⁹ - Adding HPX dependency tool (adapted from Boostdep tool)
- PR #5852⁵⁹⁰ - Optimize LCI parcelport
- PR #5851⁵⁹¹ - Forking dynamic_bitset from Boost
- PR #5850⁵⁹² - Convert perf_counters::counter_type enum to enum class.
- PR #5849⁵⁹³ - Update LCI parcelport to LCI v1.7.1
- PR #5848⁵⁹⁴ - Fedora related fixes
- PR #5847⁵⁹⁵ - Fix API, troubleshooting & people
- PR #5844⁵⁹⁶ - Attempting to fix timeouts of segmented iterator tests
- PR #5842⁵⁹⁷ - change the default value of HPX_WITH_LCL_TAG to v1.7
- PR #5841⁵⁹⁸ - Move the split_future facilities into the namespace hpx
- PR #5840⁵⁹⁹ - wait_xxx_nothrow functions return whether one of the futures is exceptional
- PR #5839⁶⁰⁰ - Moving a list of synchronization primitives into namespace hpx
- PR #5837⁶⁰¹ - Moving latch types to hpx and hpx::distributed namespaces
- PR #5835⁶⁰² - Add missing compatibility layer for id_type::management_type values
- PR #5834⁶⁰³ - API docs changes
- PR #5831⁶⁰⁴ - Further improvement actions to rotate
- PR #5830⁶⁰⁵ - Exposing zero-copy serialization threshold through configuration option
- PR #5829⁶⁰⁶ - Attempting to fix failing barrier test
- PR #5827⁶⁰⁷ - Add back explicit template parameter to *ignore_while_checking* to compile with nvcc
- PR #5826⁶⁰⁸ - Reduce number of allocations while calling async_bulk_execute
- PR #5825⁶⁰⁹ - Steal from neighboring NUMA domain only
- PR #5823⁶¹⁰ - Remove obsolete directories and adjust build system
- PR #5822⁶¹¹ - Clang-format remaining files

⁵⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5853>

⁵⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5852>

⁵⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5851>

⁵⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5850>

⁵⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5849>

⁵⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5848>

⁵⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5847>

⁵⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5844>

⁵⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5842>

⁵⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5841>

⁵⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5840>

⁶⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5839>

⁶⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5837>

⁶⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5835>

603 <https://github.com/STELLAR-GROUP/hpx/pull/5834>604 <https://github.com/STELLAR-GROUP/hpx/pull/5831>605 <https://github.com/STELLAR-GROUP/hpx/pull/5830>606 <https://github.com/STELLAR-GROUP/hpx/pull/5829>607 <https://github.com/STELLAR-GROUP/hpx/pull/5827>608 <https://github.com/STELLAR-GROUP/hpx/pull/5826>609 <https://github.com/STELLAR-GROUP/hpx/pull/5825>610 <https://github.com/STELLAR-GROUP/hpx/pull/5823>611 <https://github.com/STELLAR-GROUP/hpx/pull/5822>

- PR #5821⁶¹² - Enable permissive- flag on Windows GitHub actions builders
- PR #5820⁶¹³ - Convert throwmode enum to enum class
- PR #5819⁶¹⁴ - Marking customization points for intrusive_ptr as noexcept
- PR #5818⁶¹⁵ - Unconditionally use C++17 attributes
- PR #5817⁶¹⁶ - Modernize naming modules
- PR #5816⁶¹⁷ - Modernize cache module
- PR #5815⁶¹⁸ - Reapply flyby changes from #5467
- PR #5814⁶¹⁹ - Avoid test timeouts by reducing test sizes
- PR #5813⁶²⁰ - The CUDA problem is not fixed in V11.5 yet...
- PR #5811⁶²¹ - Make sure reduction value is properly moved, when possible
- PR #5810⁶²² - Improve error reporting during device initialization in HIP environments
- PR #5809⁶²³ - Converting scheduler enums into enum class
- PR #5808⁶²⁴ - Deprecate hpx::flush and friends
- PR #5807⁶²⁵ - Use C++20 std::source_location, if available
- PR #5806⁶²⁶ - Moving promise and packaged_task to new namespaces
- PR #5805⁶²⁷ - Attempting to fix a test failure when using the LCI parclpor
- PR #5803⁶²⁸ - Attempt to fix CUDA related OctoTiger problems
- PR #5800⁶²⁹ - Add option to restrict MPI background work to subset of cores
- PR #5798⁶³⁰ - Adding MPI as a dependency to APEX
- PR #5797⁶³¹ - Extend Sphinx role to support arbitrary text to display on a link
- PR #5796⁶³² - Disable CUDA tests that cause NVCC to silently fail without error messages
- PR #5795⁶³³ - Avoid writing path and directories into HPXCacheVariables.cmake
- PR #5793⁶³⁴ - Remove features that are deprecated since V1.6

⁶¹² <https://github.com/STELLAR-GROUP/hpx/pull/5821>

⁶¹³ <https://github.com/STELLAR-GROUP/hpx/pull/5820>

⁶¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5819>

⁶¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5818>

⁶¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5817>

⁶¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5816>

⁶¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5815>

⁶¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5814>

⁶²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5813>

⁶²¹ <https://github.com/STELLAR-GROUP/hpx/pull/5811>

⁶²² <https://github.com/STELLAR-GROUP/hpx/pull/5810>

⁶²³ <https://github.com/STELLAR-GROUP/hpx/pull/5809>

⁶²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5808>

⁶²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5807>

⁶²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5806>

⁶²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5805>

⁶²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5803>

⁶²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5800>

⁶³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5798>

⁶³¹ <https://github.com/STELLAR-GROUP/hpx/pull/5797>

⁶³² <https://github.com/STELLAR-GROUP/hpx/pull/5796>

⁶³³ <https://github.com/STELLAR-GROUP/hpx/pull/5795>

⁶³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5793>

- PR #5792⁶³⁵ - Making sure num_cores is properly handled by parallel_executor
- PR #5791⁶³⁶ - Moving bind, bind_front, bind_back to namespace hpx
- PR #5790⁶³⁷ - Moving serializable function/move_only_function into namespace hpx::distributed
- PR #5787⁶³⁸ - Remove unneeded (and commented) tests
- PR #5786⁶³⁹ - Attempting to fix hangs in distributed barrier
- PR #5785⁶⁴⁰ - add cmake code to detect arm64 on macOS
- PR #5783⁶⁴¹ - Moving function and function_ref into namespace hpx
- PR #5781⁶⁴² - Updating used version of Visual Studio
- PR #5780⁶⁴³ - Update Piz Daint Jenkins configurations from gcc/clang 7 to 8
- PR #5778⁶⁴⁴ - Updated for_loop.hpp
- PR #5777⁶⁴⁵ - Update reference for foreach benchmark
- PR #5775⁶⁴⁶ - Move optional into namespace hpx
- PR #5773⁶⁴⁷ - Moving barrier to consolidated namespaces
- PR #5772⁶⁴⁸ - Adding missing docs for ranges::find_if and find_if_not algorithms
- PR #5771⁶⁴⁹ - Moving for_loop into namespace hpx::experimental
- PR #5770⁶⁵⁰ - Fixing HIP issues
- PR #5769⁶⁵¹ - Slight improvement of small_vector performance
- PR #5766⁶⁵² - Fixing a integral conversion warning
- PR #5765⁶⁵³ - Adding a sphinx role allowing to link to a file directly in github
- PR #5763⁶⁵⁴ - add num_cores facility
- PR #5762⁶⁵⁵ - Fix Public API main page
- PR #5761⁶⁵⁶ - Add missing inline to mpi_exception.hpp error_message function
- PR #5760⁶⁵⁷ - Update cdash build url

⁶³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5792>

⁶³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5791>

⁶³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5790>

⁶³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5787>

⁶³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5786>

⁶⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5785>

⁶⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/5783>

⁶⁴² <https://github.com/STELLAR-GROUP/hpx/pull/5781>

⁶⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/5780>

⁶⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5778>

⁶⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5777>

⁶⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5775>

⁶⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5773>

⁶⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5772>

⁶⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5771>

⁶⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5770>

⁶⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/5769>

⁶⁵² <https://github.com/STELLAR-GROUP/hpx/pull/5766>

⁶⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5765>

⁶⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5763>

⁶⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5762>

⁶⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5761>

⁶⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5760>

- PR #5759⁶⁵⁸ - Switch to use generic rostam SLURM partitions
- PR #5758⁶⁵⁹ - Adding support for P2300 completion signatures
- PR #5757⁶⁶⁰ - Fix missing links in Public API
- PR #5756⁶⁶¹ - Add stop support to when_all
- PR #5755⁶⁶² - Support for data-parallelism for mismatch algorithm
- PR #5754⁶⁶³ - Support for data-parallelism for equal algorithm
- PR #5751⁶⁶⁴ - Propagate MPI dependencies to command line handling
- PR #5750⁶⁶⁵ - Make sure required MPI initialization flags are properly applied and supported
- PR #5749⁶⁶⁶ - P2300 stop token
- PR #5748⁶⁶⁷ - Adding environmental query CPOs
- PR #5747⁶⁶⁸ - Renaming set_done to set_stopped (as per P2300)
- PR #5745⁶⁶⁹ - Modernize serialization module
- PR #5743⁶⁷⁰ - Add check for MPICH and set the correct env to support multi-threaded
- PR #5742⁶⁷¹ - Remove obsolete files related to cpuid, etc.
- PR #5741⁶⁷² - Support for data-parallelism for adjacent find
- PR #5740⁶⁷³ - Support for data-parallelism for find algorithms
- PR #5739⁶⁷⁴ - Enable the option to attach a debugger on a segmentation fault (linux)
- PR #5738⁶⁷⁵ - Fixing spell-checking errors
- PR #5737⁶⁷⁶ - Attempt to fix migrate_component issue
- PR #5736⁶⁷⁷ - Set commit status from Jenkins also for special branches
- PR #5734⁶⁷⁸ - Revert #5586
- PR #5732⁶⁷⁹ - Attempt to improve build-id reporting to cdash
- PR #5731⁶⁸⁰ - Randomly delay execution of bash scripts launched by Jenkins

⁶⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5759>

⁶⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5758>

⁶⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5757>

⁶⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/5756>

⁶⁶² <https://github.com/STELLAR-GROUP/hpx/pull/5755>

⁶⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/5754>

⁶⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5751>

⁶⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5750>

⁶⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5749>

⁶⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5748>

⁶⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5747>

⁶⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5745>

⁶⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5743>

⁶⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/5742>

⁶⁷² <https://github.com/STELLAR-GROUP/hpx/pull/5741>

⁶⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/5740>

⁶⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5739>

⁶⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5738>

⁶⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5737>

⁶⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5736>

⁶⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5734>

⁶⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5732>

⁶⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5731>

- PR #5729⁶⁸¹ - Workaround for CMake/Ninja generator OOM problem
- PR #5727⁶⁸² - Moving compression plugins to components directory
- PR #5726⁶⁸³ - Moving/consolidating parcel coalescing plugin sources
- PR #5725⁶⁸⁴ - Making sure headers for serialization filters are being installed
- PR #5723⁶⁸⁵ - Moving more tests to modules
- PR #5722⁶⁸⁶ - Removing superfluous semicolons
- PR #5720⁶⁸⁷ - Moving parcelports into modules
- PR #5719⁶⁸⁸ - Moving more files to parcelset module
- PR #5718⁶⁸⁹ - build: refactor sphinx config file
- PR #5717⁶⁹⁰ - Creating parcelset modules
- PR #5716⁶⁹¹ - Avoid duplicate definition error
- PR #5715⁶⁹² - The new LCI parcelport for HPX
- PR #5714⁶⁹³ - Refine propagation of **HPX_WITH_**... options
- PR #5713⁶⁹⁴ - Significantly reduce CI jobs run on Piz Daint
- PR #5712⁶⁹⁵ - Updating jenkins configuration for Rostam2.2
- PR #5711⁶⁹⁶ - Refactor manual sections
- PR #5710⁶⁹⁷ - Making task_group serializable
- PR #5709⁶⁹⁸ - Update the MPI cmake setup
- PR #5707⁶⁹⁹ - Better diagnose parcel bootstrap problems
- PR #5704⁷⁰⁰ - Test with hwloc 2.7.0 with GCC 11
- PR #5703⁷⁰¹ - Fix *counting_iterator* container tests
- PR #5702⁷⁰² - Attempting to fix CircleCI timeouts
- PR #5699⁷⁰³ - Update CI to use Boost 1.78.0

⁶⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/5729>

⁶⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5727>

⁶⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5726>

⁶⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5725>

⁶⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5723>

⁶⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5722>

⁶⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5720>

⁶⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5719>

⁶⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5718>

⁶⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5717>

⁶⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5716>

⁶⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5715>

⁶⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5714>

⁶⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5713>

⁶⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5712>

⁶⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5711>

⁶⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5710>

⁶⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5709>

⁶⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5707>

⁷⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5704>

⁷⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5703>

⁷⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5702>

⁷⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/5699>

- PR #5697⁷⁰⁴ - Adding fork_join_executor to foreach_benchmark
- PR #5696⁷⁰⁵ - Modernize when_all and friends (when_any, when_some, when_each)
- PR #5693⁷⁰⁶ - Fix test errors with _GLIBCXX_DEBUG defined
- PR #5691⁷⁰⁷ - Rename *annotate_function* to *scoped_annotation*
- PR #5690⁷⁰⁸ - Replace tag_dispatch with tag_invoke in minmax segmented
- PR #5688⁷⁰⁹ - Remove more deprecated macros
- PR #5687⁷¹⁰ - Add most important CMake options
- PR #5685⁷¹¹ - Fix future API
- PR #5684⁷¹² - Move lock registration to separate module and remove global lock registration
- PR #5683⁷¹³ - Make hpx::wait_all etc. throw exceptions when waited futures hold exceptions and deprecate hpx::lcos::wait_all[_n] in favor of hpx::wait_all[_n]
- PR #5682⁷¹⁴ - Fix macOS test exceptions
- PR #5681⁷¹⁵ - docs: add links to hpx recepies
- PR #5680⁷¹⁶ - Embed base execution policies to datapar execution policies
- PR #5679⁷¹⁷ - Fix *fork_join_executor* with dynamic schedule
- PR #5678⁷¹⁸ - Fix compilation of service executors with nvcc
- PR #5677⁷¹⁹ - Remove compute_cuda module
- PR #5676⁷²⁰ - Don't require up-to-date approvals for bors
- PR #5675⁷²¹ - Add default template type parameters for algorithms
- PR #5674⁷²² - Allow using *any_sender* in global variables
- PR #5671⁷²³ - Making sure task_group can be reused
- PR #5670⁷²⁴ - Relax constraints on *execution::when_all*
- PR #5669⁷²⁵ - Use HPX_WITH_CXX_STANDARD for controlling C++ version
- PR #5667⁷²⁶ - Attempt to fix compilation issues with Boost V1.76

⁷⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5697>

⁷⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5696>

⁷⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5693>

⁷⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5691>

⁷⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5690>

⁷⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5688>

⁷¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5687>

⁷¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5685>

⁷¹² <https://github.com/STELLAR-GROUP/hpx/pull/5684>

⁷¹³ <https://github.com/STELLAR-GROUP/hpx/pull/5683>

⁷¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5682>

⁷¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5681>

⁷¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5680>

⁷¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5679>

⁷¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5678>

⁷¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5677>

⁷²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5676>

⁷²¹ <https://github.com/STELLAR-GROUP/hpx/pull/5675>

⁷²² <https://github.com/STELLAR-GROUP/hpx/pull/5674>

⁷²³ <https://github.com/STELLAR-GROUP/hpx/pull/5671>

⁷²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5670>

⁷²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5669>

⁷²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5667>

- PR #5664⁷²⁷ - Change logging errors to warnings in schedulers
- PR #5663⁷²⁸ - Use dynamic bitsets by default for CPU masks
- PR #5662⁷²⁹ - Disambiguate namespace for MSVC
- PR #5660⁷³⁰ - Replacing remaining std::forward and std::move with HPX_FORWARD and HPX_MOVE
- PR #5659⁷³¹ - Modernize hpx::future and related facilities
- PR #5658⁷³² - Replace HPX_INLINE_CONSTEXPR_VARIABLE with inline constexpr
- PR #5657⁷³³ - Remove tagged, tagged_pair and tagged_tuple, remove tuple/pair specializations
- PR #5656⁷³⁴ - Rename on execution::schedule_from, rename just_on to just_transfer, and add transfer
- PR #5655⁷³⁵ - Avoid for module lists to grow indefinitely in cmake cache
- PR #5649⁷³⁶ - build: replace usage of Python's reserved words and functions as variable names
- PR #5648⁷³⁷ - Modernize action modules and related code
- PR #5646⁷³⁸ - Fix ends_with test
- PR #5645⁷³⁹ - Add matrix multiplication example
- PR #5644⁷⁴⁰ - Rename execution::transform to execution::then and execution::detach to execution::start_detached
- PR #5643⁷⁴¹ - Update performance test references
- PR #5642⁷⁴² - Adapting adjacent_difference to work with proxy iterators
- PR #5641⁷⁴³ - Factorize perftests scripts
- PR #5640⁷⁴⁴ - Fixed links to sources in Sphinx documentation
- PR #5639⁷⁴⁵ - Fix generate datapar tests for Vc
- PR #5638⁷⁴⁶ - Simd all any none
- PR #5637⁷⁴⁷ - Use bors for merging pull requests
- PR #5636⁷⁴⁸ - Fix leftover std::holds_alternative usage
- PR #5635⁷⁴⁹ - Update container image tag in GitHub actions HIP configuration

⁷²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5664>

⁷²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5663>

⁷²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5662>

⁷³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5660>

⁷³¹ <https://github.com/STELLAR-GROUP/hpx/pull/5659>

⁷³² <https://github.com/STELLAR-GROUP/hpx/pull/5658>

⁷³³ <https://github.com/STELLAR-GROUP/hpx/pull/5657>

⁷³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5656>

⁷³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5655>

⁷³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5649>

⁷³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5648>

⁷³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5646>

⁷³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5645>

⁷⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5644>

⁷⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/5643>

⁷⁴² <https://github.com/STELLAR-GROUP/hpx/pull/5642>

⁷⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/5641>

⁷⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5640>

⁷⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5639>

⁷⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5638>

⁷⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5637>

⁷⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5636>

⁷⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5635>

- PR #5633⁷⁵⁰ - Moving packaged_task to module futures
- PR #5632⁷⁵¹ - Tell Asio to use std::aligned_new only if available
- PR #5631⁷⁵² - Adding tag parameter to channel communicator get/set
- PR #5630⁷⁵³ - Add partial_sort_copy and adapt partial sort to c++ 20
- PR #5629⁷⁵⁴ - Set HPX_WITH_FETCH_ASIO to OFF as available in the docker image
- PR #5628⁷⁵⁵ - Add Clang 13 CI configuration
- PR #5627⁷⁵⁶ - Replace alternative keyword
- PR #5626⁷⁵⁷ - docs: add support for BibTeX references in Sphinx docs
- PR #5624⁷⁵⁸ - Fix pkgconfig replacements involving CMAKE_INSTALL_PREFIX
- PR #5623⁷⁵⁹ - build: remove unused import from conf.py.in
- PR #5622⁷⁶⁰ - Remove HPX_WITH_VCPKG CMake option
- PR #5621⁷⁶¹ - Replacing boost::container::small_vector
- PR #5620⁷⁶² - Update Asio tag from 1.18.2 to 1.20.0
- PR #5619⁷⁶³ - Fix block_os_threads_1036 test
- PR #5618⁷⁶⁴ - Make sure condition variables are notified under a lock in the thread_pool_scheduler test
- PR #5617⁷⁶⁵ - Use advance_and_get_distance where required
- PR #5616⁷⁶⁶ - Remove separately building segmented algorithms on CircleCI
- PR #5613⁷⁶⁷ - Fix Vc datapar adjacent_difference
- PR #5609⁷⁶⁸ - docs: add anchor links to performance counter tables
- PR #5608⁷⁶⁹ - Fix header test error by adding missing numeric
- PR #5607⁷⁷⁰ - Fix simd adj diff
- PR #5605⁷⁷¹ - Fix usage of HPX_INVOKE macro
- PR #5604⁷⁷² - Make use of shell-session to allow non-copyable \$

⁷⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5633>

⁷⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/5632>

⁷⁵² <https://github.com/STELLAR-GROUP/hpx/pull/5631>

⁷⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5630>

⁷⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5629>

⁷⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5628>

⁷⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5627>

⁷⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5626>

⁷⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5624>

⁷⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5623>

⁷⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5622>

⁷⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/5621>

⁷⁶² <https://github.com/STELLAR-GROUP/hpx/pull/5620>

⁷⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/5619>

⁷⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5618>

⁷⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5617>

⁷⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5616>

⁷⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5613>

⁷⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5609>

⁷⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5608>

⁷⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5607>

⁷⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/5605>

⁷⁷² <https://github.com/STELLAR-GROUP/hpx/pull/5604>

- PR #5603⁷⁷³ - Suppress some MSVC warnings in C++20 mode
- PR #5602⁷⁷⁴ - Test HPX_DATASTRUCTURES_WITH_ADAPT_STD_TUPLE=OFF to one CI configuration
- PR #5601⁷⁷⁵ - Test case for any_sender should use hpx::tuple
- PR #5600⁷⁷⁶ - Rename tag_dispatch back to tag_invoke
- PR #5599⁷⁷⁷ - Change theme, fix Quickstart & Examples
- PR #5596⁷⁷⁸ - Use precompiled headers in tests
- PR #5595⁷⁷⁹ - Drop semicolons for macro calls
- PR #5594⁷⁸⁰ - Adapt datapar generate
- PR #5593⁷⁸¹ - Update any_sender to use tag_dispatch for execution customizations
- PR #5592⁷⁸² - Add nth_element
- PR #5591⁷⁸³ - Remove unnecessary checks for C++17 for tests
- PR #5589⁷⁸⁴ - Add HPX_FORWARD/HPX_MOVE macros
- PR #5588⁷⁸⁵ - Fixing the output formatting for id_types
- PR #5586⁷⁸⁶ - Remove local functionality
- PR #5585⁷⁸⁷ - Delete GitExternal.cmake
- PR #5584⁷⁸⁸ - Serialization of hpx::tuple must use hpx::get
- PR #5583⁷⁸⁹ - fix coroutine_traits allocate calls, add unhandled_exception() implementation.
- PR #5582⁷⁹⁰ - Make more examples work with local runtime
- PR #5581⁷⁹¹ - Add support for several performance tests in CI
- PR #5580⁷⁹² - Adapt simd adj diff
- PR #5579⁷⁹³ - Split absolute paths for generated pkg-config files into -L/-l parts
- PR #5577⁷⁹⁴ - fix unit fill test for datapar with Vc
- PR #5576⁷⁹⁵ - Update forgotten “Full” names

⁷⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/5603>

⁷⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5602>

⁷⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5601>

⁷⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5600>

⁷⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5599>

⁷⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5596>

⁷⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5595>

⁷⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5594>

⁷⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/5593>

⁷⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5592>

⁷⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5591>

⁷⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5589>

⁷⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5588>

⁷⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5586>

⁷⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5585>

⁷⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5584>

⁷⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5583>

⁷⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5582>

⁷⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5581>

⁷⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5580>

⁷⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5579>

⁷⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5577>

⁷⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5576>

- PR #5575⁷⁹⁶ - Change scan partitioner implementation
- PR #5574⁷⁹⁷ - Remove a few deprecated and unused CMake options
- PR #5572⁷⁹⁸ - Remove more guards for the distributed runtime
- PR #5571⁷⁹⁹ - Add workaround for libstdc++ in string_util trim
- PR #5569⁸⁰⁰ - Use no_unique_address in sender adaptors
- PR #5568⁸⁰¹ - Change try catch block to try_catch_exception_ptr
- PR #5567⁸⁰² - Make default_agent::yield actually yield
- PR #5564⁸⁰³ - Adjacent
- PR #5562⁸⁰⁴ - More changes to overcome build problems on Windows after recent module rearrangements
- PR #5560⁸⁰⁵ - Update tests and examples
- PR #5559⁸⁰⁶ - Fixing cmake folder names after module restructuring
- PR #5558⁸⁰⁷ - Fixing wrong module dependencies
- PR #5557⁸⁰⁸ - Adding an example for the new channel_communicator API
- PR #5556⁸⁰⁹ - Remove leftover thread pool os executor tests
- PR #5555⁸¹⁰ - Add option enabling serializing raw pointers
- PR #5554⁸¹¹ - Make sure command line aliasing is properly handled
- PR #5552⁸¹² - Modernizing some of the async facilities
- PR #5551⁸¹³ - Fixing for local executions of actions to properly set task names
- PR #5550⁸¹⁴ - Update CUDA module in clang-cuda configuration
- PR #5549⁸¹⁵ - Fixing agent_ref::yield_k to actually call yield_k
- PR #5548⁸¹⁶ - Making get_action_name() noexcept
- PR #5547⁸¹⁷ - Fixing communication set
- PR #5546⁸¹⁸ - Fixing shutdown problems caused by missing ref-counting

⁷⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5575>

⁷⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5574>

⁷⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5572>

⁷⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5571>

⁸⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5569>

⁸⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5568>

⁸⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5567>

⁸⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/5564>

⁸⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5562>

⁸⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5560>

⁸⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5559>

⁸⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5558>

⁸⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5557>

⁸⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5556>

⁸¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5555>

⁸¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5554>

⁸¹² <https://github.com/STELLAR-GROUP/hpx/pull/5552>

⁸¹³ <https://github.com/STELLAR-GROUP/hpx/pull/5551>

⁸¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5550>

⁸¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5549>

⁸¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5548>

⁸¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5547>

⁸¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5546>

- PR #5545⁸¹⁹ - Remove wrong move in thread_pool_scheduler_bulk.hpp
- PR #5543⁸²⁰ - Extend launch policy to carry stack size and scheduling hint in addition to priority
- PR #5542⁸²¹ - Simplify execution CPOs
- PR #5540⁸²² - Adapt partition, partition_copy and stable_partition to C++ 20
- PR #5539⁸²³ - Adapt mismatch to support sentinels
- PR #5538⁸²⁴ - Document specific sphinx version required for the documentation
- PR #5537⁸²⁵ - Test release and debug builds on Piz Daint
- PR #5536⁸²⁶ - This fixes referencing stale iterators during the execution of binary mismatch
- PR #5535⁸²⁷ - Rename simdpar to par_simd
- PR #5534⁸²⁸ - Fix Quick start & Manual Docs
- PR #5533⁸²⁹ - Fix *annotate_function* for `std::string`
- PR #5532⁸³⁰ - Update two remaining apex links from khuck to UO-OACISS
- PR #5531⁸³¹ - Use contiguous_index_queue in thread_pool_scheduler
- PR #5530⁸³² - Eagerly initialize a configurable number of threads on scheduler/thread queue init
- PR #5529⁸³³ - Update benchmarks and add support for scheduler_executor
- PR #5528⁸³⁴ - Add missing properties to executors/schedulers
- PR #5527⁸³⁵ - Set local thread/pool number in local/static_queue_scheduler
- PR #5526⁸³⁶ - Update Rostam HIP configuration to use 4.3.0
- PR #5525⁸³⁷ - Fix Building HPX in Quick start
- PR #5524⁸³⁸ - Upload image on cdash
- PR #5523⁸³⁹ - Modernize facilities related to hpx::sync
- PR #5522⁸⁴⁰ - Add sender overloads for remaining algorithms
- PR #5521⁸⁴¹ - Minor changes that improve performance

⁸¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5545>

⁸²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5543>

⁸²¹ <https://github.com/STELLAR-GROUP/hpx/pull/5542>

⁸²² <https://github.com/STELLAR-GROUP/hpx/pull/5540>

⁸²³ <https://github.com/STELLAR-GROUP/hpx/pull/5539>

⁸²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5538>

⁸²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5537>

⁸²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5536>

⁸²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5535>

⁸²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5534>

⁸²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5533>

⁸³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5532>

⁸³¹ <https://github.com/STELLAR-GROUP/hpx/pull/5531>

⁸³² <https://github.com/STELLAR-GROUP/hpx/pull/5530>

⁸³³ <https://github.com/STELLAR-GROUP/hpx/pull/5529>

⁸³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5528>

⁸³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5527>

⁸³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5526>

⁸³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5525>

⁸³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5524>

⁸³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5523>

⁸⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5522>

⁸⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/5521>

- PR #5520⁸⁴² - Update reference as perftests failing regularly
- PR #5519⁸⁴³ - Add transform_mpi sender adapter
- PR #5518⁸⁴⁴ - Add sender overloads to rotate/rotate_copy
- PR #5517⁸⁴⁵ - Fix coroutine integration
- PR #5515⁸⁴⁶ - Avoid deadlock in ignore_while_locked_1485 test
- PR #5514⁸⁴⁷ - Add split sender adapter
- PR #5512⁸⁴⁸ - Update Rostam HIP configuration
- PR #5511⁸⁴⁹ - Fix Asio target name for precompiled headers
- PR #5510⁸⁵⁰ - Add any_sender and unique_any_sender
- PR #5509⁸⁵¹ - Test with Boost 1.77 on gcc/clang-newest configurations
- PR #5508⁸⁵² - Minor release changes from 1.7.1
- PR #5507⁸⁵³ - Add missing commits from scheduler_executor PR
- PR #5506⁸⁵⁴ - Fix condition for checking if we should use our own variant
- PR #5501⁸⁵⁵ - Attempt to fix thread_pool_scheduler test
- PR #5493⁸⁵⁶ - Update Jenkins GitHub token to use StellarBot GitHub account
- PR #5490⁸⁵⁷ - Fix clang-format error on master
- PR #5487⁸⁵⁸ - Add get_completion_scheduler CPO and customize bulk for thread_pool_scheduler
- PR #5484⁸⁵⁹ - Add missing header to jacobi_component/server/solver.hpp
- PR #5481⁸⁶⁰ - Changing the APEX repository to the new location
- PR #5479⁸⁶¹ - Fix version check for CUDA noexcept/result_of bug
- PR #5477⁸⁶² - Require cxx17 minor comments
- PR #5476⁸⁶³ - Fix cmake format error
- PR #5475⁸⁶⁴ - Require CMake 3.18 as it is already a requirement for CUDA

⁸⁴² <https://github.com/STELLAR-GROUP/hpx/pull/5520>

⁸⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/5519>

⁸⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5518>

⁸⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5517>

⁸⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5515>

⁸⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5514>

⁸⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5512>

⁸⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5511>

⁸⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5510>

⁸⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/5509>

⁸⁵² <https://github.com/STELLAR-GROUP/hpx/pull/5508>

⁸⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5507>

⁸⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5506>

⁸⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5501>

⁸⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5493>

⁸⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5490>

⁸⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5487>

⁸⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5484>

⁸⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5481>

⁸⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/5479>

⁸⁶² <https://github.com/STELLAR-GROUP/hpx/pull/5477>

⁸⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/5476>

⁸⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5475>

- PR #5474⁸⁶⁵ - Make the cuda parameters of try_compile optional
- PR #5473⁸⁶⁶ - Update cuda arch and change cuda version
- PR #5471⁸⁶⁷ - Add corrected citation.cff
- PR #5470⁸⁶⁸ - Adapt stable_sort to C++ 20
- PR #5468⁸⁶⁹ - Experimentation to make the perftest report public
- PR #5466⁸⁷⁰ - Add shift_left and shift_right algorithms
- PR #5465⁸⁷¹ - Adapt datapar fill
- PR #5464⁸⁷² - Moving tag_dispatch to separate module
- PR #5461⁸⁷³ - Rename HPX_WITH_CUDA_COMPUTE with HPX_WITH_COMPUTE_CUDA
- PR #5460⁸⁷⁴ - Adapt sort to C++ 20
- PR #5459⁸⁷⁵ - Adapt rotate/rotate_copy to C++20
- PR #5458⁸⁷⁶ - Adapt unique and unique_copy to C++ 20
- PR #5455⁸⁷⁷ - Remove and clean up fallback sender implementations
- PR #5454⁸⁷⁸ - Make performance plot show even if similar performance
- PR #5453⁸⁷⁹ - Post 1.7.0 version bump
- PR #5452⁸⁸⁰ - Fix find_end parallel overload
- PR #5450⁸⁸¹ - Change the print-bind output to be more precise
- PR #5449⁸⁸² - Adapt swap_ranges to C++ 20
- PR #5446⁸⁸³ - Use more verbose names in sender algorithms
- PR #5443⁸⁸⁴ - Properly support ASAN with MSVC
- PR #5441⁸⁸⁵ - Adding reference counting to thread_data
- PR #5429⁸⁸⁶ - Scheduler executor
- PR #5428⁸⁸⁷ - Adapt datapar copy

⁸⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5474>

⁸⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5473>

⁸⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5471>

⁸⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5470>

⁸⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5468>

⁸⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5466>

⁸⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/5465>

⁸⁷² <https://github.com/STELLAR-GROUP/hpx/pull/5464>

⁸⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/5461>

⁸⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5460>

⁸⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5459>

⁸⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5458>

⁸⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5455>

⁸⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5454>

⁸⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5453>

⁸⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5452>

⁸⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/5450>

⁸⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5449>

⁸⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5446>

⁸⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5443>

⁸⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5441>

⁸⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5429>

⁸⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5428>

- PR #5421⁸⁸⁸ - Update CI base image to use clang-format 11
- PR #5410⁸⁸⁹ - Add ranges starts_with and ends_with algorithms
- PR #5383⁸⁹⁰ - Tentatively remove runtime_registration_wrapper from cuda futures
- PR #5377⁸⁹¹ - Fewer Asio includes and more precompiled headers
- PR #5329⁸⁹² - Sender overloads for parallel algorithms
- PR #5313⁸⁹³ - Rearrange modules between libraries
- PR #5283⁸⁹⁴ - Require minimum C++17 and change CUDA handling
- PR #5241⁸⁹⁵ - Adapt min_element, max_element and minmax_element to C++20

2.10.2 HPX V1.7.1 (Aug 12, 2021)

This is a bugfix release with a few minor fixes.

General changes

- Added a CMake option to assume that all types are bitwise serializable by default: `HPX_SERIALIZATION_WITH_ALL_TYPES_ARE_BITWISE_SERIALIZABLE`. The default value `OFF` corresponds to the old behaviour.
- Added a version check for Asio. The minimum Asio version supported by *HPX* is 1.12.0.
- Fixed a bug affecting usage of actions, where the internals of *HPX* relied on function addresses being unique. This was fixed by relying on variable addresses being unique instead.
- Made `hpx::util::bind` more strict in checking the validity of placeholders.
- Small performance improvement to spinlocks.
- Adapted the following parallel algorithms to C++20: `inclusive_scan`, `exclusive_scan`, `transform_inclusive_scan`, `transform_exclusive_scan`.

Breaking changes

- The experimental `hpx::execution::simdpar` execution policy (introduced in 1.7.0) was renamed to `hpx::execution::par_simd` for consistency with the other parallel policies.

⁸⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5421>

⁸⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5410>

⁸⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5383>

⁸⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5377>

⁸⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5329>

⁸⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5313>

⁸⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5283>

⁸⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5241>

Closed issues

- Issue #5494⁸⁹⁶ - Rename *simdpar* execution policy to *par_simd*
- Issue #5488⁸⁹⁷ - *hpx::util::bind* doesn't bounds-check placeholders
- Issue #5486⁸⁹⁸ - Possible V1.7.1 release

Closed pull requests

- PR #5500⁸⁹⁹ - Minor bug fix in transform exclusive and inclusive scan tests
- PR #5499⁹⁰⁰ - Rename *simdpar* to *par_simd*
- PR #5489⁹⁰¹ - Adding bound-checking for bind placeholders
- PR #5485⁹⁰² - Add Asio version check
- PR #5482⁹⁰³ - Change extra archive data to rely on uniqueness of a variable address, not a function address
- PR #5448⁹⁰⁴ - More fixes to enable for all types to be assumed to be bitwise copyable
- PR #5445⁹⁰⁵ - Improve performance of Spinlocks
- PR #5444⁹⁰⁶ - Adapt *transform_inclusive_scan* to C++ 20
- PR #5440⁹⁰⁷ - Adapt *transform_exclusive_scan* to C++ 20
- PR #5439⁹⁰⁸ - Adapt *inclusive_scan* to C++ 20
- PR #5436⁹⁰⁹ - Adapt *exclusive_scan* to C++20

2.10.3 HPX V1.7.0 (Jul 14, 2021)

This release is again focused on C++20 conformance of algorithms. Additionally, many new experimental sender-based algorithms have been added based on the latest proposals.

⁸⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/5494>

⁸⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5488>

⁸⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5486>

⁸⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5500>

⁹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5499>

⁹⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5489>

⁹⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5485>

⁹⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/5482>

⁹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5448>

⁹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5445>

⁹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5444>

⁹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5440>

⁹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5439>

⁹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5436>

General changes

- The following algorithms have been adapted to be C++20 conformant:
 - `remove`,
 - `remove_if`,
 - `remove_copy`,
 - `remove_copy_if`,
 - `replace`,
 - `replace_if`,
 - `reverse`, and
 - `lexicographical_compare`.
- When the compiler and standard library support the standard execution policies `std::execution::seq`, `std::execution::par`, and `std::execution::par_unseq` they can now be used in all *HPX* parallel algorithms with equivalent behaviour to the non-task policies `hpx::execution::seq`, `hpx::execution::par`, and `hpx::execution::par_unseq`.
- Vc support has been fixed, after being broken in 1.6.0. In addition, *HPX* now experimentally supports GCC’s SIMD implementation, when available. The implementation can be used through the `hpx::execution::simd` and `hpx::execution::simdpar` execution policies.
- The customization points `sync_execute`, `async_execute`, `then_execute`, `post`, `bulk_sync_execute`, `bulk_async_execute`, and `bulk_then_execute` are now implemented using `tag_dispatch` (previously `tag_invoke`). Executors can still be implemented by providing the aforementioned functions as member functions of an executor.
- New functionality, enhancements, and fixes based on P0443r14 (executors proposal) and P1897 (sender-based algorithms) have been added to the `hpx::execution::experimental` namespace. These can be accessed through the `hpx/execution.hpp` and `hpx/local/execution.hpp` headers. In particular, the following sender-based algorithms have been added:
 - `detach`,
 - `ensure_started`,
 - `just`,
 - `just_on`,
 - `let_error`,
 - `let_value`,
 - `on`,
 - `transform`, and
 - `when_all`.

Additionally, futures now implement the sender concept. `make_future` can be used to turn a sender into a future. All functionality is experimental and can change without notice.

- All `hpx::init` and `hpx::start` overloads now take `std::functions` instead of `hpx::util::function_nonsr`. No changes should be required in user code to accommodate this change.

- `hpx::util::unwrapping` and other related unwrapping functionality has been moved up into the `hpx` namespace. Names in `hpx::util` are still usable with a deprecation warning. This functionality can now be accessed through the `hpx/unwrap.hpp` and `hpx/local/unwrap.hpp` headers.
- The default tag for APEX has been update from 2.3.1 to 2.4.0. In particular, this fixes a bug which could lead to hangs in distributed runs.
- The dependency on Boost.Asio has been replaced with the standalone Asio available at <https://github.com/chriskohlhoff/asio>. By default, a system-installed Asio will be used. `ASIO_ROOT` can be given as a hint to tell CMake where to find Asio. Alternatively, Asio can be fetched automatically using CMake's `fetchcontent` by setting `HPX_WITH_FETCH_ASIO=ON`. In general, dependencies on Boost have again been reduced.
- Modularization of the library has continued. In this release almost all functionality has been moved into modules. These changes do not generally affect user code. Warnings are still issued for headers that have moved.
- `hipBLAS` is now optional when compiling with `hipcc`. A warning instead of an error will be printed if `hipBLAS` is not found during configuration.
- Previously `HPX_COMPUTE_HOST_CODE` was defined in host code only if HPX was configured with CUDA or HIP. In this release `HPX_COMPUTE_HOST_CODE` is always defined in host code.
- An experimental `HPX_WITH_PRECOMPILED_HEADERS` CMake option has been added to use precompiled headers when building HPX. This option should not be used on Windows.
- Numerous bug fixes.

Breaking changes

- The minimum required CMake version is now 3.17.
- The minimum required Boost version is now 1.71.0.
- The customization mechanism used to implement and extend sender functionality and algorithms has been renamed from `tag_invoke` to `tag_dispatch`. All customization of sender functionality should be done by overloading `tag_dispatch`.
- The following compatibility options have been removed, along with their compatibility implementations:
 - `HPX_PROGRAM_OPTIONS_WITH_BOOST_PROGRAM_OPTIONS_COMPATIBILITY`
 - `HPX_WITH_ACTION_BASE_COMPATIBILITY`
 - `HPX_WITH_EMBEDDED_THREAD_POOLS_COMPATIBILITY`
 - `HPX_WITH_POOL_EXECUTOR_COMPATIBILITY`
 - `HPX_WITH_PROMISE_ALIAS_COMPATIBILITY`
 - `HPX_WITH_REGISTER_THREAD_COMPATIBILITY`
 - `HPX_WITH_REGISTER_THREAD_OVERLOADS_COMPATIBILITY`
 - `HPX_WITH_THREAD_AWARE_TIMER_COMPATIBILITY`
 - `HPX_WITH_THREAD_EXECUTORS_COMPATIBILITY`
 - `HPX_WITH_THREAD_POOL_OS_EXECUTOR_COMPATIBILITY`
- The `HPX_WITH_THREAD_SCHEDULERS` CMake option has been removed. All schedulers are now enabled when possible.
- `HPX_WITH_INIT_START_OVERLOADS_COMPATIBILITY` has been turned off by default.

Closed issues

- Issue #5423⁹¹⁰ - Fix lvalue-ref qualified connect for when_all-sender
- Issue #5412⁹¹¹ - Link error
- Issue #5397⁹¹² - Performance regression in thread annotations
- Issue #5395⁹¹³ - HPX 1.7.0-rc1 fails to build icw APEX + OTF2
- Issue #5385⁹¹⁴ - HPX 1.7 crashes on Piz Daint > 64 nodes
- Issue #5380⁹¹⁵ - CMake should search for asio package installed on the system
- Issue #5378⁹¹⁶ - HPX 1.7.0 stopped building on Fedora
- Issue #5369⁹¹⁷ - HPX 1.6 and master hangs on Summit for > 64 nodes
- Issue #5358⁹¹⁸ - HPX init fails for single-core environments
- Issue #5345⁹¹⁹ - Rename P2220 property CPOs?
- Issue #5333⁹²⁰ - HPX does not compile on the new Mac OSX using the M1 chip
- Issue #5317⁹²¹ - Consider making hipblas optional
- Issue #5306⁹²² - asio fails to build with CUDA 10.0
- Issue #5294⁹²³ - execution::on should be based on execution::schedule
- Issue #5275⁹²⁴ - HPX V1.6.0 fails on Fedora release
- Issue #5270⁹²⁵ - HPX-1.6.0 fails to build on Windows 10
- Issue #5257⁹²⁶ - Allow triggering the output of OS thread affinity from configuration settings
- Issue #5246⁹²⁷ - HPX fails to build on ppc64le
- Issue #5232⁹²⁸ - Annotation using hpx::util::annotated_function not working
- Issue #5222⁹²⁹ - Build and link errors with ittnotify enabled
- Issue #5204⁹³⁰ - Move algorithms to tag_fallback_dispatch
- Issue #5163⁹³¹ - Remove module-specific compatibility and deprecation options

910 <https://github.com/STELLAR-GROUP/hpx/issues/5423>

911 <https://github.com/STELLAR-GROUP/hpx/issues/5412>

912 <https://github.com/STELLAR-GROUP/hpx/issues/5397>

913 <https://github.com/STELLAR-GROUP/hpx/issues/5395>

914 <https://github.com/STELLAR-GROUP/hpx/issues/5385>

915 <https://github.com/STELLAR-GROUP/hpx/issues/5380>

916 <https://github.com/STELLAR-GROUP/hpx/issues/5378>

917 <https://github.com/STELLAR-GROUP/hpx/issues/5369>

918 <https://github.com/STELLAR-GROUP/hpx/issues/5358>

919 <https://github.com/STELLAR-GROUP/hpx/issues/5345>

920 <https://github.com/STELLAR-GROUP/hpx/issues/5333>

921 <https://github.com/STELLAR-GROUP/hpx/issues/5317>

922 <https://github.com/STELLAR-GROUP/hpx/issues/5306>

923 <https://github.com/STELLAR-GROUP/hpx/issues/5294>

924 <https://github.com/STELLAR-GROUP/hpx/issues/5275>

925 <https://github.com/STELLAR-GROUP/hpx/issues/5270>

926 <https://github.com/STELLAR-GROUP/hpx/issues/5257>

927 <https://github.com/STELLAR-GROUP/hpx/issues/5246>

928 <https://github.com/STELLAR-GROUP/hpx/issues/5232>

929 <https://github.com/STELLAR-GROUP/hpx/issues/5222>

930 <https://github.com/STELLAR-GROUP/hpx/issues/5204>

931 <https://github.com/STELLAR-GROUP/hpx/issues/5163>

- Issue #5161⁹³² - Bump required CMake version to 3.17
- Issue #5143⁹³³ - Searching for HPX-Application to generate work on multiple Nodes

Closed pull requests

- PR #5438⁹³⁴ - Delete datapar/foreach_tests.hpp
- PR #5437⁹³⁵ - Add back explicit -pthread flags when available
- PR #5435⁹³⁶ - This adds support for systems that assume all types are bitwise serializable by default
- PR #5434⁹³⁷ - Update CUDA polling logging to be more verbose
- PR #5433⁹³⁸ - Fix when_all_sender connect for references
- PR #5432⁹³⁹ - Add deprecation warnings for v1.8
- PR #5431⁹⁴⁰ - Rename the new P0443/P2300 executor to thread_pool_scheduler
- PR #5430⁹⁴¹ - Revert “Adding the missing defined for HPX_HAVE_DEPRECATED_WARNINGS”
- PR #5427⁹⁴² - Removing unneeded typedef
- PR #5426⁹⁴³ - Adding more concept checks for sender/receiver algorithms
- PR #5425⁹⁴⁴ - Adding the missing defined for HPX_HAVE_DEPRECATED_WARNINGS
- PR #5424⁹⁴⁵ - Disable Vc in final docker image created in CI
- PR #5422⁹⁴⁶ - Adding execution::experimental::bulk algorithm
- PR #5420⁹⁴⁷ - Update logic to find threading library
- PR #5418⁹⁴⁸ - Reduce max size and number of files in ccache cache
- PR #5417⁹⁴⁹ - Final release notes for 1.7.0
- PR #5416⁹⁵⁰ - Adapt uninitialized_value_construct and uninitialized_value_construct_n to C++ 20
- PR #5415⁹⁵¹ - Adapt uninitialized_default_construct and uninitialized_default_construct_n to C++ 20
- PR #5414⁹⁵² - Improve integration of futures and senders

⁹³² <https://github.com/STELLAR-GROUP/hpx/issues/5161>

⁹³³ <https://github.com/STELLAR-GROUP/hpx/issues/5143>

⁹³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5438>

⁹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5437>

⁹³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5435>

⁹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5434>

⁹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5433>

⁹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5432>

⁹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5431>

⁹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/5430>

942 <https://github.com/STELLAR-GROUP/hpx/pull/5427>943 <https://github.com/STELLAR-GROUP/hpx/pull/5426>944 <https://github.com/STELLAR-GROUP/hpx/pull/5425>945 <https://github.com/STELLAR-GROUP/hpx/pull/5424>946 <https://github.com/STELLAR-GROUP/hpx/pull/5422>947 <https://github.com/STELLAR-GROUP/hpx/pull/5420>948 <https://github.com/STELLAR-GROUP/hpx/pull/5418>949 <https://github.com/STELLAR-GROUP/hpx/pull/5417>950 <https://github.com/STELLAR-GROUP/hpx/pull/5416>951 <https://github.com/STELLAR-GROUP/hpx/pull/5415>952 <https://github.com/STELLAR-GROUP/hpx/pull/5414>

- PR #5413⁹⁵³ - Fixing sender/receiver code base to compile with MSVC
- PR #5407⁹⁵⁴ - Handle exceptions thrown during initialization of parcel handler
- PR #5406⁹⁵⁵ - Simplify dispatching to annotation handlers
- PR #5405⁹⁵⁶ - Fetch Asio automatically in perftests CI
- PR #5403⁹⁵⁷ - Create generic executor that adds annotations to any other executor
- PR #5402⁹⁵⁸ - Adapt uninitialized_fill and uninitialized_fill_n to C++ 20
- PR #5401⁹⁵⁹ - Modernize a variety of facilities related to parallel algorithms
- PR #5400⁹⁶⁰ - Fix sliding semaphore test
- PR #5399⁹⁶¹ - Rename leftover tagFallback_invoke to tagFallback_dispatch
- PR #5398⁹⁶² - Improve logging in AGAS symbol namespace
- PR #5396⁹⁶³ - Introduce compatibility layer for collective operations
- PR #5394⁹⁶⁴ - Enable OTF2 in APEX CI configuration
- PR #5393⁹⁶⁵ - Update APEX tag
- PR #5392⁹⁶⁶ - Fixing wrong usage of std::forward
- PR #5391⁹⁶⁷ - Fix forwarding in transform_receiver constructor
- PR #5390⁹⁶⁸ - Make sure shared priority scheduler steals tasks on the current NUMA domain when (core) stealing is enabled
- PR #5389⁹⁶⁹ - Adapt uninitialized_move and uninitialized_move_n to C++ 20
- PR #5388⁹⁷⁰ - Fixing gather_there for used with lvalue reference argument
- PR #5387⁹⁷¹ - Extend thread state logging and change default stealing parameters
- PR #5386⁹⁷² - Attempt to fix the startup hang with nodes > 32
- PR #5384⁹⁷³ - Remove HPX 1.5.0 deprecations
- PR #5382⁹⁷⁴ - Prefer installed Asio before considering FetchContent
- PR #5379⁹⁷⁵ - Allow using pre-downloaded (not installed) versions of Asio and/or Apex

⁹⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5413>

⁹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5407>

⁹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5406>

⁹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5405>

⁹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5403>

⁹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5402>

⁹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5401>

⁹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5400>

⁹⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/5399>

⁹⁶² <https://github.com/STELLAR-GROUP/hpx/pull/5398>

⁹⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/5396>

⁹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5394>

⁹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5393>

⁹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5392>

⁹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5391>

⁹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5390>

⁹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5389>

⁹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5388>

⁹⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/5387>

⁹⁷² <https://github.com/STELLAR-GROUP/hpx/pull/5386>

⁹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/5384>

⁹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5382>

⁹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5379>

- PR #5376⁹⁷⁶ - Remove unnecessary explicit listing of library modules.rst files in CMakeLists.txt
- PR #5375⁹⁷⁷ - Slight performance improvement for hpx::copy and hpx::move et.al.
- PR #5374⁹⁷⁸ - Remove unnecessary moves from future sender implementations
- PR #5373⁹⁷⁹ - More changes to clang-cuda Jenkins configuration
- PR #5372⁹⁸⁰ - Slight improvements to min/max/minmax_element algorithms
- PR #5371⁹⁸¹ - Adapt uninitialized_copy and uninitialized_copy_n to C++ 20
- PR #5370⁹⁸² - Decay types in just_sender value_types to match stored types
- PR #5367⁹⁸³ - Disable pkgconfig by default again on macOS
- PR #5365⁹⁸⁴ - Use ccache for Jenkins builds on Piz Daint
- PR #5363⁹⁸⁵ - Update cudatoolkit module name in clang-cuda Jenkins configuration
- PR #5362⁹⁸⁶ - Adding channel_communicator
- PR #5361⁹⁸⁷ - Fix compilation with MPI enabled
- PR #5360⁹⁸⁸ - Update APEX and asio tags
- PR #5359⁹⁸⁹ - Fix check for pu-step in single-core case
- PR #5357⁹⁹⁰ - Making sure collective operations can be reused by preallocating communicator
- PR #5356⁹⁹¹ - Update API documentation
- PR #5355⁹⁹² - Make the sequenced_executor processing_units_count member function const
- PR #5354⁹⁹³ - Making sure default_stack_size is defined whenever declared
- PR #5353⁹⁹⁴ - Add CUDA timestamp support to HPX Hardware Clock
- PR #5352⁹⁹⁵ - Adding missing includes
- PR #5351⁹⁹⁶ - Adding enable_logging/disable_logging API functions
- PR #5350⁹⁹⁷ - Adapt lexicographical_compare to C++20
- PR #5349⁹⁹⁸ - Update minimum boost version needed on the docs

⁹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5376>

⁹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5375>

⁹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5374>

⁹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5373>

⁹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5372>

⁹⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/5371>

⁹⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5370>

⁹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5367>

⁹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5365>

⁹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5363>

⁹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5362>

⁹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5361>

⁹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5360>

⁹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5359>

⁹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5357>

⁹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5356>

⁹⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5355>

⁹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5354>

⁹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5353>

⁹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5352>

⁹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5351>

⁹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5350>

⁹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5349>

- PR #5348⁹⁹⁹ - Rename `tag_invoke` and related facilities to `tag_dispatch`
- PR #5347¹⁰⁰⁰ - Remove `make_` prefix for executor properties
- PR #5346¹⁰⁰¹ - Remove and disable compatibility options for 1.7.0
- PR #5343¹⁰⁰² - Fix `timed_executor` static cast conversion
- PR #5342¹⁰⁰³ - Refactor CUDA event polling
- PR #5341¹⁰⁰⁴ - Adding `make_with_annotation` and `get_annotation` properties
- PR #5339¹⁰⁰⁵ - Making sure `hpx::util::hardware::timestamp()` is always defined
- PR #5338¹⁰⁰⁶ - Fixing `timed_executor` specializations of customization points
- PR #5335¹⁰⁰⁷ - Make `partial_algorithm` work with any number of arguments
- PR #5334¹⁰⁰⁸ - Follow up `iter_sent` include on #5225
- PR #5332¹⁰⁰⁹ - Simplify `tag_invoke` and friends
- PR #5331¹⁰¹⁰ - More work on cleaning up executor CPOs
- PR #5330¹⁰¹¹ - Add option to disable `pkgconfig` generation
- PR #5328¹⁰¹² - Adapt data parallel support using std-simd
- PR #5327¹⁰¹³ - Fix missing `ifdef HPX_SMT_PAUSE`
- PR #5326¹⁰¹⁴ - Adding `resize()` to `serialize_buffer` allowing to shrink its size
- PR #5324¹⁰¹⁵ - Add get member functions to `async_rw_mutex` proxy objects for explicitly getting the wrapped value
- PR #5323¹⁰¹⁶ - Add `keep_future` algorithm
- PR #5322¹⁰¹⁷ - Replace executor customization point implementations with `tag_invoke`
- PR #5321¹⁰¹⁸ - Separate segmented algorithms for reduce
- PR #5320¹⁰¹⁹ - Fix `is_sender` trait and other small fixes to p0443 traits
- PR #5319¹⁰²⁰ - gcc 11.1 c++20 build fixes
- PR #5318¹⁰²¹ - Make `hipblas` dependency optional as not always available

⁹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5348>

¹⁰⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5347>

¹⁰⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5346>

¹⁰⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5343>

¹⁰⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/5342>

¹⁰⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5341>

¹⁰⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5339>

¹⁰⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5338>

¹⁰⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5335>

¹⁰⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5334>

¹⁰⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5332>

¹⁰¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5331>

¹⁰¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5330>

¹⁰¹² <https://github.com/STELLAR-GROUP/hpx/pull/5328>

¹⁰¹³ <https://github.com/STELLAR-GROUP/hpx/pull/5327>

¹⁰¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5326>

¹⁰¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5324>

¹⁰¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5323>

¹⁰¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5322>

¹⁰¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5321>

¹⁰¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5320>

¹⁰²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5319>

¹⁰²¹ <https://github.com/STELLAR-GROUP/hpx/pull/5318>

- PR #5316¹⁰²² - Attempt to fix checking for libatomic
- PR #5315¹⁰²³ - Add explicit keyword to fixture constructor
- PR #5314¹⁰²⁴ - Fix a race condition in async mpi affecting limiting executor
- PR #5312¹⁰²⁵ - Use local runtime and local headers in local-only modules and tests
- PR #5311¹⁰²⁶ - Add GCC 11 builder to jenkins
- PR #5310¹⁰²⁷ - Adding `hpx::execution::experimental::task_group`
- PR #5309¹⁰²⁸ - Separate datapar
- PR #5308¹⁰²⁹ - Separate segmented algorithms for `find`, `find_if`, `find_if_not`
- PR #5307¹⁰³⁰ - Separate segmented algorithms for `fill` and `generate`
- PR #5304¹⁰³¹ - Fix compilation of sender CPOs with nvcc
- PR #5300¹⁰³² - Remove PRIVATE flag that was propagated into the LANGUAGES
- PR #5298¹⁰³³ - Separate datapar
- PR #5297¹⁰³⁴ - Specify exact cmake and ninja versions when loading them in jenkins jobs
- PR #5295¹⁰³⁵ - Update clang-newest configuration to use clang 12 and Boost 1.76.0
- PR #5293¹⁰³⁶ - Fix Clang 11 cuda_future test bug
- PR #5292¹⁰³⁷ - Add `async_rw_mutex` based on senders
- PR #5291¹⁰³⁸ - “Fix” termination detection
- PR #5290¹⁰³⁹ - Fixed source file line statements in examples documentation
- PR #5289¹⁰⁴⁰ - Allow splitting of futures holding `std::tuple`
- PR #5288¹⁰⁴¹ - Move algorithms to `tag_fallback_invoke`
- PR #5287¹⁰⁴² - Move algorithms to `tag_fallback_invoke`
- PR #5285¹⁰⁴³ - Fix clang-format failure on master
- PR #5284¹⁰⁴⁴ - Replacing `util::function_nonser` on `std::function` in `hpx_init`

¹⁰²² <https://github.com/STELLAR-GROUP/hpx/pull/5316>

¹⁰²³ <https://github.com/STELLAR-GROUP/hpx/pull/5315>

¹⁰²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5314>

¹⁰²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5312>

¹⁰²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5311>

¹⁰²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5310>

1028 <https://github.com/STELLAR-GROUP/hpx/pull/5309>1029 <https://github.com/STELLAR-GROUP/hpx/pull/5308>1030 <https://github.com/STELLAR-GROUP/hpx/pull/5307>

1031 <https://github.com/STELLAR-GROUP/hpx/pull/5304>

1032 <https://github.com/STELLAR-GROUP/hpx/pull/5300>

1033 <https://github.com/STELLAR-GROUP/hpx/pull/5298>

1034 <https://github.com/STELLAR-GROUP/hpx/pull/5297>

1035 <https://github.com/STELLAR-GROUP/hpx/pull/5295>

1036 <https://github.com/STELLAR-GROUP/hpx/pull/5293>

1037 <https://github.com/STELLAR-GROUP/hpx/pull/5292>

1038 <https://github.com/STELLAR-GROUP/hpx/pull/5291>

1039 <https://github.com/STELLAR-GROUP/hpx/pull/5290>

1040 <https://github.com/STELLAR-GROUP/hpx/pull/5289>

1041 <https://github.com/STELLAR-GROUP/hpx/pull/5288>

1042 <https://github.com/STELLAR-GROUP/hpx/pull/5287>

1043 <https://github.com/STELLAR-GROUP/hpx/pull/5285>

1044 <https://github.com/STELLAR-GROUP/hpx/pull/5284>

2.10. Releases

973

- PR #5282¹⁰⁴⁵ - Update Boost for daint 20.11 after update
- PR #5281¹⁰⁴⁶ - Fix Segmentation fault on `foreach_datapar_zipiter`
- PR #5280¹⁰⁴⁷ - Avoid modulo by zero in `counting_iterator` test
- PR #5279¹⁰⁴⁸ - Fix more GCC 10 deprecation warnings
- PR #5277¹⁰⁴⁹ - Small fixes and improvements to CUDA/MPI polling
- PR #5276¹⁰⁵⁰ - Fix typo in docs
- PR #5274¹⁰⁵¹ - More P1897 algorithms
- PR #5273¹⁰⁵² - Retry CDash submissions on failure
- PR #5272¹⁰⁵³ - Fix bogus deprecation warnings with GCC 10
- PR #5271¹⁰⁵⁴ - Correcting target ids for `symbol_namespace::iterate`
- PR #5268¹⁰⁵⁵ - Adding generic `require`, `require_concept`, and `query` properties
- PR #5267¹⁰⁵⁶ - Support annotations in `hpx::transform_reduce`
- PR #5266¹⁰⁵⁷ - Making late command line options available for local runtime
- PR #5265¹⁰⁵⁸ - Leverage `no_unique_address` for `member_pack`
- PR #5264¹⁰⁵⁹ - Adopt format in more places
- PR #5262¹⁰⁶⁰ - Install HPX in Rostam Jenkins jobs
- PR #5261¹⁰⁶¹ - Limit Rostam Jenkins jobs to marvin partition temporarily
- PR #5260¹⁰⁶² - Separate segmented algorithms for `transform_reduce`
- PR #5259¹⁰⁶³ - Making sure late command line options are recognized as configuration options
- PR #5258¹⁰⁶⁴ - Allow for HPX algorithms being invoked with std execution policies
- PR #5256¹⁰⁶⁵ - Separate segmented algorithms for `transform`
- PR #5255¹⁰⁶⁶ - Future/sender adapters
- PR #5254¹⁰⁶⁷ - Fixing datapar

¹⁰⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5282>

¹⁰⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5281>

¹⁰⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5280>

¹⁰⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5279>

¹⁰⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5277>

¹⁰⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5276>

¹⁰⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/5274>

¹⁰⁵² <https://github.com/STELLAR-GROUP/hpx/pull/5273>

¹⁰⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5272>

¹⁰⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5271>

¹⁰⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5268>

¹⁰⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5267>

¹⁰⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5266>

¹⁰⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5265>

¹⁰⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5264>

¹⁰⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5262>

¹⁰⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/5261>

¹⁰⁶² <https://github.com/STELLAR-GROUP/hpx/pull/5260>

¹⁰⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/5259>

¹⁰⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5258>

¹⁰⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5256>

¹⁰⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5255>

¹⁰⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5254>

- PR #5253¹⁰⁶⁸ - Add utility to format ranges
- PR #5252¹⁰⁶⁹ - Remove uses of Boost.Bimap
- PR #5251¹⁰⁷⁰ - Banish <iostream> from library headers
- PR #5250¹⁰⁷¹ - Try fixing vc circle ci
- PR #5249¹⁰⁷² - Adding missing header
- PR #5248¹⁰⁷³ - Use old Piz Daint modules after upgrade
- PR #5247¹⁰⁷⁴ - Significantly speedup simple `for_each`, `for_loop`, and `transform`
- PR #5245¹⁰⁷⁵ - P1897 `operator|` overloads
- PR #5244¹⁰⁷⁶ - P1897 `when_all`
- PR #5243¹⁰⁷⁷ - Make sure `HPX_DEBUG` is set based on HPX's build type, not consuming project's build type
- PR #5242¹⁰⁷⁸ - Moving last files unrelated to parcel layer to modules
- PR #5240¹⁰⁷⁹ - change namespace for `transform_loop.hpp`
- PR #5238¹⁰⁸⁰ - Make sure annotations are used in the binary transform
- PR #5237¹⁰⁸¹ - Add P1897 `just`, `just_on`, and `on` algorithms
- PR #5236¹⁰⁸² - Add an example demonstrating the use of the `invoke_function_action` facility
- PR #5235¹⁰⁸³ - Attempting to fix datapar compilation issues
- PR #5234¹⁰⁸⁴ - Fix small typo in `--hpx:local` option description
- PR #5233¹⁰⁸⁵ - Only find Boost.Iostreams if required for plugins
- PR #5231¹⁰⁸⁶ - Sort printed config options
- PR #5230¹⁰⁸⁷ - Fix C++20 replace algo adaptation misses
- PR #5229¹⁰⁸⁸ - Remove leftover Boost include from `sync_wait.hpp`
- PR #5228¹⁰⁸⁹ - Print module name only if it has custom configuration settings
- PR #5227¹⁰⁹⁰ - Update `.codespell_whitelist`

¹⁰⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5253>

¹⁰⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5252>

¹⁰⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5251>

¹⁰⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/5250>

¹⁰⁷² <https://github.com/STELLAR-GROUP/hpx/pull/5249>

¹⁰⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/5248>

¹⁰⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5247>

¹⁰⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5245>

¹⁰⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5244>

¹⁰⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5243>

¹⁰⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5242>

¹⁰⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5240>

¹⁰⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5238>

¹⁰⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/5237>

¹⁰⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5236>

¹⁰⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5235>

¹⁰⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5234>

¹⁰⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5233>

¹⁰⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5231>

¹⁰⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5230>

¹⁰⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5229>

¹⁰⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5228>

¹⁰⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5227>

- PR #5226¹⁰⁹¹ - Use new docker image in all CircleCI steps
- PR #5225¹⁰⁹² - Adapt reverse to C++20
- PR #5224¹⁰⁹³ - Separate segmented algorithms for `none_of`, `any_of` and `all_of`
- PR #5223¹⁰⁹⁴ - Fixing build system for `itnotify`
- PR #5221¹⁰⁹⁵ - Moving LCO related files to modules
- PR #5220¹⁰⁹⁶ - Separate segmented algorithms for `count` and `count_if`
- PR #5218¹⁰⁹⁷ - Separate segmented algorithms for `adjacent_find`
- PR #5217¹⁰⁹⁸ - Add a HIP github action
- PR #5215¹⁰⁹⁹ - Update ROCm to 4.0.1 on Rostam
- PR #5214¹¹⁰⁰ - Fix clang-format error in `sender.hpp`
- PR #5213¹¹⁰¹ - Removing ESSENTIAL option to the doc example
- PR #5212¹¹⁰² - Separate segmented algorithms for `for_each_n`
- PR #5211¹¹⁰³ - Minor adapted algos fixes
- PR #5210¹¹⁰⁴ - Fixing `is_invocable` deprecation warnings
- PR #5209¹¹⁰⁵ - Moving more files into modules (actions, components, init_runtime, etc.)
- PR #5208¹¹⁰⁶ - Add examples and explanation on when `tag_fallback/priority` are useful
- PR #5207¹¹⁰⁷ - Always define `HPX_COMPUTE_HOST_CODE` for host code
- PR #5206¹¹⁰⁸ - Add formatting exceptions for `libhpx` to `create_module_skeleton.py`
- PR #5205¹¹⁰⁹ - Moving all distribution policies into modules
- PR #5203¹¹¹⁰ - Move copy algorithms to `tag_fallback_invoke`
- PR #5202¹¹¹¹ - Make `HPX_WITH_PSEUDO_DEPENDENCIES` a cache variable
- PR #5201¹¹¹² - Replaced `tag_invoke` with `tag_fallback_invoke` for `adjacent_find` algorithm
- PR #5200¹¹¹³ - Moving files to (distributed) runtime module

¹⁰⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5226>

¹⁰⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5225>

¹⁰⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5224>

¹⁰⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5223>

¹⁰⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5221>

¹⁰⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5220>

¹⁰⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5218>

¹⁰⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5217>

¹⁰⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5215>

¹¹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5214>

¹¹⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5213>

¹¹⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5212>

¹¹⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/5211>

¹¹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5210>

¹¹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5209>

¹¹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5208>

¹¹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5207>

¹¹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5206>

¹¹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5205>

¹¹¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5203>

¹¹¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5202>

¹¹¹² <https://github.com/STELLAR-GROUP/hpx/pull/5201>

¹¹¹³ <https://github.com/STELLAR-GROUP/hpx/pull/5200>

- PR #5199¹¹¹⁴ - Update ICC module name on Piz Daint Jenkins configuration
- PR #5198¹¹¹⁵ - Add doxygen documentation for thread_schedule_hint
- PR #5197¹¹¹⁶ - Attempt to fix compilation of context implementations with unity build enabled
- PR #5196¹¹¹⁷ - Re-enable component tests
- PR #5195¹¹¹⁸ - Moving files related to colocation logic
- PR #5194¹¹¹⁹ - Another attempt at fixing the Fedora 35 problem
- PR #5193¹¹²⁰ - Components module
- PR #5192¹¹²¹ - Adapt replace (_if) to C++20
- PR #5190¹¹²² - Set compatibility headers by default to on
- PR #5188¹¹²³ - Bump Boost minimum version to 1.71.0
- PR #5187¹¹²⁴ - Force CMake to set the -std=c++XX flag
- PR #5186¹¹²⁵ - Remove message to print .cu extension whenever .cu files are encountered
- PR #5185¹¹²⁶ - Remove some minor unnecessary CMake options
- PR #5184¹¹²⁷ - Remove some leftover HPX_WITH_*_SCEDULER uses
- PR #5183¹¹²⁸ - Remove dependency on boost/iterators/iterator_categories.hpp
- PR #5182¹¹²⁹ - Fixing Fedora 35 for Power architectures
- PR #5181¹¹³⁰ - Bump version number and tag post 1.6.0 release
- PR #5180¹¹³¹ - Fix htts_v2 tests linking
- PR #5179¹¹³² - Make sure --hpx:local command line option is respected with networking is off but distributed runtime is on
- PR #5177¹¹³³ - Remove module cmake options
- PR #5176¹¹³⁴ - Starting to separate segmented algorithms: for_each
- PR #5174¹¹³⁵ - Don't run segmented algorithms twice on CircleCI
- PR #5173¹¹³⁶ - Fetching APEX using cmake FetchContent

¹¹¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5199>

¹¹¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5198>

¹¹¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5197>

¹¹¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5196>

¹¹¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5195>

¹¹¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5194>

¹¹²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5193>

¹¹²¹ <https://github.com/STELLAR-GROUP/hpx/pull/5192>

¹¹²² <https://github.com/STELLAR-GROUP/hpx/pull/5190>

¹¹²³ <https://github.com/STELLAR-GROUP/hpx/pull/5188>

¹¹²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5187>

¹¹²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5186>

¹¹²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5185>

¹¹²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5184>

¹¹²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5183>

¹¹²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5182>

¹¹³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5181>

¹¹³¹ <https://github.com/STELLAR-GROUP/hpx/pull/5180>

¹¹³² <https://github.com/STELLAR-GROUP/hpx/pull/5179>

¹¹³³ <https://github.com/STELLAR-GROUP/hpx/pull/5177>

¹¹³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5176>

¹¹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5174>

¹¹³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5173>

- PR #5172¹¹³⁷ - Add separate local-only entry point
- PR #5171¹¹³⁸ - Remove HPX_WITH_THREAD_SCHEDULERS CMake option
- PR #5170¹¹³⁹ - Add HPX_WITH_PRECOMPILED_HEADERS option
- PR #5166¹¹⁴⁰ - Moving some action tests to modules
- PR #5165¹¹⁴¹ - Require cmake 3.17
- PR #5164¹¹⁴² - Move thread_pool_suspension_helper files to small utility module
- PR #5160¹¹⁴³ - Adding checks ensuring modules are not cross-referenced from other module categories
- PR #5158¹¹⁴⁴ - Replace boost::asio with standalone asio
- PR #5155¹¹⁴⁵ - Allow logging when distributed runtime is off
- PR #5153¹¹⁴⁶ - Components module
- PR #5152¹¹⁴⁷ - Move more files to performance counter module
- PR #5150¹¹⁴⁸ - Adapt remove_copy (_if) to C++20
- PR #5144¹¹⁴⁹ - AGAS module
- PR #5125¹¹⁵⁰ - Adapt remove and remove_if to C++20
- PR #5117¹¹⁵¹ - Attempt to fix segfaults assumed to be caused by future_data instances going out of scope.
- PR #5099¹¹⁵² - Allow mixing debug and release builds
- PR #5092¹¹⁵³ - Replace spirit.qi with x3
- PR #5053¹¹⁵⁴ - Add P0443r14 executor and a few P1897 algorithms
- PR #5044¹¹⁵⁵ - Add performance test in jenkins and reports

¹¹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5172>

¹¹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5171>

¹¹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5170>

¹¹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5166>

¹¹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/5165>

¹¹⁴² <https://github.com/STELLAR-GROUP/hpx/pull/5164>

¹¹⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/5160>

¹¹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5158>

¹¹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5155>

¹¹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5153>

¹¹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5152>

¹¹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5150>

¹¹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5144>

¹¹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5125>

¹¹⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/5117>

¹¹⁵² <https://github.com/STELLAR-GROUP/hpx/pull/5099>

¹¹⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5092>

¹¹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5053>

¹¹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5044>

2.10.4 HPX V1.6.0 (Feb 17, 2021)

General changes

This release continues the focus on C++20 conformance with multiple new algorithms adapted to be C++20 conformant and becoming customization point objects (CPOs). We have also added experimental support for HIP, allowing previous CUDA features to now be compiled with `hipcc` and run on AMD GPUs.

- The following algorithms have been adapted to be C++20 conformant: `adjacent_find`, `includes`, `inplace_merge`, `is_heap`, `is_heap_until`, `is_partitioned`, `is_sorted`, `is_sorted_until`, `merge`, `set_difference`, `set_intersection`, `set_symmetric_difference`, `set_union`.
- Experimental HIP support can be enabled by compiling *HPX* with `hipcc`. All CUDA functionality in *HPX* can now be used with HIP. The HIP functionality is for the time being exposed through the same API as the CUDA functionality, i.e. no changes are required in user code. The CUDA, and now HIP, functionality is in the `hpx::cuda` namespace.
- We have added `partial_sort` based on Francisco Tapia's implementation.
- `hpx::init` and `hpx::start` gained new overloads taking an `hpx::init_params` struct in 1.5.0. All overloads not taking an `hpx::init_params` are now deprecated.
- We have added an experimental `fork_join_executor`. This executor can be used for OpenMP-style fork-join parallelism, where the latency of a parallel region is important for performance.
- The `parallel_executor` now uses a hierarchical spawning scheme for bulk execution, which improves data locality and performance.
- `hpx::dataflow` can now be used with executors that inject additional parameters into the call of the user-provided function.
- We have added experimental support for properties as proposed in P2220¹¹⁵⁶. Currently the only supported property is the scheduling hint on `parallel_executor`.
- `hpx::util::annotated_function` can now be passed a dynamically generated `std::string`.
- In moving functionality to new namespaces, old names have been deprecated. A deprecation warning will be issued if you are using deprecated functionality, with instructions on how to correct or ignore the warning.
- We have removed all support for C and Fortran from our build system.
- We have further reduced the use of Boost types within *HPX* (`boost::system::error_code` and `boost::detail::spinlock`).
- We have enabled more warnings in our CI builds (unused variables and unused typedefs).

Breaking changes

- `hpxMP` support has been completely removed.
- The verbs `parcelport` has been removed.
- The following compatibility options have been disabled by default: `HPX_WITH_ACTION_BASE_COMPATIBILITY`, `HPX_WITH_REGISTER_THREAD_COMPATIBILITY`, `HPX_WITH_PROMISE_ALIAS_COMPATIBILITY`, `HPX_WITH_UNSCOPED_ENUM_COMPATIBILITY`, `HPX_PROGRAM_OPTIONS_WITH_BOOST_PROGRAM_OPTIONS_COMPATIBILITY`, `HPX_WITH_EMBEDDED_THREAD_POOLS_COMPATIBILITY`, `HPX_WITH_THREAD_POOL_OS_EXECUTOR_COMPATIBILITY`, `HPX_WITH_THREAD_EXECUTORS_COMPATIBILITY`, `HPX_THREAD_AWARE_TIMER_COMPATIBILITY`,

¹¹⁵⁶ <https://wg21.link/p2220>

HPX_WITH_POOL_EXECUTOR_COMPATIBILITY. Unless noted here, the above functionalities do not come with replacements. Unscoped enumerations have been replaced by scoped enumerations. Previously deprecated unscoped enumerations are disabled by HPX_WITH_UNSCOPE_ENUM_COMPATIBILITY. Newly deprecated unscoped enumerations have been given deprecation warnings and replaced by scoped enumerations. `hpx::promise` has been replaced with `hpx::distributed::promise`. `hpx::program_options` is a drop-in replacement for `boost::program_options`. `hpx::execution::parallel_executor` now has constructors which take a thread pool, covering the use case of `hpx::threads::executors::pool_executor`. A pool can be supplied with `hpx::resource::get_thread_pool`.

Closed issues

- Issue #5148¹¹⁵⁷ - `runtime_support.hpp` does not work with newer clang
- Issue #5147¹¹⁵⁸ - Wrong results with parallel reduce
- Issue #5129¹¹⁵⁹ - Missing specialization for `std::hash<hpx::thread::id>`
- Issue #5126¹¹⁶⁰ - Use `std::string` for task annotations
- Issue #5115¹¹⁶¹ - Don't expect hwloc to always report Cores
- Issue #5113¹¹⁶² - Handle threadmanager exceptions during startup
- Issue #5112¹¹⁶³ - libatomic problems causing unexpected fails
- Issue #5089¹¹⁶⁴ - Remove non-BSL files
- Issue #5088¹¹⁶⁵ - Unwrapping problem
- Issue #5087¹¹⁶⁶ - Remove hpxMP support
- Issue #5077¹¹⁶⁷ - PAPI counters are not accessible when HPX is installed
- Issue #5075¹¹⁶⁸ - Make the structs in all `iter_sent.hpp` lower case
- Issue #5067¹¹⁶⁹ - Bug `string_util/split.hpp`
- Issue #5049¹¹⁷⁰ - Change back the hipcc jenkins config to the fury partition on rostam
- Issue #5038¹¹⁷¹ - Not all examples link in the latest HPX master
- Issue #5035¹¹⁷² - Build with HPX_WITH_EXAMPLES fails
- Issue #5019¹¹⁷³ - Broken help string for hpx
- Issue #5016¹¹⁷⁴ - `hpx::parallel::fill` fails compiling

¹¹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5148>

¹¹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5147>

¹¹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/5129>

¹¹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/5126>

¹¹⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/5115>

¹¹⁶² <https://github.com/STELLAR-GROUP/hpx/issues/5113>

¹¹⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/5112>

¹¹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5089>

¹¹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5088>

¹¹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/5087>

¹¹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/5077>

¹¹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/5075>

¹¹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/5067>

¹¹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/5049>

¹¹⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/5038>

¹¹⁷² <https://github.com/STELLAR-GROUP/hpx/issues/5035>

¹¹⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/5019>

¹¹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/5016>

- Issue #5014¹¹⁷⁵ - Rename all .cc to .cpp and .hh to .hpp
- Issue #4988¹¹⁷⁶ - MPI is not finalized if running with only one locality
- Issue #4978¹¹⁷⁷ - Change feature test macros to expand to zero/one
- Issue #4949¹¹⁷⁸ - Crash when not enabling TCP parcelport
- Issue #4933¹¹⁷⁹ - Improve test coverage for unused variable warnings etc.
- Issue #4878¹¹⁸⁰ - HPX mpi async might call MPI_FINALIZE before app calls it
- Issue #4127¹¹⁸¹ - Local runtime entry-points

Closed pull requests

- PR #5178¹¹⁸² - Fix parallel remove/remove_copy/transform namespace references in docs
- PR #5169¹¹⁸³ - Attempt to get Piz Daint jenkins setup running after maintenance
- PR #5168¹¹⁸⁴ - Remove include of itself
- PR #5167¹¹⁸⁵ - Fixing deprecation warnings that slipped through the net
- PR #5159¹¹⁸⁶ - Update APEX tag to 2.3.1
- PR #5154¹¹⁸⁷ - Splitting unit tests on circleci to avoid timeouts
- PR #5151¹¹⁸⁸ - Use C++20 on clang-newest Jenkins CI configuration
- PR #5149¹¹⁸⁹ - Rename 'module' symbols to avoid keyword conflict
- PR #5145¹¹⁹⁰ - Adjust handling of CUDA/HIP options in CMake
- PR #5142¹¹⁹¹ - Store annotated_function annotations as std::strings
- PR #5140¹¹⁹² - Scheduler mode
- PR #5139¹¹⁹³ - Fix path problem in pre-commit hook, add summary commit line
- PR #5138¹¹⁹⁴ - Add program options variable map to resource partitioner init
- PR #5137¹¹⁹⁵ - Remove the use of boost::throw_exception
- PR #5136¹¹⁹⁶ - Make sure codespell checks run on CircleCI

¹¹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/5014>

¹¹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4988>

¹¹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4978>

¹¹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4949>

¹¹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4933>

¹¹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4878>

¹¹⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/4127>

¹¹⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5178>

¹¹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5169>

¹¹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5168>

¹¹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5167>

¹¹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5159>

¹¹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5154>

¹¹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5151>

¹¹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5149>

¹¹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5145>

¹¹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5142>

¹¹⁹² <https://github.com/STELLAR-GROUP/hpx/pull/5140>

¹¹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/5139>

¹¹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5138>

¹¹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5137>

¹¹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5136>

- PR #5132¹¹⁹⁷ - Fixing spelling errors
- PR #5131¹¹⁹⁸ - Mark `counting_iterator` member functions as `HPX_HOST_DEVICE`
- PR #5130¹¹⁹⁹ - Adding specialization for `std::hash<hpx::thread::id>`
- PR #5128¹²⁰⁰ - Fixing environment handling for FreeBSD
- PR #5127¹²⁰¹ - Fix typo in fibonacci documentation
- PR #5123¹²⁰² - Reduce vector sizes in partial sort benchmarks when running in debug mode
- PR #5122¹²⁰³ - Making sure exceptions during runtime initialization are correctly reported
- PR #5121¹²⁰⁴ - Working around hwloc limitation on certain platforms
- PR #5120¹²⁰⁵ - Fixing compatibility warnings in `hpx::transform` implementation
- PR #5119¹²⁰⁶ - Use `sequential_find` and friends from separate detail header
- PR #5116¹²⁰⁷ - Fix compilation with timer pool off
- PR #5114¹²⁰⁸ - Fix 5112 - make sure `libatomic` is used when needed
- PR #5109¹²⁰⁹ - Remove default runtime mode argument from init overload, again
- PR #5108¹²¹⁰ - Refactor `iter_sent.hpp` to make structs lowercase
- PR #5107¹²¹¹ - Relax dataflow internals
- PR #5106¹²¹² - Change initialization of property CPOs to satisfy older nvcc versions
- PR #5104¹²¹³ - Fix regeneration of two files that trigger unnecessary rebuilds
- PR #5103¹²¹⁴ - Remove default runtime mode argument from start/init overloads
- PR #5102¹²¹⁵ - Untie deprecated thread enums from the CMake option
- PR #5101¹²¹⁶ - Update APEX tag for 1.6.0
- PR #5100¹²¹⁷ - Bump minimum required Boost version to 1.66 and update CI configurations
- PR #5098¹²¹⁸ - Minor fixes to public API listing
- PR #5097¹²¹⁹ - Remove hpxMP support

¹¹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5132>

¹¹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5131>

¹¹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5130>

¹²⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5128>

¹²⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/5127>

¹²⁰² <https://github.com/STELLAR-GROUP/hpx/pull/5123>

¹²⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/5122>

¹²⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5121>

¹²⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5120>

¹²⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5119>

¹²⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5116>

¹²⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5114>

¹²⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5109>

¹²¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5108>

¹²¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/5107>

¹²¹² <https://github.com/STELLAR-GROUP/hpx/pull/5106>

¹²¹³ <https://github.com/STELLAR-GROUP/hpx/pull/5104>

¹²¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5103>

¹²¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5102>

¹²¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5101>

¹²¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5100>

¹²¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5098>

¹²¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5097>

- PR #5096¹²²⁰ - Remove fractals examples
- PR #5095¹²²¹ - Use all AMD nodes again on rostam
- PR #5094¹²²² - Attempt to remove macOS workaround for GH actions environment
- PR #5093¹²²³ - Remove verbs parcelport
- PR #5091¹²²⁴ - Avoid moving from lvalues
- PR #5090¹²²⁵ - Adopt C++20 std::endian
- PR #5085¹²²⁶ - Update daint CI to use Boost 1.75.0
- PR #5084¹²²⁷ - Disable compatibility options for 1.6.0 release
- PR #5083¹²²⁸ - Remove duplicated call to the `limiting_executor` in `future_overhead` test
- PR #5079¹²²⁹ - Add checks to make sure that MPI/CUDA polling is enabled/not disabled too early
- PR #5078¹²³⁰ - Add install lib directory to list of component search paths
- PR #5076¹²³¹ - Fix a typo in the jenkins clang-newest cmake config
- PR #5074¹²³² - Fixing warnings generated by MSVC
- PR #5073¹²³³ - Allow using noncopyable types with unwrapping
- PR #5072¹²³⁴ - Fix `is_convertible` args in `result_types`
- PR #5071¹²³⁵ - Fix unused parameters
- PR #5070¹²³⁶ - Fix unused variables warnings in hipcc
- PR #5069¹²³⁷ - Add support for sentinels to `adjacent_find`
- PR #5068¹²³⁸ - Fix string split function
- PR #5066¹²³⁹ - Adapt search to C++20 and Range TS
- PR #5065¹²⁴⁰ - Fix `hpx::range::adjacent_find` doxygen function signatures
- PR #5064¹²⁴¹ - Refactor runtime configuration, command line handling, and resource partitioner
- PR #5063¹²⁴² - Limit the device code guards to the distributed parts of the `future_overhead` bench

¹²²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5096>

¹²²¹ <https://github.com/STELLAR-GROUP/hpx/pull/5095>

¹²²² <https://github.com/STELLAR-GROUP/hpx/pull/5094>

¹²²³ <https://github.com/STELLAR-GROUP/hpx/pull/5093>

¹²²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5091>

¹²²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5090>

¹²²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5085>

¹²²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5084>

¹²²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5083>

¹²²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5079>

¹²³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5078>

¹²³¹ <https://github.com/STELLAR-GROUP/hpx/pull/5076>

¹²³² <https://github.com/STELLAR-GROUP/hpx/pull/5074>

¹²³³ <https://github.com/STELLAR-GROUP/hpx/pull/5073>

¹²³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5072>

¹²³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5071>

¹²³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5070>

¹²³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5069>

¹²³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5068>

¹²³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5066>

¹²⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5065>

¹²⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/5064>

¹²⁴² <https://github.com/STELLAR-GROUP/hpx/pull/5063>

- PR #5061¹²⁴³ - Remove hipcc guards in examples and tests
- PR #5060¹²⁴⁴ - Fix deprecation warnings generated by msvc
- PR #5059¹²⁴⁵ - Add warning about suspending/resuming the runtime in multi-locality scenarios
- PR #5057¹²⁴⁶ - Fix unused variable warnings
- PR #5056¹²⁴⁷ - Fix `hpx::util::get`
- PR #5055¹²⁴⁸ - Remove hipcc guards
- PR #5054¹²⁴⁹ - Fix typo
- PR #5051¹²⁵⁰ - Adapt transform to C++20
- PR #5050¹²⁵¹ - Replace old init overloads in tests and examples
- PR #5048¹²⁵² - Limit jenkins hipcc to the reno node
- PR #5047¹²⁵³ - Limit cuda jenkins run to nodes with exclusively Nvidia GPUs
- PR #5046¹²⁵⁴ - Convert thread and future enums to class enums
- PR #5043¹²⁵⁵ - Improve `hpxrun.py` for Phylanx
- PR #5042¹²⁵⁶ - Add missing header to partial sort test
- PR #5041¹²⁵⁷ - Adding Francisco Tapia's implementation of `partial_sort`
- PR #5040¹²⁵⁸ - Remove generated headers left behind from a previous configuration
- PR #5039¹²⁵⁹ - Fix GCC 10 release builds
- PR #5037¹²⁶⁰ - Add `is_invocable` typedefs to top-level hpx namespace and public API list
- PR #5036¹²⁶¹ - Deprecate `hpx::util::decay` in favor of `std::decay`
- PR #5034¹²⁶² - Use versioned container image on CircleCI
- PR #5033¹²⁶³ - Implement P2220 properties module
- PR #5032¹²⁶⁴ - Do codespell comparison only on files changed from common ancestor
- PR #5031¹²⁶⁵ - Moving traits files to `actions_base`

¹²⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/5061>

¹²⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5060>

¹²⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5059>

¹²⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5057>

¹²⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5056>

¹²⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5055>

¹²⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5054>

¹²⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5051>

¹²⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/5050>

¹²⁵² <https://github.com/STELLAR-GROUP/hpx/pull/5048>

¹²⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/5047>

¹²⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5046>

¹²⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5043>

¹²⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5042>

¹²⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5041>

¹²⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5040>

¹²⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5039>

¹²⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5037>

¹²⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/5036>

¹²⁶² <https://github.com/STELLAR-GROUP/hpx/pull/5034>

¹²⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/5033>

¹²⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5032>

¹²⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5031>

- PR #5030¹²⁶⁶ - Add codespell version print in circleci
- PR #5029¹²⁶⁷ - Work around problems in GitHub actions macOS builder
- PR #5028¹²⁶⁸ - Moving move files to naming and naming_base
- PR #5027¹²⁶⁹ - Lessen constraints on certain algorithm arguments
- PR #5025¹²⁷⁰ - Adapt is_sorted and is_sorted_until to C++20
- PR #5024¹²⁷¹ - Moving naming_base to full modules
- PR #5022¹²⁷² - Remove C language from CMakeLists.txt
- PR #5021¹²⁷³ - Warn about unused arguments given to add_hpx_module
- PR #5020¹²⁷⁴ - Fixing help string
- PR #5018¹²⁷⁵ - Update CSCS jenkins configuration to clang 11
- PR #5017¹²⁷⁶ - Fixing broken backwards compatibility for hpx::parallel::fill
- PR #5015¹²⁷⁷ - Detect if generated global header conflicts with explicitly listed module headers
- PR #5012¹²⁷⁸ - Properly reset pointer tracking data in output_archive
- PR #5011¹²⁷⁹ - Inspect command line tweaks
- PR #5010¹²⁸⁰ - Creating AGAS module
- PR #5009¹²⁸¹ - Replace boost::system::error_code with std::error_code
- PR #5008¹²⁸² - Replace uses of boost::detail::spinlock
- PR #5007¹²⁸³ - Bump minimal Boost version to 1.65.0
- PR #5006¹²⁸⁴ - Adapt is_partitioned to C++20
- PR #5005¹²⁸⁵ - Making sure reduce_by_key compiles again
- PR #5004¹²⁸⁶ - Fixing template specializations that make extra archive data types unique across module boundaries
- PR #5003¹²⁸⁷ - Relax dataflow argument constraints
- PR #5001¹²⁸⁸ - Add <random> inspect check

¹²⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5030>

¹²⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5029>

¹²⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5028>

¹²⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5027>

¹²⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5025>

¹²⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/5024>

¹²⁷² <https://github.com/STELLAR-GROUP/hpx/pull/5022>

¹²⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/5021>

¹²⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5020>

¹²⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5018>

¹²⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5017>

¹²⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5015>

¹²⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5012>

¹²⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/5011>

¹²⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/5010>

¹²⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/5009>

¹²⁸² <https://github.com/STELLAR-GROUP/hpx/pull/5008>

¹²⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/5007>

¹²⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/5006>

¹²⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/5005>

¹²⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/5004>

¹²⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/5003>

¹²⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/5001>

- PR #4999¹²⁸⁹ - Attempt to fix MacOS Github action error
- PR #4997¹²⁹⁰ - Fix unused variable and typedef warnings
- PR #4996¹²⁹¹ - Adapt adjacent_find to C++20
- PR #4995¹²⁹² - Test all schedulers in cross_pool_injection test except shared_priority_queue_scheduler
- PR #4993¹²⁹³ - Fix deprecation warnings
- PR #4991¹²⁹⁴ - Avoid unnecessarily including entire modules
- PR #4990¹²⁹⁵ - Fixing some warnings from HPX complaining about use of obsolete types
- PR #4989¹²⁹⁶ - add a *destroy* trait for ParcelPort plugins
- PR #4986¹²⁹⁷ - Remove serialization to functional module dependency
- PR #4985¹²⁹⁸ - Compatibility header generation
- PR #4980¹²⁹⁹ - Add ranges overloads to for_loop (and variants)
- PR #4979¹³⁰⁰ - Actually enable unity builds on Jenkins
- PR #4977¹³⁰¹ - Cleaning up debug::print functionalities
- PR #4976¹³⁰² - Remove indirection layer in at_index_impl
- PR #4975¹³⁰³ - Remove indirection layer in at_index_impl
- PR #4973¹³⁰⁴ - Avoid warnings/errors for older gcc complaining about multi-line comments
- PR #4970¹³⁰⁵ - Making set algorithms conform to C++20
- PR #4969¹³⁰⁶ - Moving is_execution_policy and friends into namespace hpx
- PR #4968¹³⁰⁷ - Enable deprecation warnings for 1.6.0 and move any functionality to hpx namespace
- PR #4967¹³⁰⁸ - Define deprecation macros conditionally
- PR #4966¹³⁰⁹ - Add clang-format and cmake-format version prints
- PR #4965¹³¹⁰ - Making is_heap and is_heap_until conforming to C++20
- PR #4964¹³¹¹ - Adding parallel make_heap

¹²⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4999>

¹²⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4997>

¹²⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4996>

¹²⁹² <https://github.com/STELLAR-GROUP/hpx/pull/4995>

¹²⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/4993>

¹²⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4991>

¹²⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4990>

¹²⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4989>

¹²⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4986>

¹²⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4985>

¹²⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4980>

¹³⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4979>

¹³⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/4977>

¹³⁰² <https://github.com/STELLAR-GROUP/hpx/pull/4976>

¹³⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/4975>

¹³⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4973>

¹³⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4970>

¹³⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4969>

¹³⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4968>

¹³⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4967>

¹³⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4966>

¹³¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4965>

¹³¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4964>

- PR #4962¹³¹² - Fix external timer function pointer exports
- PR #4960¹³¹³ - Fixing folder names for module tests and examples
- PR #4959¹³¹⁴ - Adding communications set
- PR #4958¹³¹⁵ - Deprecate tuple and timing functionality in `hpx::util`
- PR #4957¹³¹⁶ - Fixing unity build option for parcelports
- PR #4953¹³¹⁷ - Fixing MSVC problems after recent restructurings
- PR #4952¹³¹⁸ - Make `parallel_executor` use `thread_pool_executor` spawning mechanism
- PR #4948¹³¹⁹ - Clean up old artifacts better and more aggressively on Jenkins
- PR #4947¹³²⁰ - Add HIP support for AMD GPUs
- PR #4945¹³²¹ - Enable `HPX_WITH_UNITY_BUILD` option on one of the Jenkins configurations
- PR #4943¹³²² - Move public `hpx::parallel::execution` functionality to `hpx::execution`
- PR #4938¹³²³ - Post release cleanup
- PR #4858¹³²⁴ - Extending resilience APIs to support distributed invocations
- PR #4744¹³²⁵ - Fork-join executor
- PR #4665¹³²⁶ - Implementing sender, receiver, and `operation_state` concepts in terms of P0443r13
- PR #4649¹³²⁷ - Split `libhpx` into multiple libraries
- PR #4642¹³²⁸ - Implementing `operation_state` concept in terms of P0443r13
- PR #4640¹³²⁹ - Implementing receiver concept in terms of P0443r13
- PR #4622¹³³⁰ - Sanitizer fixes

¹³¹² <https://github.com/STELLAR-GROUP/hpx/pull/4962>

¹³¹³ <https://github.com/STELLAR-GROUP/hpx/pull/4960>

¹³¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4959>

¹³¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4958>

¹³¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4957>

¹³¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4953>

¹³¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4952>

¹³¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4948>

¹³²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4947>

¹³²¹ <https://github.com/STELLAR-GROUP/hpx/pull/4945>

¹³²² <https://github.com/STELLAR-GROUP/hpx/pull/4943>

¹³²³ <https://github.com/STELLAR-GROUP/hpx/pull/4938>

¹³²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4858>

¹³²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4744>

¹³²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4665>

¹³²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4649>

¹³²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4642>

¹³²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4640>

¹³³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4622>

2.10.5 HPX V1.5.1 (Sep 30, 2020)

General changes

This is a patch release. It contains the following changes:

- Remove restriction on suspending runtime with multiple localities, users are now responsible for synchronizing work between localities before suspending.
- Fixes several compilation problems and warnings.
- Adds notes in the documentation explaining how to cite HPX.

Closed issues

- Issue #4971¹³³¹ - Parallel sort fails to compile with C++20
- Issue #4950¹³³² - Build with *HPX_WITH_PARCELPORT_ACTION_COUNTERS ON* fails
- Issue #4940¹³³³ - Codespell report for “HPX” (on fossies.org)
- Issue #4937¹³³⁴ - Allow suspension of runtime for multiple localities

Closed pull requests

- PR #4982¹³³⁵ - Add page about citing HPX to documentation
- PR #4981¹³³⁶ - Adding the missing include
- PR #4974¹³³⁷ - Remove leftover format export hack
- PR #4972¹³³⁸ - Removing use of `get_temporary_buffer` and `return_temporary_buffer`
- PR #4963¹³³⁹ - Renaming files to avoid warnings from the vs build system
- PR #4951¹³⁴⁰ - Fixing build if `HPX_WITH_PARCELPORT_ACTION_COUNTERS=On`
- PR #4946¹³⁴¹ - Allow suspension on multiple localities
- PR #4944¹³⁴² - Fix typos reported by fossies codespell report
- PR #4941¹³⁴³ - Adding some explanation to README about how to cite HPX
- PR #4939¹³⁴⁴ - Small changes

¹³³¹ <https://github.com/STELLAR-GROUP/hpx/issues/4971>

¹³³² <https://github.com/STELLAR-GROUP/hpx/issues/4950>

¹³³³ <https://github.com/STELLAR-GROUP/hpx/issues/4940>

¹³³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4937>

¹³³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4982>

¹³³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4981>

¹³³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4974>

¹³³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4972>

1339 <https://github.com/STELLAR-GROUP/hpx/pull/4963>1340 <https://github.com/STELLAR-GROUP/hpx/pull/4951>1341 <https://github.com/STELLAR-GROUP/hpx/pull/4946>1342 <https://github.com/STELLAR-GROUP/hpx/pull/4944>1343 <https://github.com/STELLAR-GROUP/hpx/pull/4941>1344 <https://github.com/STELLAR-GROUP/hpx/pull/4939>

2.10.6 HPX V1.5.0 (Sep 02, 2020)

General changes

The main focus of this release is on APIs and C++20 conformance. We have added many new C++20 features and adapted multiple algorithms to be fully C++20 conformant. As part of the modularization we have begun specifying the public API of *HPX* in terms of headers and functionality, and aligning it more closely to the C++ standard. All non-distributed modules are now in place, along with an experimental option to completely disable distributed features in *HPX*. We have also added experimental asynchronous MPI and CUDA executors. Lastly this release introduces CMake targets for depending projects, performance improvements, and many bug fixes.

- We have added the C++20 features `hpx::jthread` and `hpx::stop_token`. `hpx::condition_variable_any` now exposes new functions supporting `hpx::stop_token`.
- We have added `hpx::stable_sort` based on Francisco Tapia's implementation.
- We have adapted existing synchronization primitives to be fully conformant C++20: `hpx::barrier`, `hpx::latch`, `hpx::counting_semaphore`, and `hpx::binary_semaphore`.
- We have started using customization point objects (CPOs) to make the corresponding algorithms fully conformant to C++20 as well as to make algorithm extension easier for the user. `all_of/any_of/none_of`, `copy`, `count`, `destroy`, `equal`, `fill`, `find`, `for_each`, `generate`, `mismatch`, `move`, `reduce`, `transform_reduce` are using those CPOs (all in namespace `hpx`). We also have adapted their corresponding `hpx::ranges` versions to be conforming to C++20 in this release.
- We have adapted support for `co_await` to C++20, in addition to `hpx::future` it now also supports `hpx::shared_future`. We have also added allocator support for futures returned by `co_return`. It is no longer in the experimental namespace.
- We added serialization support for `std::variant` and `std::tuple`.
- `result_of` and `is_callable` are now deprecated and replaced by `invoke_result` and `is_invocable` to conform to C++20.
- We continued with the modularization, making it easier for us to add the new experimental `HPX_WITH_DISTRIBUTED_RUNTIME` CMake option (see below). A significant amount of headers have been deprecated. We adapted the namespaces and headers we could to be closer to the standard ones (*Public API*). Depending code should still compile, however warnings are now generated instructing to change the include statements accordingly.
- It is now possible to have a basic CUDA support including a helper function to get a future from a CUDA stream and target handling. They are available under the `hpx::cuda::experimental` namespace and they can be enabled with the `-DHPX_WITH_ASYNC_CUDA=ON` CMake option.
- We added a new `hpx::mpi::experimental` namespace for getting futures from an asynchronous MPI call and a new minimal MPI executor `hpx::mpi::experimental::executor`. These can be enabled with the `-DHPX_WITH_ASYNC_MPI=On` CMake option.
- A polymorphic executor has been implemented to reduce compile times as a function accepting executors can potentially be instantiated only once instead of multiple times with different executors. It accepts the function signature as a template argument. It needs to be constructed from any other executor. Please note, that the function signatures that can be scheduled using `then_execute`, `bulk_sync_execute`, `bulk_async_execute` and `bulk_then_execute` are slightly different (See the comment in PR #4514¹³⁴⁵ for more details).
- The underlying executor of `block_executor` has been updated to a newer one.
- We have added a parameter to `auto_chunk_size` to control the amount of iterations to measure.

¹³⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4514>

- All executor parameter hooks can now be exposed through the executor itself. This will allow to deprecate the `.with()` functionality on execution policies in the future. This is also a first step towards simplifying our executor APIs in preparation for the upcoming C++23 executors (senders/receivers).
- We have moved all of the existing APIs related to resiliency into the namespace `hpx::resiliency::experimental`. Please note this is a breaking change without backwards-compatibility option. We have converted all of those APIs to be based on customization point objects. Two new executors have been added to enable easy integration of the existing resiliency features with other facilities (like the parallel algorithms): `replay_executor` and `replicate_executor`.
- We have added performance counters type information (aggregating, monotonically increasing, average count, average timer, etc.).
- HPX threads are now re-scheduled on the same worker thread they were suspended on to avoid cache misses from moving from one thread to the other. This behavior doesn't prevent the thread from being stolen, however.
- We have added a new configuration option `hpx.exception_verbosity` to allow to control the level of verbosity of the exceptions (3 levels available).
- `broadcast_to`, `broadcast_from`, `scatter_to` and `scatter_from` have been added to the collectives, modernization of `gather_here` and `gather_there` with futures taken by rvalue references. See the breaking change on `all_to_all` in the next section. None of the collectives need supporting macros anymore (e.g. specifying the data types used for a collective operation using `HPX_REGISTER_ALLGATHER` and similar is not needed anymore).
- New API functions have been added: a) to get the number of cores which are idle (`hpx::get_idle_core_count`) and b) returning a bitmask representing the currently idle cores (`hpx::get_idle_core_mask`).
- We have added an experimental option to only enable the local runtime, you can disable the distributed runtime with `HPX_WITH_DISTRIBUTED_RUNTIME=OFF`. You can also enable the local runtime by using the `--hpx:local` runtime option.
- We fixed task annotations for actions.
- The alias `hpx::promise` to `hpx::lcos::promise` is now deprecated. You can use `hpx::lcos::promise` directly instead. `hpx::promise` will refer to the local-only promise in the future.
- We have added `hpx::upgrade_lock` and `hpx::upgrade_to_unique_lock`, which make `hpx::shared_mutex` (and similar) usable in more flexible ways.
- We have changed the CMake targets exposed to the user, it now includes `HPX::hpx`, `HPX::wrap_main` (int `main` as the first HPX thread of the application, see [Starting the HPX runtime](#)), `HPX::plugin`, `HPX::component`. The CMake variables `HPX_INCLUDE_DIRS` and `HPX_LIBRARIES` are deprecated and will be removed in a future release, you should now link directly to the `HPX::hpx` CMake target.
- A new example is demonstrating how to create and use a wrapping executor (`quickstart/executor_with_thread_hooks.cpp`)
- A new example is demonstrating how to disable thread stealing during the execution of parallel algorithms (`quickstart/disable_thread_stealing_executor.cpp`)
- We now require for our CMake build system configuration files to be formatted using `cmake-format`.
- We have removed more dependencies on various Boost libraries.
- We have added an experimental option enabling unity builds of HPX using the `-DHPX_WITH_UNITY_BUILD=On` CMake option.

- Many bug fixes.

Breaking changes

- HPX now requires a C++14 capable compiler. We have set the HPX C++ standard automatically to C++14 and if it needs to be set explicitly, it should be specified through the `CMAKE_CXX_STANDARD` setting as mandated by CMake. The `HPX_WITH_CXX*` variables are now deprecated and will be removed in the future.
- Building and using HPX is now supported only when using CMake V3.13 or later, Boost V1.64 or newer, and when compiling with clang V5, gcc V7, or VS2019, or later. Other compilers might still work but have not been tested thoroughly.
- We have added a `hpx::init_params` struct to pass parameters for HPX initialization e.g. the resource partitioner callback to initialize thread pools ([Using the resource partitioner](#)).
- The `all_to_all` algorithm is renamed to `all_gather`, and the new `all_to_all` algorithm is not compatible with the old one.
- We have moved all of the existing APIs related to resiliency into the namespace `hpx::resiliency::experimental`.

Closed issues

- Issue #4918^{[1346](#)} - Rename distributed_executors module
- Issue #4900^{[1347](#)} - Adding JOSS status badge to README
- Issue #4897^{[1348](#)} - Compiler warning, deprecated header used by HPX itself
- Issue #4886^{[1349](#)} - A future bound to an action executing on a different locality doesn't capture exception state
- Issue #4880^{[1350](#)} - Undefined reference to main build error when `HPX_WITH_DYNAMIC_HPX_MAIN=OFF`
- Issue #4877^{[1351](#)} - `hpx_main` might not able to start hpx runtime properly
- Issue #4850^{[1352](#)} - Issues creating templated component
- Issue #4829^{[1353](#)} - Spack package & `HPX_WITH_GENERIC_CONTEXT_COROUTINES`
- Issue #4820^{[1354](#)} - PAPI counters don't work
- Issue #4818^{[1355](#)} - HPX can't be used with IO pool turned off
- Issue #4816^{[1356](#)} - Build of HPX fails when `find_package(Boost)` is called before `FetchContent_MakeAvailable(hpx)`
- Issue #4813^{[1357](#)} - HPX MPI Future failed
- Issue #4811^{[1358](#)} - Remove `HPX::hpx_no_wrap_main` target before 1.5.0 release

¹³⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4918>

¹³⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4900>

¹³⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4897>

¹³⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4886>

¹³⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4880>

¹³⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/4877>

¹³⁵² <https://github.com/STELLAR-GROUP/hpx/issues/4850>

¹³⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/4829>

¹³⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4820>

¹³⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4818>

¹³⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4816>

¹³⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4813>

¹³⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4811>

- Issue #4810¹³⁵⁹ - In hpx::for_each::invoke_projected the hpx::util::decay is misguided
- Issue #4787¹³⁶⁰ - transform_inclusive_scan gives incorrect results for non-commutative operator
- Issue #4786¹³⁶¹ - transform_inclusive_scan tries to implicitly convert between types, instead of using the provided conv function
- Issue #4779¹³⁶² - HPX build error with GCC 10.1
- Issue #4766¹³⁶³ - Move HPX.Compute functionality to experimental namespace
- Issue #4763¹³⁶⁴ - License file name
- Issue #4758¹³⁶⁵ - CMake profiling results
- Issue #4755¹³⁶⁶ - Building HPX with support for PAPI fails
- Issue #4754¹³⁶⁷ - CMake cache creation breaks when using HPX with mimalloc
- Issue #4752¹³⁶⁸ - HPX MPI Future build failed
- Issue #4746¹³⁶⁹ - Memory leak when using dataflow icw components
- Issue #4731¹³⁷⁰ - Bug in stencil example, calculation of locality IDs
- Issue #4723¹³⁷¹ - Build fail with NETWORKING OFF
- Issue #4720¹³⁷² - Add compatibility headers for modules that had their module headers implicitly generated in 1.4.1
- Issue #4719¹³⁷³ - Undeprecate some module headers
- Issue #4712¹³⁷⁴ - Rename HPX_MPI_WITH_FUTURES option
- Issue #4709¹³⁷⁵ - Make deprecation warnings overridable in dependent projects
- Issue #4691¹³⁷⁶ - Suggestion to fix and enhance the thread_mapper API
- Issue #4686¹³⁷⁷ - Fix tutorials examples
- Issue #4685¹³⁷⁸ - HPX distributed map fails to compile
- Issue #4680¹³⁷⁹ - Build error with HPX_WITH_DYNAMIC_HPX_MAIN=OFF
- Issue #4679¹³⁸⁰ - Build error for hpx w/ Apex on Summit

¹³⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4810>

¹³⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4787>

¹³⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/4786>

¹³⁶² <https://github.com/STELLAR-GROUP/hpx/issues/4779>

¹³⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/4766>

¹³⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4763>

¹³⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4755>

¹³⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4755>

¹³⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4754>

¹³⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4752>

¹³⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4746>

1370 <https://github.com/STELLAR-GROUP/hpx/issues/4731>1371 <https://github.com/STELLAR-GROUP/hpx/issues/4723>1372 <https://github.com/STELLAR-GROUP/hpx/issues/4720>1373 <https://github.com/STELLAR-GROUP/hpx/issues/4719>1374 <https://github.com/STELLAR-GROUP/hpx/issues/4712>1375 <https://github.com/STELLAR-GROUP/hpx/issues/4709>1376 <https://github.com/STELLAR-GROUP/hpx/issues/4691>1377 <https://github.com/STELLAR-GROUP/hpx/issues/4686>1378 <https://github.com/STELLAR-GROUP/hpx/issues/4685>1379 <https://github.com/STELLAR-GROUP/hpx/issues/4680>1380 <https://github.com/STELLAR-GROUP/hpx/issues/4679>

- Issue #4675¹³⁸¹ - build error with HPX_WITH_NETWORKING=OFF
- Issue #4674¹³⁸² - Error running Quickstart tests on OS X
- Issue #4662¹³⁸³ - MPI initialization broken when networking off
- Issue #4652¹³⁸⁴ - How to fix distributed action annotation
- Issue #4650¹³⁸⁵ - thread descriptions are broken... again
- Issue #4648¹³⁸⁶ - Thread stacksize not properly set
- Issue #4647¹³⁸⁷ - Rename generated collective headers in modules
- Issue #4639¹³⁸⁸ - Update deprecation warnings in compatibility headers to point to collective headers
- Issue #4628¹³⁸⁹ - mpi parcelport totally broken
- Issue #4619¹³⁹⁰ - Fully document hpx_wrap behaviour and targets
- Issue #4612¹³⁹¹ - Compilation issue with HPX 1.4.1 and 1.4.0
- Issue #4594¹³⁹² - Rename modules
- Issue #4578¹³⁹³ - Default value for HPX_WITH_THREAD_BACKTRACE_DEPTH
- Issue #4572¹³⁹⁴ - Thread manager should be given a runtime_configuration
- Issue #4571¹³⁹⁵ - Add high-level documentation to new modules
- Issue #4569¹³⁹⁶ - Annoying warning when compiling - pls suppress or fix it.
- Issue #4555¹³⁹⁷ - HPX_HAVE_THREAD_BACKTRACE_ON_SUSPENSION compilation error
- Issue #4543¹³⁹⁸ - Segfaults in Release builds using *sleep_for*
- Issue #4539¹³⁹⁹ - Compilation Error when HPX_MPI_WITH_FUTURES=ON
- Issue #4537¹⁴⁰⁰ - Linking issue with libhpx_initd.a
- Issue #4535¹⁴⁰¹ - API for checking if pool with a given name exists
- Issue #4523¹⁴⁰² - Build of PR #4311 (git tag 9955e8e) fails
- Issue #4519¹⁴⁰³ - Documentation problem

¹³⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/4675>

¹³⁸² <https://github.com/STELLAR-GROUP/hpx/issues/4674>

¹³⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/4662>

¹³⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4652>

¹³⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4650>

¹³⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4648>

¹³⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4647>

¹³⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4639>

¹³⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4628>

¹³⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4619>

¹³⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/4612>

¹³⁹² <https://github.com/STELLAR-GROUP/hpx/issues/4594>

¹³⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/4578>

¹³⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4572>

¹³⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4571>

¹³⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4569>

¹³⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4555>

¹³⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4543>

¹³⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4539>

¹⁴⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4537>

¹⁴⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/4535>

¹⁴⁰² <https://github.com/STELLAR-GROUP/hpx/issues/4523>

¹⁴⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/4519>

- Issue #4513¹⁴⁰⁴ - HPXConfig.cmake contains ill-formed paths when library paths use backslashes
- Issue #4507¹⁴⁰⁵ - User-polling introduced by MPI futures module should be more generally usable
- Issue #4506¹⁴⁰⁶ - Make sure force_linking.hpp is not included in main module header
- Issue #4501¹⁴⁰⁷ - Fix compilation of PAPI tests
- Issue #4497¹⁴⁰⁸ - Add modules CI checks
- Issue #4489¹⁴⁰⁹ - Polymorphic executor
- Issue #4476¹⁴¹⁰ - Use CMake targets defined by FindBoost
- Issue #4473¹⁴¹¹ - Add vcpkg installation instructions
- Issue #4470¹⁴¹² - Adapt hpx::future to C++20 co_await
- Issue #4468¹⁴¹³ - Compile error on Raspberry Pi 4
- Issue #4466¹⁴¹⁴ - Compile error on Windows, current stable:
- Issue #4453¹⁴¹⁵ - Installing HPX on fedora with dnf is not adding cmake files
- Issue #4448¹⁴¹⁶ - New std::variant serialization broken
- Issue #4438¹⁴¹⁷ - Add performance counter flag is monotonically increasing
- Issue #4436¹⁴¹⁸ - Build problem: same code build and works with 1.4.0 but it doesn't with 1.4.1
- Issue #4429¹⁴¹⁹ - Function descriptions not supported in distributed
- Issue #4423¹⁴²⁰ - --hpx:ini=hpx.lock_detection=0 has no effect
- Issue #4422¹⁴²¹ - Add performance counter metadata
- Issue #4419¹⁴²² - Weird behavior for --hpx:print-counter-interval with large numbers
- Issue #4401¹⁴²³ - Create module repository
- Issue #4400¹⁴²⁴ - Command line options conflict related to performance counters
- Issue #4349¹⁴²⁵ - --hpx:use-process-mask option throw an exception on OS X
- Issue #4345¹⁴²⁶ - Move gh-pages branch out of hpx repo

¹⁴⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4513>

¹⁴⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4507>

¹⁴⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4506>

¹⁴⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4501>

¹⁴⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4497>

¹⁴⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4489>

¹⁴¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4476>

¹⁴¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/4473>

¹⁴¹² <https://github.com/STELLAR-GROUP/hpx/issues/4470>

1413 <https://github.com/STELLAR-GROUP/hpx/issues/4468>1414 <https://github.com/STELLAR-GROUP/hpx/issues/4466>1415 <https://github.com/STELLAR-GROUP/hpx/issues/4453>1416 <https://github.com/STELLAR-GROUP/hpx/issues/4448>1417 <https://github.com/STELLAR-GROUP/hpx/issues/4438>1418 <https://github.com/STELLAR-GROUP/hpx/issues/4436>1419 <https://github.com/STELLAR-GROUP/hpx/issues/4429>1420 <https://github.com/STELLAR-GROUP/hpx/issues/4423>1421 <https://github.com/STELLAR-GROUP/hpx/issues/4422>1422 <https://github.com/STELLAR-GROUP/hpx/issues/4419>1423 <https://github.com/STELLAR-GROUP/hpx/issues/4401>1424 <https://github.com/STELLAR-GROUP/hpx/issues/4400>1425 <https://github.com/STELLAR-GROUP/hpx/issues/4349>1426 <https://github.com/STELLAR-GROUP/hpx/issues/4345>

- Issue #4323¹⁴²⁷ - Const-correctness error in assignment operator of compute::vector
- Issue #4318¹⁴²⁸ - ASIO breaks with C++2a concepts
- Issue #4317¹⁴²⁹ - Application runs even if `-hpx:help` is specified
- Issue #4063¹⁴³⁰ - Document hpxcxx compiler wrapper
- Issue #3983¹⁴³¹ - Implement the C++20 Synchronization Library
- Issue #3696¹⁴³² - C++11 `constexpr` support is now required
- Issue #3623¹⁴³³ - Modular HPX branch and an alternative project layout
- Issue #2836¹⁴³⁴ - The worst-case time complexity of parallel::sort seems to be O(N^2).

Closed pull requests

- PR #4936¹⁴³⁵ - Minor documentation fixes part 2
- PR #4935¹⁴³⁶ - Add copyright and license to joss paper file
- PR #4934¹⁴³⁷ - Adding Semicolon in Documentation
- PR #4932¹⁴³⁸ - Fixing compiler warnings
- PR #4931¹⁴³⁹ - Small documentation formatting fixes
- PR #4930¹⁴⁴⁰ - Documentation Distributed HPX applications localvv with local_vv
- PR #4929¹⁴⁴¹ - Add final version of the JOSS paper
- PR #4928¹⁴⁴² - Add HPX_NODISCARD to enable_user_polling structs
- PR #4926¹⁴⁴³ - Rename distributed_executors module to executors_distributed
- PR #4925¹⁴⁴⁴ - Making transform_reduce conforming to C++20
- PR #4923¹⁴⁴⁵ - Don't acquire lock if not needed
- PR #4921¹⁴⁴⁶ - Update the release notes for the release candidate 3
- PR #4920¹⁴⁴⁷ - Disable libcds release
- PR #4919¹⁴⁴⁸ - Make cuda event pool dynamic instead of fixed size

¹⁴²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4323>

¹⁴²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4318>

¹⁴²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4317>

¹⁴³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4063>

¹⁴³¹ <https://github.com/STELLAR-GROUP/hpx/issues/3983>

¹⁴³² <https://github.com/STELLAR-GROUP/hpx/issues/3696>

¹⁴³³ <https://github.com/STELLAR-GROUP/hpx/issues/3623>

¹⁴³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2836>

¹⁴³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4936>

¹⁴³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4935>

¹⁴³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4934>

1438 <https://github.com/STELLAR-GROUP/hpx/pull/4932>1439 <https://github.com/STELLAR-GROUP/hpx/pull/4931>1440 <https://github.com/STELLAR-GROUP/hpx/pull/4930>1441 <https://github.com/STELLAR-GROUP/hpx/pull/4929>1442 <https://github.com/STELLAR-GROUP/hpx/pull/4928>1443 <https://github.com/STELLAR-GROUP/hpx/pull/4926>1444 <https://github.com/STELLAR-GROUP/hpx/pull/4925>1445 <https://github.com/STELLAR-GROUP/hpx/pull/4923>1446 <https://github.com/STELLAR-GROUP/hpx/pull/4921>1447 <https://github.com/STELLAR-GROUP/hpx/pull/4920>1448 <https://github.com/STELLAR-GROUP/hpx/pull/4919>

- PR #4917¹⁴⁴⁹ - Move chrono functionality to hpx::chrono namespace
- PR #4916¹⁴⁵⁰ - HPX_HAVE_DEPRECATED_WARNINGS needs to be set even when disabled
- PR #4915¹⁴⁵¹ - Moving more action related files to actions modules
- PR #4914¹⁴⁵² - Add alias targets with namespaces used for exporting
- PR #4912¹⁴⁵³ - Aggregate initialize CPOs
- PR #4910¹⁴⁵⁴ - Explicitly specify hwloc root on Jenkins CSCS builds
- PR #4908¹⁴⁵⁵ - Fix algorithms documentation
- PR #4907¹⁴⁵⁶ - Remove HPX::hpx_no_wrap_main target
- PR #4906¹⁴⁵⁷ - Fixing unused variable warning
- PR #4905¹⁴⁵⁸ - Adding specializations for simple for_loops
- PR #4904¹⁴⁵⁹ - Update boost to 1.74.0 for the newest jenkins configs
- PR #4903¹⁴⁶⁰ - Hide GITHUB_TOKEN environment variables from environment variable output
- PR #4902¹⁴⁶¹ - Cancel previous pull requests builds before starting a new one with Jenkins
- PR #4901¹⁴⁶² - Update public API list with updated algorithms
- PR #4899¹⁴⁶³ - Suggested changes for HPX V1.5 release notes
- PR #4898¹⁴⁶⁴ - Minor tweak to hpx::equal implementation
- PR #4896¹⁴⁶⁵ - Making generate() and generate_n conforming to C++20
- PR #4895¹⁴⁶⁶ - Update apex tag
- PR #4894¹⁴⁶⁷ - Fix exception handling for tasks
- PR #4893¹⁴⁶⁸ - Remove last use of std::result_of, removed in C++20
- PR #4892¹⁴⁶⁹ - Adding replay_executor and replicate_executor
- PR #4889¹⁴⁷⁰ - Restore old behaviour of not requiring linking to hpx_wrap when HPX_WITH_DYNAMIC_HPX_MAIN=OFF
- PR #4887¹⁴⁷¹ - Making sure remotely thrown (non-hpx) exceptions are properly marshaled back to invocation

¹⁴⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4917>

¹⁴⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4916>

¹⁴⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4915>

¹⁴⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4914>

¹⁴⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4912>

¹⁴⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4910>

¹⁴⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4908>

¹⁴⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4907>

¹⁴⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4906>

¹⁴⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4905>

¹⁴⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4904>

¹⁴⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4903>

¹⁴⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/4902>

¹⁴⁶² <https://github.com/STELLAR-GROUP/hpx/pull/4901>

¹⁴⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/4899>

¹⁴⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4898>

¹⁴⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4896>

¹⁴⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4895>

¹⁴⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4894>

¹⁴⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4893>

¹⁴⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4892>

¹⁴⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4889>

¹⁴⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/4887>

site

- PR #4885¹⁴⁷² - Adapting hpx::find and friends to C++20
- PR #4884¹⁴⁷³ - Adapting mismatch to C++20
- PR #4883¹⁴⁷⁴ - Adapting hpx::equal to be conforming to C++20
- PR #4882¹⁴⁷⁵ - Fixing exception handling for hpx::copy and adding missing tests
- PR #4881¹⁴⁷⁶ - Adds different runtime exception when registering thread with the HPX runtime
- PR #4876¹⁴⁷⁷ - Adding example demonstrating how to disable thread stealing during the execution of parallel algorithms
- PR #4874¹⁴⁷⁸ - Adding non-policy tests to all_of, any_of, and none_of
- PR #4873¹⁴⁷⁹ - Set CUDA compute capability on rostam Jenkins builds
- PR #4872¹⁴⁸⁰ - Force partitioned vector scan tests to run serially
- PR #4870¹⁴⁸¹ - Making move conforming with C++20
- PR #4869¹⁴⁸² - Making destroy and destroy_n conforming to C++20
- PR #4868¹⁴⁸³ - Fix miscellaneous header problems
- PR #4867¹⁴⁸⁴ - Add CPOs for for_each
- PR #4865¹⁴⁸⁵ - Adapting count and count_if to be conforming to C++20
- PR #4864¹⁴⁸⁶ - Release notes 1.5.0
- PR #4863¹⁴⁸⁷ - adding libcds-hpx tag to prepare for hpx1.5 release
- PR #4862¹⁴⁸⁸ - Adding version specific deprecation options
- PR #4861¹⁴⁸⁹ - Limiting executor improvements
- PR #4860¹⁴⁹⁰ - Making fill and fill_n compatible with C++20
- PR #4859¹⁴⁹¹ - Adapting all_of, any_of, and none_of to C++20
- PR #4857¹⁴⁹² - Improve libCDS integration
- PR #4856¹⁴⁹³ - Correct typos in the documentation of the hpx performance counters

¹⁴⁷² <https://github.com/STELLAR-GROUP/hpx/pull/4885>

¹⁴⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/4884>

¹⁴⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4883>

¹⁴⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4882>

¹⁴⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4881>

¹⁴⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4876>

¹⁴⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4874>

¹⁴⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4873>

¹⁴⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4872>

¹⁴⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/4870>

¹⁴⁸² <https://github.com/STELLAR-GROUP/hpx/pull/4869>

¹⁴⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/4868>

¹⁴⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4867>

¹⁴⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4865>

¹⁴⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4864>

¹⁴⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4863>

¹⁴⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4862>

¹⁴⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4861>

¹⁴⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4860>

¹⁴⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4859>

¹⁴⁹² <https://github.com/STELLAR-GROUP/hpx/pull/4857>

¹⁴⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/4856>

- PR #4854¹⁴⁹⁴ - Removing obsolete code
- PR #4853¹⁴⁹⁵ - Adding test that derives component from two other components
- PR #4852¹⁴⁹⁶ - Fix mpi_ring test in distributed mode by ensuring all ranks run hpx_main
- PR #4851¹⁴⁹⁷ - Converting resiliency APIs to tag_invoke based CPOs
- PR #4849¹⁴⁹⁸ - Enable use of future_overhead test when DISTRIBUTED_RUNTIME is OFF
- PR #4847¹⁴⁹⁹ - Fixing ‘error prone’ constructs as reported by Codacy
- PR #4846¹⁵⁰⁰ - Disable Boost.Aasio concepts support
- PR #4845¹⁵⁰¹ - Fix PAPI counters
- PR #4843¹⁵⁰² - Remove dependency on various Boost headers
- PR #4841¹⁵⁰³ - Rearrange public API headers
- PR #4840¹⁵⁰⁴ - Fixing TSS problems during thread termination
- PR #4839¹⁵⁰⁵ - Fix async_cuda build problems when distributed runtime is disabled
- PR #4837¹⁵⁰⁶ - Restore compatibility for old (now deprecated) copy algorithms
- PR #4836¹⁵⁰⁷ - Adding CPOs for hpx::reduce
- PR #4835¹⁵⁰⁸ - Remove *using util::result_of* from namespace hpx
- PR #4834¹⁵⁰⁹ - Fixing the calculation of the number of idle cores and the corresponding idle masks
- PR #4833¹⁵¹⁰ - Allow thread function destructors to yield
- PR #4832¹⁵¹¹ - Fixing assertion in split_gids and memory leaks in 1d_stencil_7
- PR #4831¹⁵¹² - Making sure MPI_CXX_COMPILE_FLAGS is interpreted as a sequence of options
- PR #4830¹⁵¹³ - Update documentation on using HPX::wrap_main
- PR #4827¹⁵¹⁴ - Update clang-newest configuration to use clang 10
- PR #4826¹⁵¹⁵ - Add Jenkins configuration for rostam
- PR #4825¹⁵¹⁶ - Move all CUDA functionality to hpx::cuda::experimental namespace

¹⁴⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4854>

¹⁴⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4853>

¹⁴⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4852>

¹⁴⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4851>

¹⁴⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4849>

¹⁴⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4847>

¹⁵⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4846>

¹⁵⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/4845>

¹⁵⁰² <https://github.com/STELLAR-GROUP/hpx/pull/4843>

¹⁵⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/4841>

1504 <https://github.com/STELLAR-GROUP/hpx/pull/4840>1505 <https://github.com/STELLAR-GROUP/hpx/pull/4839>1506 <https://github.com/STELLAR-GROUP/hpx/pull/4837>1507 <https://github.com/STELLAR-GROUP/hpx/pull/4836>1508 <https://github.com/STELLAR-GROUP/hpx/pull/4835>1509 <https://github.com/STELLAR-GROUP/hpx/pull/4834>1510 <https://github.com/STELLAR-GROUP/hpx/pull/4833>1511 <https://github.com/STELLAR-GROUP/hpx/pull/4832>1512 <https://github.com/STELLAR-GROUP/hpx/pull/4831>1513 <https://github.com/STELLAR-GROUP/hpx/pull/4830>1514 <https://github.com/STELLAR-GROUP/hpx/pull/4827>1515 <https://github.com/STELLAR-GROUP/hpx/pull/4826>1516 <https://github.com/STELLAR-GROUP/hpx/pull/4825>

- PR #4824¹⁵¹⁷ - Add support for building master/release branches to Jenkins configuration
- PR #4821¹⁵¹⁸ - Implement customization point for hpx::copy and hpx::ranges::copy
- PR #4819¹⁵¹⁹ - Allow finding Boost components before finding HPX
- PR #4817¹⁵²⁰ - Adding range version of stable sort
- PR #4815¹⁵²¹ - Fix a wrong #ifdef for IO/TIMER pools causing build errors
- PR #4814¹⁵²² - Replace hpx::function_nonser with std::function in error module
- PR #4809¹⁵²³ - Foreach adapt
- PR #4808¹⁵²⁴ - Make internal algorithms functions const
- PR #4807¹⁵²⁵ - Add Jenkins configuration for running on Piz Daint
- PR #4806¹⁵²⁶ - Update documentation links to new domain name
- PR #4805¹⁵²⁷ - Applying changes that resolve time complexity issues in sort
- PR #4803¹⁵²⁸ - Adding implementation of stable_sort
- PR #4802¹⁵²⁹ - Fix datapar header paths
- PR #4801¹⁵³⁰ - Replace boost::shared_array<T> with std::shared_ptr<T[]> if supported
- PR #4799¹⁵³¹ - Fixing #include paths in compatibility headers
- PR #4798¹⁵³² - Include the main module header (fixes partially #4488)
- PR #4797¹⁵³³ - Change cmake targets
- PR #4794¹⁵³⁴ - Removing 128bit integer emulation
- PR #4793¹⁵³⁵ - Make sure global variable is handled properly
- PR #4792¹⁵³⁶ - Replace enable_if with **HPX_CONCEPT_REQUIREMENTS** and add is_sentinel_for constraint
- PR #4790¹⁵³⁷ - Move deprecation warnings from base template to template specializations for result_of etc. structs
- PR #4789¹⁵³⁸ - Fix hangs during assertion handling and distributed runtime construction
- PR #4788¹⁵³⁹ - Fixing inclusive transform scan algorithm to properly handle initial value

¹⁵¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4824>

¹⁵¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4821>

¹⁵¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4819>

¹⁵²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4817>

¹⁵²¹ <https://github.com/STELLAR-GROUP/hpx/pull/4815>

¹⁵²² <https://github.com/STELLAR-GROUP/hpx/pull/4814>

¹⁵²³ <https://github.com/STELLAR-GROUP/hpx/pull/4809>

¹⁵²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4808>

¹⁵²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4807>

¹⁵²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4806>

¹⁵²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4805>

¹⁵²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4803>

¹⁵²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4802>

¹⁵³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4801>

¹⁵³¹ <https://github.com/STELLAR-GROUP/hpx/pull/4799>

¹⁵³² <https://github.com/STELLAR-GROUP/hpx/pull/4798>

¹⁵³³ <https://github.com/STELLAR-GROUP/hpx/pull/4797>

¹⁵³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4794>

¹⁵³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4793>

¹⁵³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4792>

¹⁵³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4790>

¹⁵³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4789>

¹⁵³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4788>

- PR #4785¹⁵⁴⁰ - Fixing barrier test
- PR #4784¹⁵⁴¹ - Fixing deleter argument bindings in serialize_buffer
- PR #4783¹⁵⁴² - Add coveralls badge
- PR #4782¹⁵⁴³ - Make header tests parallel again
- PR #4780¹⁵⁴⁴ - Remove outdated comment about hpx::stop in documentation
- PR #4776¹⁵⁴⁵ - debug print improvements
- PR #4775¹⁵⁴⁶ - Checkpoint cleanup
- PR #4771¹⁵⁴⁷ - Fix compilation with HPX_WITH_NETWORKING=OFF
- PR #4767¹⁵⁴⁸ - Remove all force linking leftovers
- PR #4765¹⁵⁴⁹ - Fix 1d stencil index calculation
- PR #4764¹⁵⁵⁰ - Force some tests to run serially
- PR #4762¹⁵⁵¹ - Update pointees in compatibility headers
- PR #4761¹⁵⁵² - Fix running and building of execution module tests on CircleCI
- PR #4760¹⁵⁵³ - Storing hpx_options in global property to speed up summary report
- PR #4759¹⁵⁵⁴ - Reduce memory requirements for our main shared state
- PR #4757¹⁵⁵⁵ - Fix mimalloc linking on Windows
- PR #4756¹⁵⁵⁶ - Fix compilation issues
- PR #4753¹⁵⁵⁷ - Re-adding API functions that were lost during merges
- PR #4751¹⁵⁵⁸ - Revert “Create coverage reports and upload them to codecov.io”
- PR #4750¹⁵⁵⁹ - Fixing possible race condition during termination detection
- PR #4749¹⁵⁶⁰ - Deprecate result_of and friends
- PR #4748¹⁵⁶¹ - Create coverage reports and upload them to codecov.io
- PR #4747¹⁵⁶² - Changing #include for MPI parcelport

¹⁵⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4785>

¹⁵⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/4784>

¹⁵⁴² <https://github.com/STELLAR-GROUP/hpx/pull/4783>

¹⁵⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/4782>

¹⁵⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4780>

¹⁵⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4776>

¹⁵⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4775>

¹⁵⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4771>

¹⁵⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4767>

¹⁵⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4765>

¹⁵⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4764>

¹⁵⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4762>

¹⁵⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4761>

¹⁵⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4760>

¹⁵⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4759>

¹⁵⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4757>

¹⁵⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4756>

¹⁵⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4753>

¹⁵⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4751>

¹⁵⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4750>

¹⁵⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4749>

¹⁵⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/4748>

¹⁵⁶² <https://github.com/STELLAR-GROUP/hpx/pull/4747>

- PR #4745¹⁵⁶³ - Add *is_sentinel_for* trait implementation and test
- PR #4743¹⁵⁶⁴ - Fix init_globally example after runtime mode changes
- PR #4742¹⁵⁶⁵ - Update SUPPORT.md
- PR #4741¹⁵⁶⁶ - Fixing a warning generated for unity builds with msvc
- PR #4740¹⁵⁶⁷ - Rename local_lcoss and basic_execution modules
- PR #4739¹⁵⁶⁸ - Undeprecate a couple of hpx/modulename.hpp headers
- PR #4738¹⁵⁶⁹ - Conditionally test schedulers in thread_stacksize_current test
- PR #4734¹⁵⁷⁰ - Fixing a bunch of codacy warnings
- PR #4733¹⁵⁷¹ - Add experimental unity build option to CMake configuration
- PR #4730¹⁵⁷² - Fixing compilation problems with unordered map
- PR #4729¹⁵⁷³ - Fix APEX build
- PR #4727¹⁵⁷⁴ - Fix missing runtime includes for distributed runtime
- PR #4726¹⁵⁷⁵ - Add more API headers
- PR #4725¹⁵⁷⁶ - Add more compatibility headers for deprecated module headers
- PR #4724¹⁵⁷⁷ - Fix 4723
- PR #4721¹⁵⁷⁸ - Attempt to fixing migration tests
- PR #4717¹⁵⁷⁹ - Make the compatibility headers macro conditional
- PR #4716¹⁵⁸⁰ - Add hpx/runtime.hpp and hpx/distributed/runtime.hpp API headers
- PR #4714¹⁵⁸¹ - Add hpx/future.hpp header
- PR #4713¹⁵⁸² - Remove hpx/runtime/threads_fwd.hpp and hpx/util_fwd.hpp
- PR #4711¹⁵⁸³ - Make module deprecation warnings overridable
- PR #4710¹⁵⁸⁴ - Add compatibility headers and other fixes after module header renaming
- PR #4708¹⁵⁸⁵ - Add termination handler for parallel algorithms

¹⁵⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/4745>

¹⁵⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4743>

¹⁵⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4742>

¹⁵⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4741>

¹⁵⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4740>

¹⁵⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4739>

¹⁵⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4738>

¹⁵⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4734>

¹⁵⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/4733>

¹⁵⁷² <https://github.com/STELLAR-GROUP/hpx/pull/4730>

¹⁵⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/4729>

¹⁵⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4727>

¹⁵⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4726>

¹⁵⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4725>

¹⁵⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4724>

¹⁵⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4721>

¹⁵⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4717>

¹⁵⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4716>

¹⁵⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/4714>

¹⁵⁸² <https://github.com/STELLAR-GROUP/hpx/pull/4713>

¹⁵⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/4711>

¹⁵⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4710>

¹⁵⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4708>

- PR #4707¹⁵⁸⁶ - Use hpx::function_nonsr instead of std::function internally
- PR #4706¹⁵⁸⁷ - Move header file to module
- PR #4705¹⁵⁸⁸ - Fix incorrect behaviour of cmake-format check
- PR #4704¹⁵⁸⁹ - Fix resource tests
- PR #4701¹⁵⁹⁰ - Fix missing includes for future::then specializations
- PR #4700¹⁵⁹¹ - Removing obsolete memory component
- PR #4699¹⁵⁹² - Add short descriptions to modules missing documentation
- PR #4696¹⁵⁹³ - Rename generated modules headers
- PR #4693¹⁵⁹⁴ - Overhauling thread_mapper for public consumption
- PR #4688¹⁵⁹⁵ - Fix thread stack size handling
- PR #4687¹⁵⁹⁶ - Adding all_gather and fixing all_to_all
- PR #4684¹⁵⁹⁷ - Miscellaneous compilation fixes
- PR #4683¹⁵⁹⁸ - Fix HPX_WITH_DYNAMIC_HPX_MAIN=OFF
- PR #4682¹⁵⁹⁹ - Fix compilation of pack_traversal_rebind_container.hpp
- PR #4681¹⁶⁰⁰ - Add missing hpx/execution.hpp includes for future::then
- PR #4678¹⁶⁰¹ - Typeless communicator
- PR #4677¹⁶⁰² - Forcing registry option to be accepted without checks.
- PR #4676¹⁶⁰³ - Adding scatter_to/scatter_from collective operations
- PR #4673¹⁶⁰⁴ - Fix PAPI counters compilation
- PR #4671¹⁶⁰⁵ - Deprecate hpx::promise alias to hpx::lcos::promise
- PR #4670¹⁶⁰⁶ - Explicitly instantiate get_exception
- PR #4667¹⁶⁰⁷ - Add *stopValue* in *Sentinel* struct instead of *Iterator*
- PR #4666¹⁶⁰⁸ - Add release build on Windows to GitHub actions

1586 <https://github.com/STELLAR-GROUP/hpx/pull/4707>

1587 <https://github.com/STELLAR-GROUP/hpx/pull/4706>

1588 <https://github.com/STELLAR-GROUP/hpx/pull/4705>

1589 <https://github.com/STELLAR-GROUP/hpx/pull/4704>

1590 <https://github.com/STELLAR-GROUP/hpx/pull/4701>

1591 <https://github.com/STELLAR-GROUP/hpx/pull/4700>

1592 <https://github.com/STELLAR-GROUP/hpx/pull/4699>

1593 <https://github.com/STELLAR-GROUP/hpx/pull/4696>

1594 <https://github.com/STELLAR-GROUP/hpx/pull/4693>

1595 <https://github.com/STELLAR-GROUP/hpx/pull/4688>

1596 <https://github.com/STELLAR-GROUP/hpx/pull/4687>

1597 <https://github.com/STELLAR-GROUP/hpx/pull/4684>

1598 <https://github.com/STELLAR-GROUP/hpx/pull/4683>

1599 <https://github.com/STELLAR-GROUP/hpx/pull/4682>

1600 <https://github.com/STELLAR-GROUP/hpx/pull/4681>

1601 <https://github.com/STELLAR-GROUP/hpx/pull/4678>

1602 <https://github.com/STELLAR-GROUP/hpx/pull/4677>

1603 <https://github.com/STELLAR-GROUP/hpx/pull/4676>

1604 <https://github.com/STELLAR-GROUP/hpx/pull/4673>

1605 <https://github.com/STELLAR-GROUP/hpx/pull/4671>

1606 <https://github.com/STELLAR-GROUP/hpx/pull/4670>

1607 <https://github.com/STELLAR-GROUP/hpx/pull/4667>

1608 <https://github.com/STELLAR-GROUP/hpx/pull/4666>

- PR #4664¹⁶⁰⁹ - Creating itt_notify module.
- PR #4663¹⁶¹⁰ - Mpi fixes
- PR #4659¹⁶¹¹ - Making sure declarations match definitions in register_locks implementation
- PR #4655¹⁶¹² - Fixing task annotations for actions
- PR #4653¹⁶¹³ - Making sure APEX is linked into every application, if needed
- PR #4651¹⁶¹⁴ - Update get_function_annotation.hpp
- PR #4646¹⁶¹⁵ - Runtime type
- PR #4645¹⁶¹⁶ - Add a few more API headers
- PR #4644¹⁶¹⁷ - Fixing support for mpirun (and similar)
- PR #4643¹⁶¹⁸ - Fixing the fix for get_idle_core_count() API
- PR #4638¹⁶¹⁹ - Remove HPX_API_EXPORT missed in previous cleanup
- PR #4636¹⁶²⁰ - Adding C++20 barrier
- PR #4635¹⁶²¹ - Adding C++20 latch API
- PR #4634¹⁶²² - Adding C++20 counting semaphore API
- PR #4633¹⁶²³ - Unify execution parameters customization points
- PR #4632¹⁶²⁴ - Adding missing bulk_sync_execute wrapper to example executor
- PR #4631¹⁶²⁵ - Updates to documentation; grammar edits.
- PR #4630¹⁶²⁶ - Updates to documentation; moved hyperlink
- PR #4624¹⁶²⁷ - Export set_self_ptr in thread_data.hpp instead of with forward declarations where used
- PR #4623¹⁶²⁸ - Clean up export macros
- PR #4621¹⁶²⁹ - Trigger an error for older boost versions on power architectures
- PR #4617¹⁶³⁰ - Ignore user-set compatibility header options if the module does not have compatibility headers
- PR #4616¹⁶³¹ - Fix cmake-format warning

¹⁶⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4664>

¹⁶¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4663>

¹⁶¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4659>

¹⁶¹² <https://github.com/STELLAR-GROUP/hpx/pull/4655>

¹⁶¹³ <https://github.com/STELLAR-GROUP/hpx/pull/4653>

¹⁶¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4651>

¹⁶¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4646>

¹⁶¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4645>

¹⁶¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4644>

1618 <https://github.com/STELLAR-GROUP/hpx/pull/4643>1619 <https://github.com/STELLAR-GROUP/hpx/pull/4638>1620 <https://github.com/STELLAR-GROUP/hpx/pull/4636>1621 <https://github.com/STELLAR-GROUP/hpx/pull/4635>1622 <https://github.com/STELLAR-GROUP/hpx/pull/4634>1623 <https://github.com/STELLAR-GROUP/hpx/pull/4633>1624 <https://github.com/STELLAR-GROUP/hpx/pull/4632>1625 <https://github.com/STELLAR-GROUP/hpx/pull/4631>1626 <https://github.com/STELLAR-GROUP/hpx/pull/4630>1627 <https://github.com/STELLAR-GROUP/hpx/pull/4624>1628 <https://github.com/STELLAR-GROUP/hpx/pull/4623>1629 <https://github.com/STELLAR-GROUP/hpx/pull/4621>1630 <https://github.com/STELLAR-GROUP/hpx/pull/4617>1631 <https://github.com/STELLAR-GROUP/hpx/pull/4616>

- PR #4615¹⁶³² - Add handler for serializing custom exceptions
- PR #4614¹⁶³³ - Fix error message when HPX_IGNORE_CMAKE_BUILD_TYPE_COMPATIBILITY=OFF
- PR #4613¹⁶³⁴ - Make partitioner constructor private
- PR #4611¹⁶³⁵ - Making auto_chunk_size execute the given function using the given executor
- PR #4610¹⁶³⁶ - Making sure the thread-local lock registration data is moving to the core the suspended HPX thread is resumed on
- PR #4609¹⁶³⁷ - Adding an API function that exposes the number of idle cores
- PR #4608¹⁶³⁸ - Fixing moodycamel namespace
- PR #4607¹⁶³⁹ - Moving winsocket initialization to core library
- PR #4606¹⁶⁴⁰ - Local runtime module etc.
- PR #4604¹⁶⁴¹ - Add config_registry module
- PR #4603¹⁶⁴² - Deal with distributed modules in their respective CMakeLists.txt
- PR #4602¹⁶⁴³ - Small module fixes
- PR #4598¹⁶⁴⁴ - Making sure current_executor and service_executor functions are linked into the core library
- PR #4597¹⁶⁴⁵ - Adding broadcast_to/broadcast_from to collectives module
- PR #4596¹⁶⁴⁶ - Fix performance regression in block_executor
- PR #4595¹⁶⁴⁷ - Making sure main.cpp is built as a library if HPX_WITH_DYNAMIC_MAIN=OFF
- PR #4592¹⁶⁴⁸ - Futures module
- PR #4591¹⁶⁴⁹ - Adapting co_await support for C++20
- PR #4590¹⁶⁵⁰ - Adding missing exception test for for_loop()
- PR #4587¹⁶⁵¹ - Move traits headers to hpx/modulename/traits directory
- PR #4586¹⁶⁵² - Remove Travis CI config
- PR #4585¹⁶⁵³ - Update macOS test blacklist
- PR #4584¹⁶⁵⁴ - Attempting to fix missing symbols in stack trace

¹⁶³² <https://github.com/STELLAR-GROUP/hpx/pull/4615>

¹⁶³³ <https://github.com/STELLAR-GROUP/hpx/pull/4614>

¹⁶³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4613>

¹⁶³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4611>

¹⁶³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4610>

¹⁶³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4609>

¹⁶³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4608>

¹⁶³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4607>

¹⁶⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4606>

¹⁶⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/4604>

¹⁶⁴² <https://github.com/STELLAR-GROUP/hpx/pull/4603>

¹⁶⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/4602>

¹⁶⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4598>

¹⁶⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4597>

¹⁶⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4596>

¹⁶⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4595>

¹⁶⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4592>

¹⁶⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4591>

¹⁶⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4590>

¹⁶⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4587>

¹⁶⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4586>

¹⁶⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4585>

¹⁶⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4584>

- PR #4583¹⁶⁵⁵ - Fixing bad static_cast
- PR #4582¹⁶⁵⁶ - Changing download url for Windows prerequisites to circumvent bandwidth limitations
- PR #4581¹⁶⁵⁷ - Adding missing using placeholder::_X
- PR #4579¹⁶⁵⁸ - Move get_stack_size_name and related functions
- PR #4575¹⁶⁵⁹ - Excluding unconditional definition of class backtrace from global header
- PR #4574¹⁶⁶⁰ - Changing return type of hardware_concurrency() to unsigned int
- PR #4570¹⁶⁶¹ - Move tests to modules
- PR #4564¹⁶⁶² - Reshuffle internal targets and add HPX::hpx_no_wrap_main target
- PR #4563¹⁶⁶³ - fix CMake option typo
- PR #4562¹⁶⁶⁴ - Unregister lock earlier to avoid holding it while suspending
- PR #4561¹⁶⁶⁵ - Adding test macros supporting custom output stream
- PR #4560¹⁶⁶⁶ - Making sure hash_any::operator()() is linked into core library
- PR #4559¹⁶⁶⁷ - Fixing compilation if HPX_WITH_THREAD_BACKTRACE_ON_SUSPENSION=On
- PR #4557¹⁶⁶⁸ - Improve spinlock implementation to perform better in high-contention situations
- PR #4553¹⁶⁶⁹ - Fix a runtime_ptr problem at shutdown when apex is enabled
- PR #4552¹⁶⁷⁰ - Add configuration option for making exceptions less noisy
- PR #4551¹⁶⁷¹ - Clean up thread creation parameters
- PR #4549¹⁶⁷² - Test FetchContent build on GitHub actions
- PR #4548¹⁶⁷³ - Fix stack size
- PR #4545¹⁶⁷⁴ - Fix header tests
- PR #4544¹⁶⁷⁵ - Fix a typo in sanitizer build
- PR #4541¹⁶⁷⁶ - Add API to check if a thread pool exists
- PR #4540¹⁶⁷⁷ - Making sure MPI support is enabled if MPI futures are used but networking is disabled

¹⁶⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4583>

¹⁶⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4582>

¹⁶⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4581>

¹⁶⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4579>

¹⁶⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4575>

¹⁶⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4574>

¹⁶⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/4570>

¹⁶⁶² <https://github.com/STELLAR-GROUP/hpx/pull/4564>

¹⁶⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/4563>

¹⁶⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4562>

¹⁶⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4561>

¹⁶⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4560>

¹⁶⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4559>

¹⁶⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4557>

¹⁶⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4553>

¹⁶⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4552>

¹⁶⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/4551>

¹⁶⁷² <https://github.com/STELLAR-GROUP/hpx/pull/4549>

¹⁶⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/4548>

¹⁶⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4545>

¹⁶⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4544>

¹⁶⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4541>

¹⁶⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4540>

- PR #4538¹⁶⁷⁸ - Move channel documentation examples to examples directory
- PR #4536¹⁶⁷⁹ - Add generic allocator for execution policies
- PR #4534¹⁶⁸⁰ - Enable compatibility headers for thread_executors module
- PR #4532¹⁶⁸¹ - Fixing broken url in README.rst
- PR #4531¹⁶⁸² - Update scripts
- PR #4530¹⁶⁸³ - Make sure module API docs show up in correct order
- PR #4529¹⁶⁸⁴ - Adding missing template code to module creation script
- PR #4528¹⁶⁸⁵ - Make sure version module uses HPX's binary dir, not the parent's
- PR #4527¹⁶⁸⁶ - Creating actions_base and actions module
- PR #4526¹⁶⁸⁷ - Shared state for cv
- PR #4525¹⁶⁸⁸ - Changing sub-name sequencing for experimental namespace
- PR #4524¹⁶⁸⁹ - Add API guarantee notes to API reference documentation
- PR #4522¹⁶⁹⁰ - Enable and fix deprecation warnings in execution module
- PR #4521¹⁶⁹¹ - Moves more miscellaneous files to modules
- PR #4520¹⁶⁹² - Skip execution customization points when executor is known
- PR #4518¹⁶⁹³ - Module distributed lcos
- PR #4516¹⁶⁹⁴ - Fix various builds
- PR #4515¹⁶⁹⁵ - Replace backslashes by slashes in windows paths
- PR #4514¹⁶⁹⁶ - Adding polymorphic_executor
- PR #4512¹⁶⁹⁷ - Adding C++20 jthread and stop_token
- PR #4510¹⁶⁹⁸ - Attempt to fix APEX linking in external packages again
- PR #4508¹⁶⁹⁹ - Only test pull requests (not all branches) with GitHub actions
- PR #4505¹⁷⁰⁰ - Fix duplicate linking in tests (ODR violations)

¹⁶⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4538>

¹⁶⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4536>

¹⁶⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4534>

¹⁶⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/4532>

¹⁶⁸² <https://github.com/STELLAR-GROUP/hpx/pull/4531>

¹⁶⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/4530>

¹⁶⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4529>

¹⁶⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4528>

¹⁶⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4527>

¹⁶⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4526>

¹⁶⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4525>

¹⁶⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4524>

¹⁶⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4522>

¹⁶⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4521>

1692 <https://github.com/STELLAR-GROUP/hpx/pull/4520>1693 <https://github.com/STELLAR-GROUP/hpx/pull/4518>1694 <https://github.com/STELLAR-GROUP/hpx/pull/4516>1695 <https://github.com/STELLAR-GROUP/hpx/pull/4515>1696 <https://github.com/STELLAR-GROUP/hpx/pull/4514>1697 <https://github.com/STELLAR-GROUP/hpx/pull/4512>1698 <https://github.com/STELLAR-GROUP/hpx/pull/4510>1699 <https://github.com/STELLAR-GROUP/hpx/pull/4508>1700 <https://github.com/STELLAR-GROUP/hpx/pull/4505>

- PR #4504¹⁷⁰¹ - Fix C++ standard handling
- PR #4503¹⁷⁰² - Add CMakeLists file check
- PR #4500¹⁷⁰³ - Fix .clang-format version requirement comment
- PR #4499¹⁷⁰⁴ - Attempting to fix hpx_init linking on macOS
- PR #4498¹⁷⁰⁵ - Fix compatibility of *pool_executor*
- PR #4496¹⁷⁰⁶ - Removing superfluous SPDX tags
- PR #4494¹⁷⁰⁷ - Module executors
- PR #4493¹⁷⁰⁸ - Pack traversal module
- PR #4492¹⁷⁰⁹ - Update copyright year in documentation
- PR #4491¹⁷¹⁰ - Add missing current_executor header
- PR #4490¹⁷¹¹ - Update GitHub actions configs
- PR #4487¹⁷¹² - Properly dispatch exceptions thrown from hpx_main to be rethrown from hpx::init/hpx::stop
- PR #4486¹⁷¹³ - Fixing an initialization order problem
- PR #4485¹⁷¹⁴ - Move miscellaneous files to their rightful modules
- PR #4483¹⁷¹⁵ - Clean up imported CMake target naming
- PR #4481¹⁷¹⁶ - Add vcpkg installation instructions
- PR #4479¹⁷¹⁷ - Add hints to allow to specify MIMALLOC_ROOT
- PR #4478¹⁷¹⁸ - Async modules
- PR #4475¹⁷¹⁹ - Fix rp init changes
- PR #4474¹⁷²⁰ - Use #pragma once in headers
- PR #4472¹⁷²¹ - Add more descriptive error message when using x86 coroutines on non-x86 platforms
- PR #4467¹⁷²² - Add mimalloc find cmake script
- PR #4465¹⁷²³ - Add thread_executors module

¹⁷⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/4504>

¹⁷⁰² <https://github.com/STELLAR-GROUP/hpx/pull/4503>

¹⁷⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/4500>

¹⁷⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4499>

¹⁷⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4498>

¹⁷⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4496>

¹⁷⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4494>

¹⁷⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4493>

¹⁷⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4492>

¹⁷¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4491>

¹⁷¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4490>

¹⁷¹² <https://github.com/STELLAR-GROUP/hpx/pull/4487>

¹⁷¹³ <https://github.com/STELLAR-GROUP/hpx/pull/4486>

¹⁷¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4485>

¹⁷¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4483>

¹⁷¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4481>

¹⁷¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4479>

¹⁷¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4478>

¹⁷¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4475>

¹⁷²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4474>

¹⁷²¹ <https://github.com/STELLAR-GROUP/hpx/pull/4472>

¹⁷²² <https://github.com/STELLAR-GROUP/hpx/pull/4467>

¹⁷²³ <https://github.com/STELLAR-GROUP/hpx/pull/4465>

- PR #4464¹⁷²⁴ - Include module
- PR #4462¹⁷²⁵ - Merge hpx_init and hpx_wrap into one static library
- PR #4461¹⁷²⁶ - Making thread_data test more realistic
- PR #4460¹⁷²⁷ - Suppress MPI warnings in version.cpp
- PR #4459¹⁷²⁸ - Make sure pkgconfig applications link with hpx_init
- PR #4458¹⁷²⁹ - Added example demonstrating how to create and use a wrapping executor
- PR #4457¹⁷³⁰ - Fixing execution of thread exit functions
- PR #4456¹⁷³¹ - Move backtrace files to debugging module
- PR #4455¹⁷³² - Move deadlock_detection and maintain_queue_wait_times source files into schedulers module
- PR #4450¹⁷³³ - Fixing compilation with std::filesystem enabled
- PR #4449¹⁷³⁴ - Fixing build system to actually build variant test
- PR #4447¹⁷³⁵ - This fixes an obsolete #include
- PR #4446¹⁷³⁶ - Resume tasks where they were suspended
- PR #4444¹⁷³⁷ - Minor CUDA fixes
- PR #4443¹⁷³⁸ - Add missing tests to CircleCI config
- PR #4442¹⁷³⁹ - Adding a tag to all auto-generated files allowing for tools to visually distinguish those
- PR #4441¹⁷⁴⁰ - Adding performance counter type information
- PR #4440¹⁷⁴¹ - Fixing MSVC build
- PR #4439¹⁷⁴² - Link HPX::plugin and component privately in hpx_setup_target
- PR #4437¹⁷⁴³ - Adding a test that verifies the problem can be solved using a trait specialization
- PR #4434¹⁷⁴⁴ - Clean up Boost dependencies and copy string algorithms to new module
- PR #4433¹⁷⁴⁵ - Fixing compilation issues (!) if MPI parcelport is enabled
- PR #4431¹⁷⁴⁶ - Ignore warnings about name mangling changing

¹⁷²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4464>

¹⁷²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4462>

¹⁷²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4461>

¹⁷²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4460>

¹⁷²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4459>

¹⁷²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4458>

¹⁷³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4457>

¹⁷³¹ <https://github.com/STELLAR-GROUP/hpx/pull/4456>

¹⁷³² <https://github.com/STELLAR-GROUP/hpx/pull/4455>

¹⁷³³ <https://github.com/STELLAR-GROUP/hpx/pull/4450>

¹⁷³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4449>

¹⁷³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4447>

¹⁷³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4446>

¹⁷³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4444>

¹⁷³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4443>

¹⁷³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4442>

¹⁷⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4441>

¹⁷⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/4440>

¹⁷⁴² <https://github.com/STELLAR-GROUP/hpx/pull/4439>

¹⁷⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/4437>

¹⁷⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4434>

¹⁷⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4433>

¹⁷⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4431>

- PR #4430¹⁷⁴⁷ - Add performance_counters module
- PR #4428¹⁷⁴⁸ - Don't add compatibility headers to module API reference
- PR #4426¹⁷⁴⁹ - Add currently failing tests on GitHub actions to blacklist
- PR #4425¹⁷⁵⁰ - Clean up and correct minimum required versions
- PR #4424¹⁷⁵¹ - Making sure hpx.lock_detection=0 works as advertised
- PR #4421¹⁷⁵² - Making sure interval time stops underlying timer thread on termination
- PR #4417¹⁷⁵³ - Adding serialization support for std::variant (if available) and std::tuple
- PR #4415¹⁷⁵⁴ - Partially reverting changes applied by PR 4373
- PR #4414¹⁷⁵⁵ - Added documentation for the compiler-wrapper script hpxcxx.in in creating_hpx_projects.rst
- PR #4413¹⁷⁵⁶ - Merging from V1.4.1 release
- PR #4412¹⁷⁵⁷ - Making sure to issue a warning if a file specified using --hpx:options-file is not found
- PR #4411¹⁷⁵⁸ - Make test specific to HPX_WITH_SHARED_PRIORITY_SCHEDULER
- PR #4407¹⁷⁵⁹ - Adding minimal MPI executor
- PR #4405¹⁷⁶⁰ - Fix cross pool injection test, use default scheduler as fallback
- PR #4404¹⁷⁶¹ - Fix a race condition and clean-up usage of scheduler mode
- PR #4399¹⁷⁶² - Add more threading modules
- PR #4398¹⁷⁶³ - Add CODEOWNERS file
- PR #4395¹⁷⁶⁴ - Adding a parameter to auto_chunk_size allowing to control the amount of iterations to measure
- PR #4393¹⁷⁶⁵ - Use appropriate cache-line size defaults for different platforms
- PR #4391¹⁷⁶⁶ - Fixing use of allocator for C++20
- PR #4390¹⁷⁶⁷ - Making --hpx:help behavior consistent
- PR #4388¹⁷⁶⁸ - Change the resource partitioner initialization
- PR #4387¹⁷⁶⁹ - Fix roll_release.sh

¹⁷⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4430>

¹⁷⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4428>

¹⁷⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4426>

¹⁷⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4425>

¹⁷⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4424>

¹⁷⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4421>

¹⁷⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4417>

¹⁷⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4415>

¹⁷⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4414>

¹⁷⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4413>

¹⁷⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4412>

¹⁷⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4411>

¹⁷⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4407>

¹⁷⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4405>

¹⁷⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/4404>

¹⁷⁶² <https://github.com/STELLAR-GROUP/hpx/pull/4399>

¹⁷⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/4398>

¹⁷⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4395>

¹⁷⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4393>

¹⁷⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4391>

¹⁷⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4390>

¹⁷⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4388>

¹⁷⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4387>

- PR #4386¹⁷⁷⁰ - Add warning messages for using thread binding options on macOS
- PR #4385¹⁷⁷¹ - Cuda futures
- PR #4384¹⁷⁷² - Make enabling dynamic hpx_main on non-Linux systems a configuration error
- PR #4383¹⁷⁷³ - Use configure_file for HPXCacheVariables.cmake
- PR #4382¹⁷⁷⁴ - Update spellchecking whitelist and fix more typos
- PR #4380¹⁷⁷⁵ - Add a helper function to get a future from a cuda stream
- PR #4379¹⁷⁷⁶ - Add Windows and macOS CI with GitHub actions
- PR #4378¹⁷⁷⁷ - Change C++ standard handling
- PR #4377¹⁷⁷⁸ - Remove Python scripts
- PR #4374¹⁷⁷⁹ - Adding overload for *hpx::init/hpx::start* for use with resource partitioner
- PR #4373¹⁷⁸⁰ - Adding test that verifies for 4369 to be fixed
- PR #4372¹⁷⁸¹ - Another attempt at fixing the integral mismatch and conversion warnings
- PR #4370¹⁷⁸² - Doc updates quick start
- PR #4368¹⁷⁸³ - Add a whitelist of words for weird spelling suggestions
- PR #4366¹⁷⁸⁴ - Suppress or fix clang-tidy-9 warnings
- PR #4365¹⁷⁸⁵ - Removing more Boost dependencies
- PR #4363¹⁷⁸⁶ - Update clang-format config file for version 9
- PR #4362¹⁷⁸⁷ - Fix indices typo
- PR #4361¹⁷⁸⁸ - Boost cleanup
- PR #4360¹⁷⁸⁹ - Move plugins
- PR #4358¹⁷⁹⁰ - Doc updates; generating documentation. Will likely need heavy editing.
- PR #4356¹⁷⁹¹ - Remove some minor unused and unnecessary Boost includes
- PR #4355¹⁷⁹² - Fix spellcheck step in CircleCI config

¹⁷⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4386>

¹⁷⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/4385>

¹⁷⁷² <https://github.com/STELLAR-GROUP/hpx/pull/4384>

¹⁷⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/4383>

¹⁷⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4382>

¹⁷⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4380>

¹⁷⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4379>

¹⁷⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4378>

¹⁷⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4377>

¹⁷⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4374>

¹⁷⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4373>

¹⁷⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/4372>

¹⁷⁸² <https://github.com/STELLAR-GROUP/hpx/pull/4370>

¹⁷⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/4368>

1784 <https://github.com/STELLAR-GROUP/hpx/pull/4366>1785 <https://github.com/STELLAR-GROUP/hpx/pull/4365>1786 <https://github.com/STELLAR-GROUP/hpx/pull/4363>1787 <https://github.com/STELLAR-GROUP/hpx/pull/4362>1788 <https://github.com/STELLAR-GROUP/hpx/pull/4361>1789 <https://github.com/STELLAR-GROUP/hpx/pull/4360>1790 <https://github.com/STELLAR-GROUP/hpx/pull/4358>1791 <https://github.com/STELLAR-GROUP/hpx/pull/4356>1792 <https://github.com/STELLAR-GROUP/hpx/pull/4355>

- PR #4354¹⁷⁹³ - Lightweight utility to hold a pack as members
- PR #4352¹⁷⁹⁴ - Minor fixes to the C++ standard detection for MSVC
- PR #4351¹⁷⁹⁵ - Move generated documentation to hpx-docs repo
- PR #4347¹⁷⁹⁶ - Add cmake policy - CMP0074
- PR #4346¹⁷⁹⁷ - Remove file committed by mistake
- PR #4342¹⁷⁹⁸ - Remove HCC and SYCL options from CMakeLists.txt
- PR #4341¹⁷⁹⁹ - Fix launch process test with APEX enabled
- PR #4340¹⁸⁰⁰ - Testing Cirrus CI
- PR #4339¹⁸⁰¹ - Post 1.4.0 updates
- PR #4338¹⁸⁰² - Spelling corrections and CircleCI spell check
- PR #4333¹⁸⁰³ - Flatten bound callables
- PR #4332¹⁸⁰⁴ - This is a collection of mostly minor (cleanup) fixes
- PR #4331¹⁸⁰⁵ - This adds the missing tests for async_colocated and async_continue_colocated
- PR #4330¹⁸⁰⁶ - Remove HPX.Compute host default_executor
- PR #4328¹⁸⁰⁷ - Generate global header for basic_execution module
- PR #4327¹⁸⁰⁸ - Use INTERNAL_FLAGS option for all examples and components
- PR #4326¹⁸⁰⁹ - Usage of temporary allocator in assignment operator of compute::vector
- PR #4325¹⁸¹⁰ - Use hpx::threads::get_cache_line_size in prefetching.hpp
- PR #4324¹⁸¹¹ - Enable compatibility headers option for execution module
- PR #4316¹⁸¹² - Add clang format indentppdirectives
- PR #4313¹⁸¹³ - Introduce index_pack alias to pack of size_t
- PR #4312¹⁸¹⁴ - Fixing compatibility header for pack.hpp
- PR #4311¹⁸¹⁵ - Dataflow annotations for APEX

¹⁷⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/4354>

¹⁷⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4352>

¹⁷⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4351>

¹⁷⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4347>

¹⁷⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4346>

¹⁷⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4342>

¹⁷⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4341>

¹⁸⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4340>

¹⁸⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/4339>

¹⁸⁰² <https://github.com/STELLAR-GROUP/hpx/pull/4338>

¹⁸⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/4333>

¹⁸⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4332>

¹⁸⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4331>

¹⁸⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4330>

¹⁸⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4328>

¹⁸⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4327>

¹⁸⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4326>

¹⁸¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4325>

¹⁸¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4324>

¹⁸¹² <https://github.com/STELLAR-GROUP/hpx/pull/4316>

¹⁸¹³ <https://github.com/STELLAR-GROUP/hpx/pull/4313>

¹⁸¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4312>

¹⁸¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4311>

- PR #4309¹⁸¹⁶ - Update launching_and_configuring_hpx_applications.rst
- PR #4306¹⁸¹⁷ - Fix schedule hint not being taken from executor
- PR #4305¹⁸¹⁸ - Implementing *hpx::functional::tag_invoke*
- PR #4304¹⁸¹⁹ - Improve pack support utilities
- PR #4303¹⁸²⁰ - Remove errors module dependency on datastructures
- PR #4301¹⁸²¹ - Clean up thread executors
- PR #4294¹⁸²² - Logging revamp
- PR #4292¹⁸²³ - Remove SPDX tag from Boost License file to allow for github to recognize it
- PR #4291¹⁸²⁴ - Add format support for std::tm
- PR #4290¹⁸²⁵ - Simplify compatible tuples check
- PR #4288¹⁸²⁶ - A lightweight take on boost::lexical_cast
- PR #4287¹⁸²⁷ - Forking boost::lexical_cast as a new module
- PR #4277¹⁸²⁸ - MPI_futures
- PR #4270¹⁸²⁹ - Refactor future implementation
- PR #4265¹⁸³⁰ - Threading module
- PR #4259¹⁸³¹ - Module naming base
- PR #4251¹⁸³² - Local workrequesting scheduler
- PR #4250¹⁸³³ - Inline execution of scoped tasks, if possible
- PR #4247¹⁸³⁴ - Add execution in module headers
- PR #4246¹⁸³⁵ - Expose CMake targets officially
- PR #4239¹⁸³⁶ - Doc updates miscellaneous (partially completed during Google Season of Docs)
- PR #4233¹⁸³⁷ - Remove project() from modules + fix CMAKE_SOURCE_DIR issue
- PR #4231¹⁸³⁸ - Module local lcos

¹⁸¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4309>

¹⁸¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4306>

¹⁸¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4305>

¹⁸¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4304>

¹⁸²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4303>

¹⁸²¹ <https://github.com/STELLAR-GROUP/hpx/pull/4301>

¹⁸²² <https://github.com/STELLAR-GROUP/hpx/pull/4294>

¹⁸²³ <https://github.com/STELLAR-GROUP/hpx/pull/4292>

¹⁸²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4291>

¹⁸²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4290>

¹⁸²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4288>

¹⁸²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4287>

¹⁸²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4277>

¹⁸²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4270>

¹⁸³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4265>

¹⁸³¹ <https://github.com/STELLAR-GROUP/hpx/pull/4259>

¹⁸³² <https://github.com/STELLAR-GROUP/hpx/pull/4251>

¹⁸³³ <https://github.com/STELLAR-GROUP/hpx/pull/4250>

¹⁸³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4247>

¹⁸³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4246>

¹⁸³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4239>

¹⁸³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4233>

¹⁸³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4231>

- PR #4207¹⁸³⁹ - Command line handling module
- PR #4206¹⁸⁴⁰ - Runtime configuration module
- PR #4141¹⁸⁴¹ - Doc updates examples local to remote (partially completed during Google Season of Docs)
- PR #4091¹⁸⁴² - Split runtime into local and distributed parts
- PR #4017¹⁸⁴³ - Require C++14

2.10.7 HPX V1.4.1 (Feb 12, 2020)

General changes

This is a bugfix release. It contains the following changes:

- Fix compilation issues on Windows, macOS, FreeBSD, and with gcc 10
- Install missing .pdb files on Windows
- Allow running tests using an installed version of *HPX*
- Skip MPI finalization if HPX has not initialized MPI
- Give a hard error when attempting to use IO counters on Windows

Closed issues

- Issue #4320¹⁸⁴⁴ - HPX 1.4.0 does not compile with gcc 10
- Issue #4336¹⁸⁴⁵ - Building HPX 1.4.0 with IO Counters breaks (Windows)
- Issue #4334¹⁸⁴⁶ - HPX Debug and RelWithDebInfo builds on Windows not installing .pdb files
- Issue #4322¹⁸⁴⁷ - Undefine VT1 and VT2 after boost includes
- Issue #4314¹⁸⁴⁸ - Compile error on 1.4.0
- Issue #4307¹⁸⁴⁹ - ld: error: duplicate symbol: freebsd_environ

¹⁸³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4207>

¹⁸⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4206>

¹⁸⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/4141>

¹⁸⁴² <https://github.com/STELLAR-GROUP/hpx/pull/4091>

¹⁸⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/4017>

¹⁸⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4320>

¹⁸⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4336>

¹⁸⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4334>

¹⁸⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4322>

¹⁸⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4314>

¹⁸⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4307>

Closed pull requests

- PR #4376¹⁸⁵⁰ - Attempt to fix some test build errors on Windows
- PR #4357¹⁸⁵¹ - Adding missing #includes to fix gcc V10 linker problems
- PR #4353¹⁸⁵² - Skip MPI_Finalize if MPI_Init is not called from HPX
- PR #4343¹⁸⁵³ - Give a hard error if IO counters are enabled on non-Linux systems
- PR #4337¹⁸⁵⁴ - Installing pdb files on Windows
- PR #4335¹⁸⁵⁵ - Adding capability to buildsystem to use an installed version of HPX
- PR #4315¹⁸⁵⁶ - Forcing exported symbols from composable_guard to be linked into core library
- PR #4310¹⁸⁵⁷ - Remove environment handling from exception.cpp

2.10.8 HPX V1.4.0 (January 15, 2020)

General changes

- We have added the collectives all_to_all and all_reduce.
- We have added APIs for resiliency, which allows replication and replay for failed tasks. See the *documentation* for more details.
- Components can now be checkpointed.
- Performance improvements to schedulers and coroutines. A significant change is the addition of stackless coroutines. These are to be used for tasks that do not need to be suspended and can reduce overheads noticeably in applications with short tasks. A stackless coroutine can be created with the new stack size `thread_stacksize_nostack`.
- We have added an implementation of `unique_any`, which is a non-copyable version of `any`.
- The `shared_priority_queue_scheduler` has been improved. It now has lower overheads than the default scheduler in many situations. Unlike the default scheduler it fully supports NUMA scheduling hints. Enable it with the command line option `--hpx:queuing=shared-priority`. This scheduler should still be considered experimental, but its use is encouraged in real applications to help us make it production ready.
- We have added the performance counters `background-receive-duration` and `background-receive-overhead` for inspecting the time and overhead spent on receiving parcels in the background.
- Compilation time has been further improved when `HPX_WITH_NETWORKING=OFF`.
- We no longer require compiled Boost dependencies in certain configurations. This requires at least Boost 1.70, compiling on x86 with GCC 9, clang (libc++) 9, or VS2019 in C++17 mode. The dependency on Boost.Filesystem can explicitly be turned on with `HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY=ON` (it is off by default if the standard library supports `std::filesystem`). Boost.ProgramOptions has been copied into the HPX repository. We have a compatibility layer for users who must explicitly use

¹⁸⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4376>

¹⁸⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4357>

¹⁸⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4353>

¹⁸⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4343>

¹⁸⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4337>

¹⁸⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4335>

¹⁸⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4315>

¹⁸⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4310>

Boost.ProgramOptions instead of the ProgramOptions provided by HPX. To remove the dependency `HPX_PROGRAM_OPTIONS_WITH_BOOST_PROGRAM_OPTIONS_COMPATIBILITY` must be explicitly set to OFF. This option will be removed in a future release. We have also removed several other header-only dependencies on Boost.

- It is now possible to use the process affinity mask set by tools like numactl and various batch environments with the command line option `--hpx:use-process-mask`. Enabling this option implies `--hpx:ignore-batch-env`.
- It is now possible to create standalone thread pools without starting the runtime. See the `standalone_thread_pool_executor.cpp` test in the execution module for an example.
- Tasks annotated with `hpx::util::annotated_function` now have their correct name when using APEX to generate OTF2 files.
- Cloning of APEX was defective in previous releases (it required manual intervention to check out the correct tag or branch). This has been fixed.
- The option `HPX_WITH_MORE_THAN_64_THREADS` is now ignored and will be removed in a future release. The value is instead derived directly from `HPX_WITH_MAX_CPU_COUNT` option.
- We have deprecated compiling in C++11 mode. The next release will require a C++14 capable compiler.
- We have deprecated support for the Vc library. This option will be replaced with SIMD support from the standard library in a future release.
- We have significantly refactored our CMake setup. This is intended to be a non-breaking change and will allow for using HPX through CMake targets in the future.
- We have continued modularizing the HPX library. In the process we have rearranged many header files into module-specific directories. All moved headers have compatibility headers which forward from the old location to the new location, together with a deprecation warning. The compatibility headers will eventually be removed.
- We now enforce formatting with `clang-format` on the majority of our source files.
- We have added SPDX license tags to all files.
- Many bugfixes.

Breaking changes

- The `HPX_WITH_THREAD_COMPATIBILITY` option and the associated compatibility layer has been removed.
- The `HPX_WITH_INCLUSIVE_SCAN_COMPATIBILITY` option and the associated compatibility layer has been removed.
- The `HPX_WITH_UNWRAPPED_COMPATIBILITY` option and the associated compatibility layer has been removed.

Closed issues

- Issue #4282¹⁸⁵⁸ - Build Issues with Release on Windows
- Issue #4278¹⁸⁵⁹ - Build Issues with CMake 3.14.4
- Issue #4273¹⁸⁶⁰ - Clients of HPX 1.4.0-rc2 with APEX ar not linked to libhpx-apex

¹⁸⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4282>

¹⁸⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4278>

¹⁸⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4273>

- Issue #4269¹⁸⁶¹ - Building HPX 1.4.0-rc2 with support for APEX fails
- Issue #4263¹⁸⁶² - Compilation fail on latest master
- Issue #4232¹⁸⁶³ - Configure of HPX project using CMake FetchContent fails
- Issue #4223¹⁸⁶⁴ - “Re-using the main() function as the main HPX entry point” doesn’t work
- Issue #4220¹⁸⁶⁵ - HPX won’t compile - error building `resource_partitioner`
- Issue #4215¹⁸⁶⁶ - HPX 1.4.0rc1 does not link on s390x
- Issue #4204¹⁸⁶⁷ - Trouble compiling HPX with Intel compiler
- Issue #4199¹⁸⁶⁸ - Refactor APEX to eliminate circular dependency
- Issue #4187¹⁸⁶⁹ - HPX can’t build on OSX
- Issue #4185¹⁸⁷⁰ - Simple debug output for development
- Issue #4182¹⁸⁷¹ - @HPX_CONF_PREFIX@ is the empty string
- Issue #4169¹⁸⁷² - HPX won’t build with APEX
- Issue #4163¹⁸⁷³ - Add back `HPX_LIBRARIES` and `HPX_INCLUDE_DIRS`
- Issue #4161¹⁸⁷⁴ - It should be possible to call `find_package(HPX)` multiple times
- Issue #4155¹⁸⁷⁵ - `get_self_id()` for stackless threads returns `invalid_thread_id`
- Issue #4151¹⁸⁷⁶ - build error with MPI code
- Issue #4150¹⁸⁷⁷ - hpx won’t build on POWER9 with clang 8
- Issue #4148¹⁸⁷⁸ - `cacheline_data` delivers poor performance with C++17 compared to C++14
- Issue #4144¹⁸⁷⁹ - target general in `HPX_LIBRARIES` does not exist
- Issue #4134¹⁸⁸⁰ - CMake Error when `-DHGX_WITH_HPXMP=ON`
- Issue #4132¹⁸⁸¹ - parallel fill leaves elements unfilled
- Issue #4123¹⁸⁸² - PAPI performance counters are inaccessible
- Issue #4118¹⁸⁸³ - `static_chunk_size` is not obeyed in scan algorithms

¹⁸⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/4269>

¹⁸⁶² <https://github.com/STELLAR-GROUP/hpx/issues/4263>

¹⁸⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/4232>

¹⁸⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4223>

¹⁸⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4220>

¹⁸⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4215>

¹⁸⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4204>

¹⁸⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4199>

¹⁸⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4187>

¹⁸⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4185>

¹⁸⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/4169>

¹⁸⁷² <https://github.com/STELLAR-GROUP/hpx/issues/4169>

¹⁸⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/4163>

¹⁸⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4161>

¹⁸⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4155>

¹⁸⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4151>

¹⁸⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4150>

¹⁸⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4148>

¹⁸⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4144>

¹⁸⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/4134>

¹⁸⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/4132>

¹⁸⁸² <https://github.com/STELLAR-GROUP/hpx/issues/4123>

¹⁸⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/4118>

- Issue #4115¹⁸⁸⁴ - dependency chaining error with APEX
- Issue #4107¹⁸⁸⁵ - Initializing runtime without entry point function and command line arguments
- Issue #4105¹⁸⁸⁶ - Bug in hpx:bind=numa-balanced
- Issue #4101¹⁸⁸⁷ - Bound tasks
- Issue #4100¹⁸⁸⁸ - Add SPDX identifier to all files
- Issue #4085¹⁸⁸⁹ - hpx_topology library should depend on hwloc
- Issue #4067¹⁸⁹⁰ - HPX fails to build on macOS
- Issue #4056¹⁸⁹¹ - Building without thread manager idle backoff fails
- Issue #4052¹⁸⁹² - Enforce clang-format style for modules
- Issue #4032¹⁸⁹³ - Simple hello world fails to launch correctly
- Issue #4030¹⁸⁹⁴ - Allow threads to skip context switching
- Issue #4029¹⁸⁹⁵ - Add support for mimalloc
- Issue #4005¹⁸⁹⁶ - Can't link HPX when APEX enabled
- Issue #4002¹⁸⁹⁷ - Missing header for algorithm module
- Issue #3989¹⁸⁹⁸ - conversion from long to unsigned int requires a narrowing conversion on MSVC
- Issue #3958¹⁸⁹⁹ - /statistics/average@ perf counter can't be created
- Issue #3953¹⁹⁰⁰ - CMake errors from HPX_AddPseudoDependencies
- Issue #3941¹⁹⁰¹ - CMake error for APEX install target
- Issue #3940¹⁹⁰² - Convert pseudo-doxygen function documentation into actual doxygen documentation
- Issue #3935¹⁹⁰³ - HPX compiler match too strict?
- Issue #3929¹⁹⁰⁴ - Buildbot failures on latest HPX stable
- Issue #3912¹⁹⁰⁵ - I recommend publishing a version that does not depend on the boost library
- Issue #3890¹⁹⁰⁶ - hpx.ini not working

¹⁸⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/4115>

¹⁸⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/4107>

¹⁸⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/4105>

¹⁸⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/4101>

¹⁸⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/4100>

¹⁸⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/4085>

1890 <https://github.com/STELLAR-GROUP/hpx/issues/4067>1891 <https://github.com/STELLAR-GROUP/hpx/issues/4056>1892 <https://github.com/STELLAR-GROUP/hpx/issues/4052>1893 <https://github.com/STELLAR-GROUP/hpx/issues/4032>1894 <https://github.com/STELLAR-GROUP/hpx/issues/4030>1895 <https://github.com/STELLAR-GROUP/hpx/issues/4029>1896 <https://github.com/STELLAR-GROUP/hpx/issues/4005>1897 <https://github.com/STELLAR-GROUP/hpx/issues/4002>1898 <https://github.com/STELLAR-GROUP/hpx/issues/3989>1899 <https://github.com/STELLAR-GROUP/hpx/issues/3958>1900 <https://github.com/STELLAR-GROUP/hpx/issues/3953>1901 <https://github.com/STELLAR-GROUP/hpx/issues/3941>1902 <https://github.com/STELLAR-GROUP/hpx/issues/3940>1903 <https://github.com/STELLAR-GROUP/hpx/issues/3935>1904 <https://github.com/STELLAR-GROUP/hpx/issues/3929>1905 <https://github.com/STELLAR-GROUP/hpx/issues/3912>1906 <https://github.com/STELLAR-GROUP/hpx/issues/3890>

- Issue #3883¹⁹⁰⁷ - cuda compilation fails because of `-faligned-new`
- Issue #3879¹⁹⁰⁸ - HPX fails to configure with `-DHPX_WITH_TESTS=OFF`
- Issue #3871¹⁹⁰⁹ - dataflow does not support void allocators
- Issue #3867¹⁹¹⁰ - Latest HTML docs placed in wrong directory on GitHub pages
- Issue #3866¹⁹¹¹ - Make sure all tests use `HPX_TEST*` macros and not `HPX_ASSERT`
- Issue #3857¹⁹¹² - CMake all-keyword or all-plain for `target_link_libraries`
- Issue #3856¹⁹¹³ - `hpx_setup_target` adds rogue flags
- Issue #3850¹⁹¹⁴ - HPX fails to build on POWER8 with Clang7
- Issue #3848¹⁹¹⁵ - Remove `lva` member from `thread_init_data`
- Issue #3838¹⁹¹⁶ - `hpx::parallel::count/count_if` failing tests
- Issue #3651¹⁹¹⁷ - `hpx::parallel::transform_reduce` with non const reference as lambda parameter
- Issue #3560¹⁹¹⁸ - Apex integration with HPX not working properly
- Issue #3322¹⁹¹⁹ - No warning when mixing debug/release builds

Closed pull requests

- PR #4300¹⁹²⁰ - Checks for `MPI_Init` being called twice
- PR #4299¹⁹²¹ - Small CMake fixes
- PR #4298¹⁹²² - Remove extra call to annotate function that messes up traces
- PR #4296¹⁹²³ - Fixing collectives locking problem
- PR #4295¹⁹²⁴ - Do not check `LICENSE_1_0.txt` for inspect violations
- PR #4293¹⁹²⁵ - Applying two small changes fixing carious MSVC/Windows problems
- PR #4285¹⁹²⁶ - Delete `apex.hpp`
- PR #4276¹⁹²⁷ - Disable doxygen generation for `hpx/debugging/print.hpp` file
- PR #4275¹⁹²⁸ - Make sure APEX is linked to even when not explicitly referenced

¹⁹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3883>

¹⁹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3879>

¹⁹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3871>

¹⁹¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3867>

¹⁹¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/3866>

¹⁹¹² <https://github.com/STELLAR-GROUP/hpx/issues/3857>

¹⁹¹³ <https://github.com/STELLAR-GROUP/hpx/issues/3856>

1914 <https://github.com/STELLAR-GROUP/hpx/issues/3850>1915 <https://github.com/STELLAR-GROUP/hpx/issues/3848>1916 <https://github.com/STELLAR-GROUP/hpx/issues/3838>1917 <https://github.com/STELLAR-GROUP/hpx/issues/3651>1918 <https://github.com/STELLAR-GROUP/hpx/issues/3560>1919 <https://github.com/STELLAR-GROUP/hpx/issues/3322>1920 <https://github.com/STELLAR-GROUP/hpx/pull/4300>1921 <https://github.com/STELLAR-GROUP/hpx/pull/4299>1922 <https://github.com/STELLAR-GROUP/hpx/pull/4298>1923 <https://github.com/STELLAR-GROUP/hpx/pull/4296>1924 <https://github.com/STELLAR-GROUP/hpx/pull/4295>1925 <https://github.com/STELLAR-GROUP/hpx/pull/4293>1926 <https://github.com/STELLAR-GROUP/hpx/pull/4285>1927 <https://github.com/STELLAR-GROUP/hpx/pull/4276>1928 <https://github.com/STELLAR-GROUP/hpx/pull/4275>

- PR #4272¹⁹²⁹ - Fix pushing of documentation
- PR #4271¹⁹³⁰ - Updating APEX tag, don't create new task_wrapper on operator= of hpx_thread object
- PR #4268¹⁹³¹ - Testing for noexcept function specializations in C++11/14 mode
- PR #4267¹⁹³² - Fixing MSVC warning
- PR #4266¹⁹³³ - Make sure macOS Travis CI fails if build step fails
- PR #4264¹⁹³⁴ - Clean up compatibility header options
- PR #4262¹⁹³⁵ - Cleanup modules CMakeLists.txt
- PR #4261¹⁹³⁶ - Fixing HPX/APEX linking and dependencies for external projects like Phylanx
- PR #4260¹⁹³⁷ - Fix docs compilation problems
- PR #4258¹⁹³⁸ - Couple of minor changes
- PR #4257¹⁹³⁹ - Fix apex annotation for async dispatch
- PR #4256¹⁹⁴⁰ - Remove lambdas from assert expressions
- PR #4255¹⁹⁴¹ - Ignoring lock in all_to_all and all_reduce
- PR #4254¹⁹⁴² - Adding action specializations for noexcept functions
- PR #4253¹⁹⁴³ - Move partlit.hpp to affinity module
- PR #4252¹⁹⁴⁴ - Make mismatching build types a hard error in CMake
- PR #4249¹⁹⁴⁵ - Scheduler improvement
- PR #4248¹⁹⁴⁶ - update hpxmp tag to v0.3.0
- PR #4245¹⁹⁴⁷ - Adding high performance channels
- PR #4244¹⁹⁴⁸ - Ignore lock in ignore_while_locked_1485 test
- PR #4243¹⁹⁴⁹ - Fix PAPI command line option documentation
- PR #4242¹⁹⁵⁰ - Ignore lock in target_distribution_policy
- PR #4241¹⁹⁵¹ - Fix start_stop_callbacks test

¹⁹²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4272>

¹⁹³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4271>

¹⁹³¹ <https://github.com/STELLAR-GROUP/hpx/pull/4268>

¹⁹³² <https://github.com/STELLAR-GROUP/hpx/pull/4267>

¹⁹³³ <https://github.com/STELLAR-GROUP/hpx/pull/4266>

¹⁹³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4264>

¹⁹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4262>

¹⁹³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4261>

¹⁹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4260>

¹⁹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4258>

¹⁹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4257>

¹⁹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4256>

¹⁹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/4255>

¹⁹⁴² <https://github.com/STELLAR-GROUP/hpx/pull/4254>

¹⁹⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/4253>

¹⁹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4252>

¹⁹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4249>

¹⁹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4248>

¹⁹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4245>

¹⁹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4244>

¹⁹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4243>

¹⁹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4242>

¹⁹⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4241>

- PR #4240¹⁹⁵² - Mostly fix clang CUDA compilation
- PR #4238¹⁹⁵³ - Google Season of Docs updates to documentation; grammar edits.
- PR #4237¹⁹⁵⁴ - fixing annotated task to use the name, not the desc
- PR #4236¹⁹⁵⁵ - Move module print summary to modules
- PR #4235¹⁹⁵⁶ - Don't use alignas in cache_{aligned, line}_data
- PR #4234¹⁹⁵⁷ - Add basic overview sentence to all modules
- PR #4230¹⁹⁵⁸ - Add OS X builds to Travis CI
- PR #4229¹⁹⁵⁹ - Remove leftover queue compatibility checks
- PR #4226¹⁹⁶⁰ - Fixing APEX shutdown by explicitly shutting down throttling
- PR #4225¹⁹⁶¹ - Allow CMAKE_INSTALL_PREFIX to be a relative path
- PR #4224¹⁹⁶² - Deprecate verbs parcelport
- PR #4222¹⁹⁶³ - Update register_{thread, work} namespaces
- PR #4221¹⁹⁶⁴ - Changing HPX_GCC_VERSION check from 70000 to 70300
- PR #4218¹⁹⁶⁵ - Google Season of Docs updates to documentation; grammar edits.
- PR #4217¹⁹⁶⁶ - Google Season of Docs updates to documentation; grammar edits.
- PR #4216¹⁹⁶⁷ - Fixing gcc warning on 32bit platforms (integer truncation)
- PR #4214¹⁹⁶⁸ - Apex callback refactoring
- PR #4213¹⁹⁶⁹ - Clean up allocator checks for dependent projects
- PR #4212¹⁹⁷⁰ - Google Season of Docs updates to documentation; grammar edits.
- PR #4211¹⁹⁷¹ - Google Season of Docs updates to documentation; contributing to hpx
- PR #4210¹⁹⁷² - Attempting to fix Intel compilation
- PR #4209¹⁹⁷³ - Fix CUDA 10 build
- PR #4205¹⁹⁷⁴ - Making sure that differences in CMAKE_BUILD_TYPE are not reported on multi-configuration cmake generators

¹⁹⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4240>

¹⁹⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4238>

¹⁹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4237>

¹⁹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4236>

¹⁹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4235>

¹⁹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4234>

¹⁹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4230>

¹⁹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4229>

¹⁹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4226>

¹⁹⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/4225>

¹⁹⁶² <https://github.com/STELLAR-GROUP/hpx/pull/4224>

¹⁹⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/4222>

¹⁹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4221>

¹⁹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4218>

¹⁹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4217>

¹⁹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4216>

¹⁹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4214>

¹⁹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4213>

¹⁹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4212>

¹⁹⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/4211>

¹⁹⁷² <https://github.com/STELLAR-GROUP/hpx/pull/4210>

¹⁹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/4209>

¹⁹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4205>

- PR #4203¹⁹⁷⁵ - Deprecate Vc
- PR #4202¹⁹⁷⁶ - Fix CUDA configuration
- PR #4200¹⁹⁷⁷ - Making sure hpx_wrap is not passed on to linker on non-Linux systems
- PR #4198¹⁹⁷⁸ - Fix execution_agent.cpp compilation with GCC 5
- PR #4197¹⁹⁷⁹ - Remove deprecated options for 1.4.0 release
- PR #4196¹⁹⁸⁰ - minor fixes for building on OSX Darwin
- PR #4195¹⁹⁸¹ - Use full clone on CircleCI for pushing stable tag
- PR #4193¹⁹⁸² - Add scheduling hints to hello_world_distributed
- PR #4192¹⁹⁸³ - Set up CUDA in HPXConfig.cmake
- PR #4191¹⁹⁸⁴ - Export allocators root variables
- PR #4190¹⁹⁸⁵ - Don't use constexpr in thread_data with GCC <= 6
- PR #4189¹⁹⁸⁶ - Only use quick_exit if available
- PR #4188¹⁹⁸⁷ - Google Season of Docs updates to documentation; writing single node hpx applications
- PR #4186¹⁹⁸⁸ - correct vc to cuda in cuda cmake
- PR #4184¹⁹⁸⁹ - Resetting some cached variables to make sure those are re-filled
- PR #4183¹⁹⁹⁰ - Fix hpxcxx configuration
- PR #4181¹⁹⁹¹ - Rename base libraries var
- PR #4180¹⁹⁹² - Move header left behind earlier to plugin module
- PR #4179¹⁹⁹³ - Moving zip_iterator and transform_iterator to iterator_support module
- PR #4178¹⁹⁹⁴ - Move checkpointing support to its own module
- PR #4177¹⁹⁹⁵ - Small const fix to basic_execution module
- PR #4176¹⁹⁹⁶ - Add back HPX_LIBRARIES and friends to HPXConfig.cmake
- PR #4175¹⁹⁹⁷ - Make Vc public and add it to HPXConfig.cmake

¹⁹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4203>

¹⁹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4202>

¹⁹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4200>

¹⁹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4198>

¹⁹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4197>

¹⁹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4196>

¹⁹⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/4195>

¹⁹⁸² <https://github.com/STELLAR-GROUP/hpx/pull/4193>

¹⁹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/4192>

¹⁹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4191>

¹⁹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4190>

¹⁹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4189>

¹⁹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4188>

¹⁹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4186>

¹⁹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4184>

¹⁹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4183>

¹⁹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4181>

¹⁹⁹² <https://github.com/STELLAR-GROUP/hpx/pull/4180>

¹⁹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/4179>

¹⁹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4178>

¹⁹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4177>

¹⁹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4176>

¹⁹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4175>

- PR #4173¹⁹⁹⁸ - Wait for runtime to be running before returning from hpx::start
- PR #4172¹⁹⁹⁹ - More protection against shutdown problems in error handling scenarios.
- PR #4171²⁰⁰⁰ - Ignore lock in condition_variable::wait
- PR #4170²⁰⁰¹ - Adding APEX dependency to MPI parcelport
- PR #4168²⁰⁰² - Adding utility include
- PR #4167²⁰⁰³ - Add a condition to setup the external libraries
- PR #4166²⁰⁰⁴ - Add an INTERNAL_FLAGS option to link to hpx_internal_flags
- PR #4165²⁰⁰⁵ - Forward HPX_* cmake cache variables to external projects
- PR #4164²⁰⁰⁶ - Affinity and batch environment modules
- PR #4162²⁰⁰⁷ - Handle quick_exit
- PR #4160²⁰⁰⁸ - Using target_link_libraries for cmake versions >= 3.12
- PR #4159²⁰⁰⁹ - Make sure HPX_WITH_NATIVE_TLS is forwarded to dependent projects
- PR #4158²⁰¹⁰ - Adding allocator imported target as a dependency of allocator module
- PR #4157²⁰¹¹ - Add hpx_memory as a dependency of parcelport plugins
- PR #4156²⁰¹² - Stackless coroutines now can refer to themselves (through get_self() and friends)
- PR #4154²⁰¹³ - Added CMake policy CMP0060 for HPX applications.
- PR #4153²⁰¹⁴ - add header iomanip to tests and tool
- PR #4152²⁰¹⁵ - Casting MPI tag value
- PR #4149²⁰¹⁶ - Add back private m_desc member variable in program_options module
- PR #4147²⁰¹⁷ - Resource partitioner and threadmanager modules
- PR #4146²⁰¹⁸ - Google Season of Docs updates to documentation; creating hpx projects
- PR #4145²⁰¹⁹ - Adding basic support for stackless threads
- PR #4143²⁰²⁰ - Exclude test_client_1950 from all target

¹⁹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4173>

¹⁹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4172>

²⁰⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4171>

²⁰⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/4170>

²⁰⁰² <https://github.com/STELLAR-GROUP/hpx/pull/4168>

²⁰⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/4167>

²⁰⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4166>

²⁰⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4165>

²⁰⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4164>

²⁰⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4162>

²⁰⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4160>

²⁰⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4159>

²⁰¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4158>

²⁰¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4157>

2012 <https://github.com/STELLAR-GROUP/hpx/pull/4156>2013 <https://github.com/STELLAR-GROUP/hpx/pull/4154>2014 <https://github.com/STELLAR-GROUP/hpx/pull/4153>2015 <https://github.com/STELLAR-GROUP/hpx/pull/4152>2016 <https://github.com/STELLAR-GROUP/hpx/pull/4149>2017 <https://github.com/STELLAR-GROUP/hpx/pull/4147>2018 <https://github.com/STELLAR-GROUP/hpx/pull/4146>2019 <https://github.com/STELLAR-GROUP/hpx/pull/4145>2020 <https://github.com/STELLAR-GROUP/hpx/pull/4143>

- PR #4142²⁰²¹ - Add a new `thread_pool_executor`
- PR #4140²⁰²² - Google Season of Docs updates to documentation; why hpx
- PR #4139²⁰²³ - Remove runtime includes from coroutines module
- PR #4138²⁰²⁴ - Forking `boost::intrusive_ptr` and adding it as `hpx::intrusive_ptr`
- PR #4137²⁰²⁵ - Fixing TSS destruction
- PR #4136²⁰²⁶ - HPX.Compute modules
- PR #4133²⁰²⁷ - Fix `block_executor`
- PR #4131²⁰²⁸ - Applying fixes based on reports from PVS Studio
- PR #4130²⁰²⁹ - Adding missing header to build system
- PR #4129²⁰³⁰ - Fixing compilation if `HPX_WITH_DATAPAR_VC` is enabled
- PR #4128²⁰³¹ - Renaming `moveonly_any` to `unique_any`
- PR #4126²⁰³² - Attempt to fix `basic_any` constructor for gcc 7
- PR #4125²⁰³³ - Changing `extra_archive_data` implementation
- PR #4124²⁰³⁴ - Don't link to Boost.System unless required
- PR #4122²⁰³⁵ - Add kernel launch helper utility (+saxpy demo) and merge in octotiger changes
- PR #4121²⁰³⁶ - Fixing migration test if networking is disabled.
- PR #4120²⁰³⁷ - Google Season of Docs updates to documentation; hpx build system v1
- PR #4119²⁰³⁸ - Making sure `chunk_size` and `max_chunk` are actually applied to parallel algorithms if specified
- PR #4117²⁰³⁹ - Make CircleCI formatting check store diff
- PR #4116²⁰⁴⁰ - Fix automatically setting C++ standard
- PR #4114²⁰⁴¹ - Module serialization
- PR #4113²⁰⁴² - Module datastructures
- PR #4111²⁰⁴³ - Fixing performance regression introduced earlier

²⁰²¹ <https://github.com/STELLAR-GROUP/hpx/pull/4142>

²⁰²² <https://github.com/STELLAR-GROUP/hpx/pull/4140>

²⁰²³ <https://github.com/STELLAR-GROUP/hpx/pull/4139>

²⁰²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4138>

²⁰²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4137>

²⁰²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4136>

²⁰²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4133>

²⁰²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4131>

²⁰²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4130>

²⁰³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4129>

²⁰³¹ <https://github.com/STELLAR-GROUP/hpx/pull/4128>

²⁰³² <https://github.com/STELLAR-GROUP/hpx/pull/4126>

²⁰³³ <https://github.com/STELLAR-GROUP/hpx/pull/4125>

²⁰³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4124>

²⁰³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4122>

²⁰³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4121>

²⁰³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4120>

²⁰³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4119>

²⁰³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4117>

²⁰⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4116>

²⁰⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/4114>

²⁰⁴² <https://github.com/STELLAR-GROUP/hpx/pull/4113>

²⁰⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/4111>

- PR #4110²⁰⁴⁴ - Adding missing SPDX tags
- PR #4109²⁰⁴⁵ - Overload for start without entry point/argv.
- PR #4108²⁰⁴⁶ - Making sure C++ standard is properly detected and propagated
- PR #4106²⁰⁴⁷ - use std::round for guaranteed rounding without errors
- PR #4104²⁰⁴⁸ - Extend scheduler_mode with new work_stealing and task assignment modes
- PR #4103²⁰⁴⁹ - Add this to lambda capture list
- PR #4102²⁰⁵⁰ - Add spdx license and check
- PR #4099²⁰⁵¹ - Module coroutines
- PR #4098²⁰⁵² - Fix append module path in module CMakeLists template
- PR #4097²⁰⁵³ - Function tests
- PR #4096²⁰⁵⁴ - Removing return of thread_result_type from functions not needing them
- PR #4095²⁰⁵⁵ - Stop-gap measure until cmake overhaul is in place
- PR #4094²⁰⁵⁶ - Deprecate HPX_WITH_MORE_THAN_64_THREADS
- PR #4093²⁰⁵⁷ - Fix initialization of global_num_tasks in parallel_executor
- PR #4092²⁰⁵⁸ - Add support for mi-malloc
- PR #4090²⁰⁵⁹ - Execution context
- PR #4089²⁰⁶⁰ - Make counters in coroutines optional
- PR #4087²⁰⁶¹ - Making hpx::util::any compatible with C++17
- PR #4084²⁰⁶² - Making sure destination array for std::transform is properly resized
- PR #4083²⁰⁶³ - Adapting thread_queue_mc to behave even if no 128bit atomics are available
- PR #4082²⁰⁶⁴ - Fix compilation on GCC 5
- PR #4081²⁰⁶⁵ - Adding option allowing to force using Boost.FileSystem
- PR #4080²⁰⁶⁶ - Updating module dependencies

²⁰⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4110>

²⁰⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4109>

²⁰⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4108>

²⁰⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4106>

²⁰⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4104>

²⁰⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4103>

²⁰⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4102>

²⁰⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/4099>

²⁰⁵² <https://github.com/STELLAR-GROUP/hpx/pull/4098>

²⁰⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/4097>

²⁰⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4096>

²⁰⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4095>

²⁰⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4094>

²⁰⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4093>

²⁰⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4092>

²⁰⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4090>

²⁰⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4089>

²⁰⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/4087>

²⁰⁶² <https://github.com/STELLAR-GROUP/hpx/pull/4084>

²⁰⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/4083>

²⁰⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4082>

²⁰⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4081>

²⁰⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4080>

- PR #4079²⁰⁶⁷ - Add missing tests for iterator_support module
- PR #4078²⁰⁶⁸ - Disable parcel-layer if networking is disabled
- PR #4077²⁰⁶⁹ - Add missing include that causes build fails
- PR #4076²⁰⁷⁰ - Enable compatibility headers for functional module
- PR #4075²⁰⁷¹ - Coroutines module
- PR #4073²⁰⁷² - Use configure_file for generated files in modules
- PR #4071²⁰⁷³ - Fixing MPI detection for PMIx
- PR #4070²⁰⁷⁴ - Fix macOS builds
- PR #4069²⁰⁷⁵ - Moving more facilities to the collectives module
- PR #4068²⁰⁷⁶ - Adding main HPX #include directory to modules
- PR #4066²⁰⁷⁷ - Switching the use of message (STATUS "...") to hpx_info
- PR #4065²⁰⁷⁸ - Move Boost.Filesystem handling to filesystem module
- PR #4064²⁰⁷⁹ - Fix program_options test with older boost versions
- PR #4062²⁰⁸⁰ - The cpu_features tool fails to compile on anything but x86 architectures
- PR #4061²⁰⁸¹ - Add clang-format checking step for modules
- PR #4060²⁰⁸² - Making sure HPX_IDLE_BACKOFF_TIME_MAX is always defined (even if its unused)
- PR #4059²⁰⁸³ - Renaming module hpx_parallel_executors into hpx_execution
- PR #4058²⁰⁸⁴ - Do not build networking tests when networking disabled
- PR #4057²⁰⁸⁵ - Printing configuration summary for modules as well
- PR #4055²⁰⁸⁶ - Google Season of Docs updates to documentation; hpx build systems
- PR #4054²⁰⁸⁷ - Add troubleshooting section to manual
- PR #4051²⁰⁸⁸ - Add more variations to future_overhead test
- PR #4050²⁰⁸⁹ - Creating plugin module

²⁰⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4079>

²⁰⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4078>

²⁰⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4077>

²⁰⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4076>

²⁰⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/4075>

²⁰⁷² <https://github.com/STELLAR-GROUP/hpx/pull/4073>

²⁰⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/4071>

²⁰⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4070>

²⁰⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4069>

²⁰⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4068>

²⁰⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4066>

²⁰⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4065>

²⁰⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4064>

²⁰⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4062>

²⁰⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/4061>

²⁰⁸² <https://github.com/STELLAR-GROUP/hpx/pull/4060>

²⁰⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/4059>

²⁰⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4058>

²⁰⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4057>

²⁰⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4055>

²⁰⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4054>

²⁰⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4051>

²⁰⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4050>

- PR #4049²⁰⁹⁰ - Move missing modules tests
- PR #4047²⁰⁹¹ - Add boost/filesystem headers to inspect deprecated headers
- PR #4045²⁰⁹² - Module functional
- PR #4043²⁰⁹³ - Fix preconditions and error messages for suspension functions
- PR #4041²⁰⁹⁴ - Pass HPX_STANDARD on to dependent projects via HPXConfig.cmake
- PR #4040²⁰⁹⁵ - Program options module
- PR #4039²⁰⁹⁶ - Moving non-serializable any (any_nonsr) to datastructures module
- PR #4038²⁰⁹⁷ - Adding MPark's variant (V1.4.0) to HPX
- PR #4037²⁰⁹⁸ - Adding resiliency module
- PR #4036²⁰⁹⁹ - Add C++17 filesystem compatibility header
- PR #4035²¹⁰⁰ - Fixing support for mpirun
- PR #4028²¹⁰¹ - CMake to target based directives
- PR #4027²¹⁰² - Remove GitLab CI configuration
- PR #4026²¹⁰³ - Threading refactoring
- PR #4025²¹⁰⁴ - Refactoring thread queue configuration options
- PR #4024²¹⁰⁵ - Fix padding calculation in cache_aligned_data.hpp
- PR #4023²¹⁰⁶ - Fixing Codacy issues
- PR #4022²¹⁰⁷ - Make sure process mask option is passed to affinity_data
- PR #4021²¹⁰⁸ - Warn about compiling in C++11 mode
- PR #4020²¹⁰⁹ - Module concurrency
- PR #4019²¹¹⁰ - Module topology
- PR #4018²¹¹¹ - Update deprecated header in thread_queue_mc.hpp
- PR #4015²¹¹² - Avoid overwriting artifacts

²⁰⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4049>

²⁰⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4047>

²⁰⁹² <https://github.com/STELLAR-GROUP/hpx/pull/4045>

²⁰⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/4043>

²⁰⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4041>

²⁰⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4040>

²⁰⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4039>

²⁰⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4038>

²⁰⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4037>

²⁰⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4036>

²¹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4035>

²¹⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/4028>

²¹⁰² <https://github.com/STELLAR-GROUP/hpx/pull/4027>

²¹⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/4026>

²¹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4025>

²¹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4024>

²¹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4023>

²¹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4022>

²¹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4021>

²¹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4020>

²¹¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4019>

²¹¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/4018>

²¹¹² <https://github.com/STELLAR-GROUP/hpx/pull/4015>

- PR #4014²¹¹³ - Future overheads
- PR #4013²¹¹⁴ - Update URL to test output conversion script
- PR #4012²¹¹⁵ - Fix CUDA compilation
- PR #4011²¹¹⁶ - Fixing cyclic dependencies between modules
- PR #4010²¹¹⁷ - Ignore stable tag on CircleCI
- PR #4009²¹¹⁸ - Check circular dependencies in a circle ci step
- PR #4008²¹¹⁹ - Extend cache aligned data to handle tuple-like data
- PR #4007²¹²⁰ - Fixing migration for components that have actions returning a client
- PR #4006²¹²¹ - Move is_value_proxy.hpp to algorithms module
- PR #4004²¹²² - Shorten CTest timeout on CircleCI
- PR #4003²¹²³ - Refactoring to remove (internal) dependencies
- PR #4001²¹²⁴ - Exclude tests from all target
- PR #4000²¹²⁵ - Module errors
- PR #3999²¹²⁶ - Enable support for compatibility headers for logging module
- PR #3998²¹²⁷ - Add process thread binding option
- PR #3997²¹²⁸ - Export handle_assert function
- PR #3996²¹²⁹ - Attempt to solve issue where `-latomic` does not support 128bit atomics
- PR #3993²¹³⁰ - Make sure `__LINE__` is an unsigned
- PR #3991²¹³¹ - Fix dependencies and flags for header tests
- PR #3990²¹³² - Documentation tags fixes
- PR #3988²¹³³ - Adding missing solution folder for format module test
- PR #3987²¹³⁴ - Move runtime-dependent functions out of command line handling
- PR #3986²¹³⁵ - Fix CMake configuration with PAPI on

²¹¹³ <https://github.com/STELLAR-GROUP/hpx/pull/4014>

²¹¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4013>

²¹¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4012>

²¹¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/4011>

²¹¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/4010>

²¹¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/4009>

²¹¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/4008>

²¹²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/4007>

²¹²¹ <https://github.com/STELLAR-GROUP/hpx/pull/4006>

²¹²² <https://github.com/STELLAR-GROUP/hpx/pull/4004>

²¹²³ <https://github.com/STELLAR-GROUP/hpx/pull/4003>

²¹²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/4001>

²¹²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/4000>

²¹²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3999>

²¹²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3998>

²¹²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3997>

²¹²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3996>

²¹³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3993>

²¹³¹ <https://github.com/STELLAR-GROUP/hpx/pull/3991>

²¹³² <https://github.com/STELLAR-GROUP/hpx/pull/3990>

²¹³³ <https://github.com/STELLAR-GROUP/hpx/pull/3988>

²¹³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3987>

²¹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3986>

- PR #3985²¹³⁶ - Module timing
- PR #3984²¹³⁷ - Fix default behaviour of paths in add_hpx_component
- PR #3982²¹³⁸ - Parallel executors module
- PR #3981²¹³⁹ - Segmented algorithms module
- PR #3980²¹⁴⁰ - Module logging
- PR #3979²¹⁴¹ - Module util
- PR #3978²¹⁴² - Fix clang-tidy step on CircleCI
- PR #3977²¹⁴³ - Fixing solution folders for moved components
- PR #3976²¹⁴⁴ - Module format
- PR #3975²¹⁴⁵ - Enable deprecation warnings on CircleCI
- PR #3974²¹⁴⁶ - Fix typos in documentation
- PR #3973²¹⁴⁷ - Fix compilation with GCC 9
- PR #3972²¹⁴⁸ - Add condition to clone apex + use of new cmake var APEX_ROOT
- PR #3971²¹⁴⁹ - Add testing module
- PR #3968²¹⁵⁰ - Remove unneeded file in hardware module
- PR #3967²¹⁵¹ - Remove leftover PIC settings from main CMakeLists.txt
- PR #3966²¹⁵² - Add missing export option in add_hpx_module
- PR #3965²¹⁵³ - Change current_function_helper back to non-constexpr
- PR #3964²¹⁵⁴ - Fixing merge problems
- PR #3962²¹⁵⁵ - Add a trait for std::array for unwrapping
- PR #3961²¹⁵⁶ - Making hpx::util::tuple<Ts...> and std::tuple<Ts...> convertible
- PR #3960²¹⁵⁷ - fix compilation with CUDA 10 and GCC 6
- PR #3959²¹⁵⁸ - Fix C++11 incompatibility

²¹³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3985>

²¹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3984>

²¹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3982>

²¹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3981>

²¹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3980>

²¹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/3979>

²¹⁴² <https://github.com/STELLAR-GROUP/hpx/pull/3978>

²¹⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/3977>

²¹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3976>

²¹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3975>

²¹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3974>

²¹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3973>

²¹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3972>

²¹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3971>

²¹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3968>

²¹⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3967>

²¹⁵² <https://github.com/STELLAR-GROUP/hpx/pull/3966>

²¹⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3965>

²¹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3964>

²¹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3962>

²¹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3961>

²¹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3960>

²¹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3959>

- PR #3957²¹⁵⁹ - Algorithms module
- PR #3956²¹⁶⁰ - [HPX_AddModule] Fix lower name var to upper
- PR #3955²¹⁶¹ - Fix CMake configuration with examples off and tests on
- PR #3954²¹⁶² - Move components to separate subdirectory in root of repository
- PR #3952²¹⁶³ - Update papi.cpp
- PR #3951²¹⁶⁴ - Exclude modules header tests from all target
- PR #3950²¹⁶⁵ - Adding all_reduce facility to collectives module
- PR #3949²¹⁶⁶ - This adds a configuration file that will cause for stale issues to be automatically closed
- PR #3948²¹⁶⁷ - Fixing ALPS environment
- PR #3947²¹⁶⁸ - Add major compiler version check for building hpx as a binary package
- PR #3946²¹⁶⁹ - [Modules] Move the location of the generated headers
- PR #3945²¹⁷⁰ - Simplify tests and examples cmake
- PR #3943²¹⁷¹ - Remove example module
- PR #3942²¹⁷² - Add NOEXPORT option to add_hpx_{component,library}
- PR #3938²¹⁷³ - Use https for CDash submissions
- PR #3937²¹⁷⁴ - Add HPX_WITH_BUILD_BINARY_PACKAGE to the compiler check (refs #3935)
- PR #3936²¹⁷⁵ - Fixing installation of binaries on windows
- PR #3934²¹⁷⁶ - Add set function for sliding_semaphore max_difference
- PR #3933²¹⁷⁷ - Remove cudadevrt from compile/link flags as it breaks downstream projects
- PR #3932²¹⁷⁸ - Fixing 3929
- PR #3931²¹⁷⁹ - Adding all_to_all
- PR #3930²¹⁸⁰ - Add test demonstrating the use of broadcast with component actions
- PR #3928²¹⁸¹ - fixed number of tasks and number of threads for heterogeneous slurm environments

²¹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3957>

²¹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3956>

²¹⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3955>

²¹⁶² <https://github.com/STELLAR-GROUP/hpx/pull/3954>

²¹⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/3952>

²¹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3951>

²¹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3950>

²¹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3949>

²¹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3948>

²¹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3947>

²¹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3946>

²¹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3945>

²¹⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3943>

²¹⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3942>

²¹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3938>

²¹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3937>

²¹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3936>

²¹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3934>

²¹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3933>

²¹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3932>

²¹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3931>

²¹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3930>

²¹⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/3928>

- PR #3927²¹⁸² - Moving Cache module's tests into separate solution folder
- PR #3926²¹⁸³ - Move unit tests to cache module
- PR #3925²¹⁸⁴ - Move version check to config module
- PR #3924²¹⁸⁵ - Add schedule hint executor parameters
- PR #3923²¹⁸⁶ - Allow aligning objects bigger than the cache line size
- PR #3922²¹⁸⁷ - Add Windows builds with Travis CI
- PR #3921²¹⁸⁸ - Add ccls cache directory to gitignore
- PR #3920²¹⁸⁹ - Fix git_external fetching of tags
- PR #3905²¹⁹⁰ - Correct rostambod url. Fix typo in doc
- PR #3904²¹⁹¹ - Fix bug in context_base.hpp
- PR #3903²¹⁹² - Adding new performance counters
- PR #3902²¹⁹³ - Add add_hpx_module function
- PR #3901²¹⁹⁴ - Factoring out container remapping into a separate trait
- PR #3900²¹⁹⁵ - Making sure errors during command line processing are properly reported and will not cause assertions
- PR #3899²¹⁹⁶ - Remove old compatibility bases from make_action
- PR #3898²¹⁹⁷ - Make parameter size be of type size_t
- PR #3897²¹⁹⁸ - Making sure all tests are disabled if HPX_WITH_TESTS=OFF
- PR #3895²¹⁹⁹ - Add documentation for annotated_function
- PR #3894²²⁰⁰ - Working around VS2019 problem with make_action
- PR #3892²²⁰¹ - Avoid MSVC compatibility warning in internal allocator
- PR #3891²²⁰² - Removal of the default intel config include
- PR #3888²²⁰³ - Fix async_customization dataflow example and Clarify what's being tested
- PR #3887²²⁰⁴ - Add Doxygen documentation

²¹⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3927>

²¹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/3926>

²¹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3925>

²¹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3924>

²¹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3923>

²¹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3922>

²¹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3921>

²¹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3920>

²¹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3905>

²¹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3904>

²¹⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3903>

²¹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3902>

²¹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3901>

²¹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3900>

²¹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3899>

²¹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3898>

²¹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3897>

²¹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3895>

²²⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3894>

²²⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3892>

²²⁰² <https://github.com/STELLAR-GROUP/hpx/pull/3891>

²²⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/3888>

²²⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3887>

- PR #3882²²⁰⁵ - Minor docs fixes
- PR #3880²²⁰⁶ - Updating APEX version tag
- PR #3878²²⁰⁷ - Making sure symbols are properly exported from modules (needed for Windows/MacOS)
- PR #3877²²⁰⁸ - Documentation
- PR #3876²²⁰⁹ - Module hardware
- PR #3875²²¹⁰ - Converted typedefs in actions submodule to using directives
- PR #3874²²¹¹ - Allow one to suppress target keywords in hpx_setup_target for backwards compatibility
- PR #3873²²¹² - Add scripts to create releases and generate lists of PRs and issues
- PR #3872²²¹³ - Fix latest HTML docs location
- PR #3870²²¹⁴ - Module cache
- PR #3869²²¹⁵ - Post 1.3.0 version bumps
- PR #3868²²¹⁶ - Replace the macro HPX_ASSERT by HPX_TEST in tests
- PR #3845²²¹⁷ - Assertion module
- PR #3839²²¹⁸ - Make tuple serialization non-intrusive
- PR #3832²²¹⁹ - Config module
- PR #3799²²²⁰ - Remove compat namespace and its contents
- PR #3701²²²¹ - MoodyCamel lockfree
- PR #3496²²²² - Disabling MPI's (deprecated) C++ interface
- PR #3192²²²³ - Move type info into hpx::debug namespace and add print helper functions
- PR #3159²²²⁴ - Support Checkpointing Components

²²⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3882>

²²⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3880>

²²⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3878>

²²⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3877>

²²⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3876>

²²¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3875>

²²¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3874>

²²¹² <https://github.com/STELLAR-GROUP/hpx/pull/3873>

²²¹³ <https://github.com/STELLAR-GROUP/hpx/pull/3872>

²²¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3870>

²²¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3869>

²²¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3868>

²²¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3845>

²²¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3839>

²²¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3832>

²²²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3799>

²²²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3701>

²²²² <https://github.com/STELLAR-GROUP/hpx/pull/3496>

²²²³ <https://github.com/STELLAR-GROUP/hpx/pull/3192>

²²²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3159>

2.10.9 HPX V1.3.0 (May 23, 2019)

General changes

- Performance improvements: the schedulers have significantly reduced overheads from removing false sharing and the parallel executor has been updated to create fewer futures.
- HPX now defaults to not turning on networking when running on one locality. This means that you can run multiple instances on the same system without adding command line options.
- Multiple issues reported by Clang sanitizers have been fixed.
- We have added (back) single-page HTML documentation and PDF documentation.
- We have started modularizing the HPX library. This is useful both for developers and users. In the long term users will be able to consume only parts of the HPX libraries if they do not require all the functionality that HPX currently provides.
- We have added an implementation of `function_ref`.
- The `barrier` and `latch` classes have gained a few additional member functions.

Breaking changes

- Executable and library targets are now created without the `_exe` and `_lib` suffix respectively. For example, the target `1d_stencil_1_exe` is now simply called `1d_stencil_1`.
- We have removed the following deprecated functionality: `queue`, `scoped_unlock`, and support for input iterators in algorithms.
- We have turned off the compatibility layer for `unwrapped` by default. The functionality will be removed in the next release. The option can still be turned on using the CMake²²²⁵ option `HPX_WITH_UNWRAPPED_SUPPORT`. Likewise, `inclusive_scan` compatibility overloads have been turned off by default. They can still be turned on with `HPX_WITH_INCLUSIVE_SCAN_COMPATIBILITY`.
- The minimum compiler and dependency versions have been updated. We now support GCC from version 5 onwards, Clang from version 4 onwards, and Boost from version 1.61.0 onwards.
- The headers for preprocessor macros have moved as a result of the functionality being moved to a separate module. The old headers are deprecated and will be removed in a future version of HPX. You can turn off the warnings by setting `HPX_PREPROCESSOR_WITH_DEPRECATED_WARNINGS=OFF` or turn off the compatibility headers completely with `HPX_PREPROCESSOR_WITH_COMPATIBILITY_HEADERS=OFF`.

Closed issues

- Issue #3863²²²⁶ - shouldn't “-faligned-new” be a usage requirement?
- Issue #3841²²²⁷ - Build error with msvc 19 caused by SFINAE and C++17
- Issue #3836²²²⁸ - master branch does not build with idle rate counters enabled
- Issue #3819²²²⁹ - Add debug suffix to modules built in debug mode
- Issue #3817²²³⁰ - `HPX_INCLUDE_DIRS` contains non-existent directory

²²²⁵ <https://www.cmake.org>

²²²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3863>

²²²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3841>

²²²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3836>

²²²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3819>

²²³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3817>

- Issue #3810²²³¹ - Source groups are not created for files in modules
- Issue #3805²²³² - HPX won't compile with `-DHPX_WITH_APEX=TRUE`
- Issue #3792²²³³ - Barrier Hangs When Locality Zero not included
- Issue #3778²²³⁴ - Replace `throw()` with `noexcept`
- Issue #3763²²³⁵ - configurable sort limit per task
- Issue #3758²²³⁶ - dataflow doesn't convert `future<future<T>>` to `future<T>`
- Issue #3757²²³⁷ - When compiling undefined reference to `hpx::hpx_check_version_1_2` HPX V1.2.1, Ubuntu 18.04.01 Server Edition
- Issue #3753²²³⁸ - `--hpx:list-counters=full` crashes
- Issue #3746²²³⁹ - Detection of MPI with pmix
- Issue #3744²²⁴⁰ - Separate spinlock from same cacheline as internal data for all LCOs
- Issue #3743²²⁴¹ - `hpxcxx`'s shebang doesn't specify the python version
- Issue #3738²²⁴² - Unable to debug parcelport on a single node
- Issue #3735²²⁴³ - Latest master: Can't compile in MSVC
- Issue #3731²²⁴⁴ - `util::bound` seems broken on Clang with older libstdc++
- Issue #3724²²⁴⁵ - Allow to pre-set command line options through environment
- Issue #3723²²⁴⁶ - examples/resource_partitioner build issue on master branch / ubuntu 18
- Issue #3721²²⁴⁷ - faced a building error
- Issue #3720²²⁴⁸ - Hello World example fails to link
- Issue #3719²²⁴⁹ - pkg-config produces invalid output: `-l-pthread`
- Issue #3718²²⁵⁰ - Please make the python executable configurable through cmake
- Issue #3717²²⁵¹ - interested to contribute to the organisation
- Issue #3699²²⁵² - Remove 'HPX runtime' executable
- Issue #3698²²⁵³ - Ignore all locks while handling asserts

²²³¹ <https://github.com/STELLAR-GROUP/hpx/issues/3810>

²²³² <https://github.com/STELLAR-GROUP/hpx/issues/3805>

²²³³ <https://github.com/STELLAR-GROUP/hpx/issues/3792>

²²³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3778>

²²³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3763>

²²³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3758>

²²³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3757>

²²³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3753>

²²³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3746>

²²⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3744>

²²⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/3743>

²²⁴² <https://github.com/STELLAR-GROUP/hpx/issues/3738>

²²⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/3735>

²²⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3731>

²²⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3724>

²²⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3723>

²²⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3721>

²²⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3720>

²²⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3719>

²²⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3718>

²²⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/3717>

²²⁵² <https://github.com/STELLAR-GROUP/hpx/issues/3699>

²²⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/3698>

- Issue #3689²²⁵⁴ - Incorrect and inconsistent website structure <http://stellar.cct.lsu.edu/downloads/>.
- Issue #3681²²⁵⁵ - Broken links on <http://stellar.cct.lsu.edu/2015/05/hpx-archives-now-on-gmane/>
- Issue #3676²²⁵⁶ - HPX master built from source, cmake fails to link main.cpp example in docs
- Issue #3673²²⁵⁷ - HPX build fails with `std::atomic` missing error
- Issue #3670²²⁵⁸ - Generate PDF again from documentation (with Sphinx)
- Issue #3643²²⁵⁹ - Warnings when compiling HPX 1.2.1 with gcc 9
- Issue #3641²²⁶⁰ - Trouble with using ranges-v3 and `hpx::parallel::reduce`
- Issue #3639²²⁶¹ - `util::unwrapping` does not work well with member functions
- Issue #3634²²⁶² - The build fails if `shared_future<>::then` is called with a thread executor
- Issue #3622²²⁶³ - VTune Amplifier 2019 not working with `use_itt_notify=1`
- Issue #3616²²⁶⁴ - HPX Fails to Build with CUDA 10
- Issue #3612²²⁶⁵ - False sharing of scheduling counters
- Issue #3609²²⁶⁶ - `executor_parameters timeout` with gcc <= 7 and Debug mode
- Issue #3601²²⁶⁷ - Misleading error message on power pc for `rdtsc` and `rdtscp`
- Issue #3598²²⁶⁸ - Build of some examples fails when using Vc
- Issue #3594²²⁶⁹ - Error: The number of OS threads requested (20) does not match the number of threads to bind (12): HPX(bad_parameter)
- Issue #3592²²⁷⁰ - Undefined Reference Error
- Issue #3589²²⁷¹ - include could not find load file: `HPX_Utils.cmake`
- Issue #3587²²⁷² - HPX won't compile on POWER8 with Clang 7
- Issue #3583²²⁷³ - Fedora and openSUSE instructions missing on "Distribution Packages" page
- Issue #3578²²⁷⁴ - Build error when configuring with `HPX_HAVE_ALGORITHM_INPUT_ITERATOR_SUPPORT=ON`
- Issue #3575²²⁷⁵ - Merge openSUSE reproducible patch
- Issue #3570²²⁷⁶ - Update HPX to work with the latest VC version

²²⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3689>

²²⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3681>

²²⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3676>

²²⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3673>

²²⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3670>

²²⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3643>

²²⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3641>

²²⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/3639>

²²⁶² <https://github.com/STELLAR-GROUP/hpx/issues/3634>

²²⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/3622>

²²⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3616>

²²⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3612>

²²⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3609>

²²⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3601>

²²⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3598>

²²⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3594>

²²⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3592>

²²⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/3589>

²²⁷² <https://github.com/STELLAR-GROUP/hpx/issues/3587>

²²⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/3583>

²²⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3578>

²²⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3575>

²²⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3570>

- Issue #3567²²⁷⁷ - Build succeed and make failed for hpx : cout
- Issue #3565²²⁷⁸ - Polymorphic simple component destructor not getting called
- Issue #3559²²⁷⁹ - 1.2.0 is missing from download page
- Issue #3554²²⁸⁰ - Clang 6.0 warning of hiding overloaded virtual function
- Issue #3510²²⁸¹ - Build on ppc64 fails
- Issue #3482²²⁸² - Improve error message when HPX_WITH_MAX_CPU_COUNT is too low for given system
- Issue #3453²²⁸³ - Two HPX applications can't run at the same time.
- Issue #3452²²⁸⁴ - Scaling issue on the change to 2 NUMA domains
- Issue #3442²²⁸⁵ - HPX set_difference, set_intersection failure cases
- Issue #3437²²⁸⁶ - Ensure parent_task pointer when child task is created and child/parent are on same locality
- Issue #3255²²⁸⁷ - Suspension with lock for --hpx:list-component-types
- Issue #3034²²⁸⁸ - Use C++17 structured bindings for serialization
- Issue #2999²²⁸⁹ - Change thread scheduling use of size_t for thread indexing

Closed pull requests

- PR #3865²²⁹⁰ - adds hpx_target_compile_option_if_available
- PR #3864²²⁹¹ - Helper functions that are useful in numa binding and testing of allocator
- PR #3862²²⁹² - Temporary fix to local_dataflow_boost_small_vector test
- PR #3860²²⁹³ - Add cache line padding to intermediate results in for loop reduction
- PR #3859²²⁹⁴ - Remove HPX_TLL_PUBLIC and HPX_TLL_PRIVATE from CMake files
- PR #3858²²⁹⁵ - Add compile flags and definitions to modules
- PR #3851²²⁹⁶ - update hpxmp release tag to v0.2.0
- PR #3849²²⁹⁷ - Correct BOOST_ROOT variable name in quick start guide
- PR #3847²²⁹⁸ - Fix attach_debugger configuration option

²²⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3567>

²²⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3565>

²²⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3559>

²²⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3554>

²²⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/3510>

²²⁸² <https://github.com/STELLAR-GROUP/hpx/issues/3482>

²²⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/3453>

²²⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3452>

²²⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3442>

²²⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3437>

²²⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3255>

²²⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3034>

²²⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2999>

²²⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3865>

²²⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3864>

²²⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3862>

²²⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3860>

²²⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3859>

²²⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3858>

²²⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3851>

²²⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3849>

²²⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3847>

- PR #3846²²⁹⁹ - Add tests for libs header tests
- PR #3844²³⁰⁰ - Fixing source_groups in preprocessor module to properly handle compatibility headers
- PR #3843²³⁰¹ - This fixes the launch_process/launched_process pair of tests
- PR #3842²³⁰² - Fix macro call with ITTNNOTIFY enabled
- PR #3840²³⁰³ - Fixing SLURM environment parsing
- PR #3837²³⁰⁴ - Fixing misplaced #endif
- PR #3835²³⁰⁵ - make all latch members protected for consistency
- PR #3834²³⁰⁶ - Disable transpose_block numa example on CircleCI
- PR #3833²³⁰⁷ - make latch **counter** protected for deriving latch in hpxmp
- PR #3831²³⁰⁸ - Fix CircleCI config for modules
- PR #3830²³⁰⁹ - minor fix: option HPX_WITH_TEST was not working correctly
- PR #3828²³¹⁰ - Avoid for binaries that depend on HPX to directly link against internal modules
- PR #3827²³¹¹ - Adding shortcut for `hpx::get_ptr<>(sync, id)` for a local, non-migratable objects
- PR #3826²³¹² - Fix and update modules documentation
- PR #3825²³¹³ - Updating default APEX version to 2.1.3 with HPX
- PR #3823²³¹⁴ - Fix pkgconfig libs handling
- PR #3822²³¹⁵ - Change includes in `hpx_wrap.cpp` to more specific includes
- PR #3821²³¹⁶ - Disable barrier_3792 test when networking is disabled
- PR #3820²³¹⁷ - Assorted CMake fixes
- PR #3815²³¹⁸ - Removing left-over debug output
- PR #3814²³¹⁹ - Allow setting default scheduler mode via the configuration database
- PR #3813²³²⁰ - Make the deprecation warnings issued by the old pp headers optional
- PR #3812²³²¹ - Windows requires to handle symlinks to directories differently from those linking files

²²⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3846>

²³⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3844>

²³⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3843>

²³⁰² <https://github.com/STELLAR-GROUP/hpx/pull/3842>

²³⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/3840>

²³⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3837>

²³⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3835>

²³⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3834>

²³⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3833>

²³⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3831>

²³⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3830>

²³¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3828>

²³¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3827>

²³¹² <https://github.com/STELLAR-GROUP/hpx/pull/3826>

²³¹³ <https://github.com/STELLAR-GROUP/hpx/pull/3825>

²³¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3823>

²³¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3822>

²³¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3821>

²³¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3820>

²³¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3815>

²³¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3814>

²³²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3813>

²³²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3812>

- PR #3811²³²² - Clean up PP module and library skeleton
- PR #3806²³²³ - Moving include path configuration to before APEX
- PR #3804²³²⁴ - Fix latch
- PR #3803²³²⁵ - Update hpxcxx to look at lib64 and use python3
- PR #3802²³²⁶ - Numa binding allocator
- PR #3801²³²⁷ - Remove duplicated includes
- PR #3800²³²⁸ - Attempt to fix Posix context switching after lazy init changes
- PR #3798²³²⁹ - count and count_if accepts different iterator types
- PR #3797²³³⁰ - Adding a couple of `override` keywords to overloaded virtual functions
- PR #3796²³³¹ - Re-enable testing all schedulers in `shutdown_suspended_test`
- PR #3795²³³² - Change `std::terminate` to `std::abort` in SIGSEGV handler
- PR #3794²³³³ - Fixing #3792
- PR #3793²³³⁴ - Extending `migrate_polymorphic_component` unit test
- PR #3791²³³⁵ - Change `throw()` to `noexcept`
- PR #3790²³³⁶ - Remove deprecated options for 1.3.0 release
- PR #3789²³³⁷ - Remove Boost filesystem compatibility header
- PR #3788²³³⁸ - Disabled even more spots that should not execute if networking is disabled
- PR #3787²³³⁹ - Bump minimal boost supported version to 1.61.0
- PR #3786²³⁴⁰ - Bump minimum required versions for 1.3.0 release
- PR #3785²³⁴¹ - Explicitly set number of jobs for all ninja invocations on CircleCI
- PR #3784²³⁴² - Fix leak and address sanitizer problems
- PR #3783²³⁴³ - Disabled even more spots that should not execute if networking is disabled
- PR #3782²³⁴⁴ - Cherry-picked tuple and `thread_init_data` fixes from #3701

²³²² <https://github.com/STELLAR-GROUP/hpx/pull/3811>²³²³ <https://github.com/STELLAR-GROUP/hpx/pull/3806>²³²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3804>²³²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3803>²³²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3802>²³²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3801>²³²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3800>²³²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3798>²³³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3797>²³³¹ <https://github.com/STELLAR-GROUP/hpx/pull/3796>2332 <https://github.com/STELLAR-GROUP/hpx/pull/3795>2333 <https://github.com/STELLAR-GROUP/hpx/pull/3794>2334 <https://github.com/STELLAR-GROUP/hpx/pull/3793>2335 <https://github.com/STELLAR-GROUP/hpx/pull/3791>2336 <https://github.com/STELLAR-GROUP/hpx/pull/3790>2337 <https://github.com/STELLAR-GROUP/hpx/pull/3789>2338 <https://github.com/STELLAR-GROUP/hpx/pull/3788>2339 <https://github.com/STELLAR-GROUP/hpx/pull/3787>2340 <https://github.com/STELLAR-GROUP/hpx/pull/3786>2341 <https://github.com/STELLAR-GROUP/hpx/pull/3785>2342 <https://github.com/STELLAR-GROUP/hpx/pull/3784>2343 <https://github.com/STELLAR-GROUP/hpx/pull/3783>2344 <https://github.com/STELLAR-GROUP/hpx/pull/3782>

- PR #3781²³⁴⁵ - Fix generic context coroutines after lazy stack allocation changes
- PR #3780²³⁴⁶ - Rename hello world examples
- PR #3776²³⁴⁷ - Sort algorithms now use the supplied chunker to determine the required minimal chunk size
- PR #3775²³⁴⁸ - Disable Boost auto-linking
- PR #3774²³⁴⁹ - Tag and push stable builds
- PR #3773²³⁵⁰ - Enable migration of polymorphic components
- PR #3771²³⁵¹ - Fix link to stackoverflow in documentation
- PR #3770²³⁵² - Replacing constexpr if in brace-serialization code
- PR #3769²³⁵³ - Fix SIGSEGV handler
- PR #3768²³⁵⁴ - Adding flags to scheduler allowing to control thread stealing and idle back-off
- PR #3767²³⁵⁵ - Fix help formatting in hpxrun.py
- PR #3765²³⁵⁶ - Fix a couple of bugs in the thread test
- PR #3764²³⁵⁷ - Workaround for SFINAE regression in msvc14.2
- PR #3762²³⁵⁸ - Prevent MSVC from prematurely instantiating things
- PR #3761²³⁵⁹ - Update python scripts to work with python 3
- PR #3760²³⁶⁰ - Fix callable vtable for GCC4.9
- PR #3759²³⁶¹ - Rename PAGE_SIZE to PAGE_SIZE_ because AppleClang
- PR #3755²³⁶² - Making sure locks are not held during suspension
- PR #3754²³⁶³ - Disable more code if networking is not available/not enabled
- PR #3752²³⁶⁴ - Move util::format implementation to source file
- PR #3751²³⁶⁵ - Fixing problems with lcos::barrier and iostreams
- PR #3750²³⁶⁶ - Change error message to take into account use_guard_page setting
- PR #3749²³⁶⁷ - Fix lifetime problem in run_as_hpx_thread

²³⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3781>

²³⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3780>

²³⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3776>

²³⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3775>

²³⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3774>

²³⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3773>

²³⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3771>

²³⁵² <https://github.com/STELLAR-GROUP/hpx/pull/3770>

²³⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3769>

²³⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3768>

²³⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3767>

²³⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3765>

²³⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3764>

²³⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3762>

²³⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3761>

²³⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3760>

²³⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3759>

²³⁶² <https://github.com/STELLAR-GROUP/hpx/pull/3755>

²³⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/3754>

²³⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3752>

²³⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3751>

²³⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3750>

²³⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3749>

- PR #3748²³⁶⁸ - Fixed unusable behavior of the clang code analyzer.
- PR #3747²³⁶⁹ - Added PMIX_RANK to the defaults of HPX_WITH_PARCELPORT_MPI_ENV.
- PR #3745²³⁷⁰ - Introduced cache_aligned_data and cache_line_data helper structure
- PR #3742²³⁷¹ - Remove more unused functionality from util/logging
- PR #3740²³⁷² - Fix includes in partitioned vector tests
- PR #3739²³⁷³ - More fixes to make sure that std::flush really flushes all output
- PR #3737²³⁷⁴ - Fix potential shutdown problems
- PR #3736²³⁷⁵ - Fix guided_pool_executor after dataflow changes caused compilation fail
- PR #3734²³⁷⁶ - Limiting executor
- PR #3732²³⁷⁷ - More constrained bound constructors
- PR #3730²³⁷⁸ - Attempt to fix deadlocks during component loading
- PR #3729²³⁷⁹ - Add latch member function count_up and reset, requested by hpxMP
- PR #3728²³⁸⁰ - Send even empty buffers on hpx::endl and hpx::flush
- PR #3727²³⁸¹ - Adding example demonstrating how to customize the memory management for a component
- PR #3726²³⁸² - Adding support for passing command line options through the HPX_COMMANDLINE_OPTIONS environment variable
- PR #3722²³⁸³ - Document known broken OpenMPI builds
- PR #3716²³⁸⁴ - Add barrier reset function, requested by hpxMP for reusing barrier
- PR #3715²³⁸⁵ - More work on functions and vtables
- PR #3714²³⁸⁶ - Generate single-page HTML, PDF, manpage from documentation
- PR #3713²³⁸⁷ - Updating default APEX version to 2.1.2
- PR #3712²³⁸⁸ - Update release procedure
- PR #3710²³⁸⁹ - Fix the C++11 build, after #3704
- PR #3709²³⁹⁰ - Move some component_registry functionality to source file

²³⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3748>

²³⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3747>

²³⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3745>

²³⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3742>

²³⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3740>

²³⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3739>

²³⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3737>

²³⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3736>

²³⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3734>

²³⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3732>

²³⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3730>

²³⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3729>

²³⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3728>

²³⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/3727>

²³⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3726>

²³⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/3722>

²³⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3716>

²³⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3715>

²³⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3714>

²³⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3713>

²³⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3712>

²³⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3710>

²³⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3709>

- PR #3708²³⁹¹ - Ignore all locks while handling assertions
- PR #3707²³⁹² - Remove obsolete hpx runtime executable
- PR #3705²³⁹³ - Fix and simplify `make_ready_future` overload sets
- PR #3704²³⁹⁴ - Reduce use of binders
- PR #3703²³⁹⁵ - Ini
- PR #3702²³⁹⁶ - Fixing CUDA compiler errors
- PR #3700²³⁹⁷ - Added `barrier::increment` function to increase total number of thread
- PR #3697²³⁹⁸ - One more attempt to fix migration...
- PR #3694²³⁹⁹ - Fixing component migration
- PR #3693²⁴⁰⁰ - Print thread state when getting disallowed value in `set_thread_state`
- PR #3692²⁴⁰¹ - Only disable `constexpr` with clang-cuda, not nvcc+gcc
- PR #3691²⁴⁰² - Link with `libspsc++` if needed for `thread_local`
- PR #3690²⁴⁰³ - Remove thousands separators in `set_operations_3442` to comply with C++11
- PR #3688²⁴⁰⁴ - Decouple serialization from function vtables
- PR #3687²⁴⁰⁵ - Fix a couple of test failures
- PR #3686²⁴⁰⁶ - Make sure `tests.unit.build` are run after install on CircleCI
- PR #3685²⁴⁰⁷ - Revise quickstart CMakeLists.txt explanation
- PR #3684²⁴⁰⁸ - Provide concept emulation for Ranges-TS concepts
- PR #3683²⁴⁰⁹ - Ignore uninitialized chunks
- PR #3682²⁴¹⁰ - Ignore uninitialized chunks. Check proper indices.
- PR #3680²⁴¹¹ - Ignore uninitialized chunks. Check proper range indices
- PR #3679²⁴¹² - Simplify basic action implementations
- PR #3678²⁴¹³ - Making sure `HPX_HAVE_LIBATOMIC` is unset before checking

²³⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3708>

²³⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3707>

²³⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3705>

²³⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3704>

²³⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3703>

²³⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3702>

²³⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3700>

²³⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3697>

²³⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3694>

²⁴⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3693>

²⁴⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3692>

²⁴⁰² <https://github.com/STELLAR-GROUP/hpx/pull/3691>

²⁴⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/3690>

²⁴⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3688>

²⁴⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3687>

²⁴⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3686>

²⁴⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3685>

²⁴⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3684>

²⁴⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3683>

²⁴¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3682>

²⁴¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3680>

²⁴¹² <https://github.com/STELLAR-GROUP/hpx/pull/3679>

²⁴¹³ <https://github.com/STELLAR-GROUP/hpx/pull/3678>

- PR #3677²⁴¹⁴ - Fix generated full version number to be usable in expressions
- PR #3674²⁴¹⁵ - Reduce functional utilities call depth
- PR #3672²⁴¹⁶ - Change new build system to use existing macros related to pseudo dependencies
- PR #3669²⁴¹⁷ - Remove indirection in `function_ref` when thread description is disabled
- PR #3668²⁴¹⁸ - Unbreaking `async_*cb*` tests
- PR #3667²⁴¹⁹ - Generate `version.hpp`
- PR #3665²⁴²⁰ - Enabling MPI parcelport for gitlab runners
- PR #3664²⁴²¹ - making clang-tidy work properly again
- PR #3662²⁴²² - Attempt to fix exception handling
- PR #3661²⁴²³ - Move `lcos::latch` to source file
- PR #3660²⁴²⁴ - Fix accidentally explicit `gid_type` default constructor
- PR #3659²⁴²⁵ - Parallel executor latch
- PR #3658²⁴²⁶ - Fixing `execution_parameters`
- PR #3657²⁴²⁷ - Avoid dangling references in `wait_all`
- PR #3656²⁴²⁸ - Avoiding lifetime problems with `sync_put_parcel`
- PR #3655²⁴²⁹ - Fixing `nullptr` dereference inside of function
- PR #3652²⁴³⁰ - Attempt to fix `thread_map_type` definition with C++11
- PR #3650²⁴³¹ - Allowing for end iterator being different from begin iterator
- PR #3649²⁴³² - Added architecture identification to `cmake` to be able to detect timestamp support
- PR #3645²⁴³³ - Enabling sanitizers on gitlab runner
- PR #3644²⁴³⁴ - Attempt to tackle timeouts during startup
- PR #3642²⁴³⁵ - Cleanup parallel partitioners
- PR #3640²⁴³⁶ - Dataflow now works with functions that return a reference

²⁴¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3677>

²⁴¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3674>

²⁴¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3672>

²⁴¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3669>

²⁴¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3668>

²⁴¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3667>

²⁴²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3665>

²⁴²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3664>

²⁴²² <https://github.com/STELLAR-GROUP/hpx/pull/3662>

²⁴²³ <https://github.com/STELLAR-GROUP/hpx/pull/3661>

²⁴²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3660>

²⁴²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3659>

²⁴²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3658>

²⁴²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3657>

²⁴²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3656>

²⁴²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3655>

²⁴³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3652>

²⁴³¹ <https://github.com/STELLAR-GROUP/hpx/pull/3650>

²⁴³² <https://github.com/STELLAR-GROUP/hpx/pull/3649>

²⁴³³ <https://github.com/STELLAR-GROUP/hpx/pull/3645>

²⁴³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3644>

²⁴³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3642>

²⁴³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3640>

- PR #3637²⁴³⁷ - Merging the executor-enabled overloads of `shared_future<>::then`
- PR #3633²⁴³⁸ - Replace deprecated boost endian macros
- PR #3632²⁴³⁹ - Add instructions on getting HPX to documentation
- PR #3631²⁴⁴⁰ - Simplify parcel creation
- PR #3630²⁴⁴¹ - Small additions and fixes to release procedure
- PR #3629²⁴⁴² - Modular pp
- PR #3627²⁴⁴³ - Implement `util::function_ref`
- PR #3626²⁴⁴⁴ - Fix cancelable_action_client example
- PR #3625²⁴⁴⁵ - Added automatic serialization for simple structs (see #3034)
- PR #3624²⁴⁴⁶ - Updating the default order of priority for `thread_description`
- PR #3621²⁴⁴⁷ - Update copyright year and other small formatting fixes
- PR #3620²⁴⁴⁸ - Adding support for gitlab runner
- PR #3619²⁴⁴⁹ - Store debug logs and core dumps on CircleCI
- PR #3618²⁴⁵⁰ - Various optimizations
- PR #3617²⁴⁵¹ - Fix link to the gpg key (#2)
- PR #3615²⁴⁵² - Fix unused variable warnings with networking off
- PR #3614²⁴⁵³ - Restructuring counter data in scheduler to reduce false sharing
- PR #3613²⁴⁵⁴ - Adding support for gitlab runners
- PR #3610²⁴⁵⁵ - Don't wait for `stop_condition` in main thread
- PR #3608²⁴⁵⁶ - Add inline keyword to `invalid_thread_id` definition for nvcc
- PR #3607²⁴⁵⁷ - Adding configuration key that allows one to explicitly add a directory to the component search path
- PR #3606²⁴⁵⁸ - Add nvcc to exclude constexpress since it is not supported by nvcc
- PR #3605²⁴⁵⁹ - Add inline to definition of checkpoint stream operators to fix link error

²⁴³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3637>

²⁴³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3633>

²⁴³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3632>

²⁴⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3631>

²⁴⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/3630>

²⁴⁴² <https://github.com/STELLAR-GROUP/hpx/pull/3629>

²⁴⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/3627>

²⁴⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3626>

²⁴⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3625>

²⁴⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3624>

²⁴⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3621>

²⁴⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3620>

²⁴⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3619>

²⁴⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3618>

²⁴⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3617>

²⁴⁵² <https://github.com/STELLAR-GROUP/hpx/pull/3615>

²⁴⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3614>

²⁴⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3613>

²⁴⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3610>

²⁴⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3608>

²⁴⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3607>

²⁴⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3606>

²⁴⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3605>

- PR #3604²⁴⁶⁰ - Use format for string formatting
- PR #3603²⁴⁶¹ - Improve the error message for using to less MAX_CPU_COUNT
- PR #3602²⁴⁶² - Improve the error message for to small values of MAX_CPU_COUNT
- PR #3600²⁴⁶³ - Parallel executor aggregated
- PR #3599²⁴⁶⁴ - Making sure networking is disabled for default one-locality-runs
- PR #3596²⁴⁶⁵ - Store thread exit functions in `forward_list` instead of `deque` to avoid allocations
- PR #3590²⁴⁶⁶ - Fix typo/mistake in thread queue `cleanup_terminated`
- PR #3588²⁴⁶⁷ - Fix formatting errors in `launching_and_configuring_hpx_applications.rst`
- PR #3586²⁴⁶⁸ - Make bind propagate value category
- PR #3585²⁴⁶⁹ - Extend Cmake for building hpx as distribution packages (refs #3575)
- PR #3584²⁴⁷⁰ - Untangle function storage from object pointer
- PR #3582²⁴⁷¹ - Towards Modularized HPX
- PR #3580²⁴⁷² - Remove extra `||` in `merge.hpp`
- PR #3577²⁴⁷³ - Partially revert “Remove vtable empty flag”
- PR #3576²⁴⁷⁴ - Make sure empty startup/shutdown functions are not being used
- PR #3574²⁴⁷⁵ - Make sure DATAPAR settings are conveyed to depending projects
- PR #3573²⁴⁷⁶ - Make sure HPX is usable with latest released version of Vc (V1.4.1)
- PR #3572²⁴⁷⁷ - Adding test ensuring ticket 3565 is fixed
- PR #3571²⁴⁷⁸ - Make empty `[unique_]` function vtable non-dependent
- PR #3566²⁴⁷⁹ - Fix compilation with dynamic bitset for CPU masks
- PR #3563²⁴⁸⁰ - Drop `util::[unique_]` function target_type
- PR #3562²⁴⁸¹ - Removing the target suffixes
- PR #3561²⁴⁸² - Replace executor traits return type deduction (keep non-SFINAE)

²⁴⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3604>

²⁴⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3603>

²⁴⁶² <https://github.com/STELLAR-GROUP/hpx/pull/3602>

²⁴⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/3600>

²⁴⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3599>

²⁴⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3596>

²⁴⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3590>

²⁴⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3588>

²⁴⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3586>

²⁴⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3585>

²⁴⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3584>

²⁴⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3582>

²⁴⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3580>

²⁴⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3577>

²⁴⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3576>

²⁴⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3574>

²⁴⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3573>

²⁴⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3572>

²⁴⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3571>

²⁴⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3566>

²⁴⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3563>

²⁴⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/3562>

²⁴⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3561>

- PR #3557²⁴⁸³ - Replace the last usages of boost::atomic
- PR #3556²⁴⁸⁴ - Replace boost::scoped_array with std::unique_ptr
- PR #3552²⁴⁸⁵ - (Re)move APEX readme
- PR #3548²⁴⁸⁶ - Replace boost::scoped_ptr with std::unique_ptr
- PR #3547²⁴⁸⁷ - Remove last use of Boost.Signals2
- PR #3544²⁴⁸⁸ - Post 1.2.0 version bumps
- PR #3543²⁴⁸⁹ - added Ubuntu dependency list to readme
- PR #3531²⁴⁹⁰ - Warnings, warnings...
- PR #3527²⁴⁹¹ - Add CircleCI filter for building all tags
- PR #3525²⁴⁹² - Segmented algorithms
- PR #3517²⁴⁹³ - Replace boost::regex with C++11 <regex>
- PR #3514²⁴⁹⁴ - Cleaning up the build system
- PR #3505²⁴⁹⁵ - Fixing type attribute warning for transfer_action
- PR #3504²⁴⁹⁶ - Add support for rpm packaging
- PR #3499²⁴⁹⁷ - Improving spinlock pools
- PR #3498²⁴⁹⁸ - Remove thread specific ptr
- PR #3486²⁴⁹⁹ - Fix comparison for expect_connecting_localities config entry
- PR #3469²⁵⁰⁰ - Enable (existing) code for extracting stack pointer on Power platform

2.10.10 HPX V1.2.1 (Feb 19, 2019)

General changes

This is a bugfix release. It contains the following changes:

- Fix compilation on ARM, s390x and 32-bit architectures.
- Fix a critical bug in the future implementation.
- Fix several problems in the CMake configuration which affects external projects.

²⁴⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/3557>

²⁴⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3556>

²⁴⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3552>

²⁴⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3548>

²⁴⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3547>

²⁴⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3544>

²⁴⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3543>

²⁴⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3531>

²⁴⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3527>

²⁴⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3525>

²⁴⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3517>

²⁴⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3514>

²⁴⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3505>

²⁴⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3504>

²⁴⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3499>

²⁴⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3498>

²⁴⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3486>

²⁵⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3469>

- Add support for Boost 1.69.0.

Closed issues

- Issue #3638²⁵⁰¹ - Build HPX 1.2 with boost 1.69
- Issue #3635²⁵⁰² - Non-deterministic crashing on Stampede2
- Issue #3550²⁵⁰³ - 1>e:000workhpxsrcthrow_exception.cpp(54): error C2440: ‘<function-style-cast>’: cannot convert from ‘boost::system::error_code’ to ‘hpx::exception’
- Issue #3549²⁵⁰⁴ - HPX 1.2.0 does not build on i686, but release candidate did
- Issue #3511²⁵⁰⁵ - Build on s390x fails
- Issue #3509²⁵⁰⁶ - Build on armv7l fails

Closed pull requests

- PR #3695²⁵⁰⁷ - Don’t install CMake templates and packaging files
- PR #3666²⁵⁰⁸ - Fixing yet another race in future_data
- PR #3663²⁵⁰⁹ - Fixing race between setting and getting the value inside future_data
- PR #3648²⁵¹⁰ - Adding timestamp option for S390x platform
- PR #3647²⁵¹¹ - Blind attempt to fix warnings issued by gcc V9
- PR #3611²⁵¹² - Include GNUInstallDirs earlier to have it available for subdirectories
- PR #3595²⁵¹³ - Use GNUInstallDirs lib path in pkgconfig config file
- PR #3593²⁵¹⁴ - Add include(GNUInstallDirs) to HPXMacros.cmake
- PR #3591²⁵¹⁵ - Fix compilation error on arm7 architecture. Compiles and runs on Fedora 29 on Pi 3.
- PR #3558²⁵¹⁶ - Adding constructor *exception(boost::system::error_code const&)*
- PR #3555²⁵¹⁷ - cmake: make install locations configurable
- PR #3551²⁵¹⁸ - Fix uint64_t causing compilation fail on i686

2501 <https://github.com/STELLAR-GROUP/hpx/issues/3638>

2502 <https://github.com/STELLAR-GROUP/hpx/issues/3635>

2503 <https://github.com/STELLAR-GROUP/hpx/issues/3550>

2504 <https://github.com/STELLAR-GROUP/hpx/issues/3549>

2505 <https://github.com/STELLAR-GROUP/hpx/issues/3511>

2506 <https://github.com/STELLAR-GROUP/hpx/issues/3509>

2507 <https://github.com/STELLAR-GROUP/hpx/pull/3695>

2508 <https://github.com/STELLAR-GROUP/hpx/pull/3666>

2509 <https://github.com/STELLAR-GROUP/hpx/pull/3663>

2510 <https://github.com/STELLAR-GROUP/hpx/pull/3648>

2511 <https://github.com/STELLAR-GROUP/hpx/pull/3647>

2512 <https://github.com/STELLAR-GROUP/hpx/pull/3611>

2513 <https://github.com/STELLAR-GROUP/hpx/pull/3595>

2514 <https://github.com/STELLAR-GROUP/hpx/pull/3593>

2515 <https://github.com/STELLAR-GROUP/hpx/pull/3591>

2516 <https://github.com/STELLAR-GROUP/hpx/pull/3558>

2517 <https://github.com/STELLAR-GROUP/hpx/pull/3555>

2518 <https://github.com/STELLAR-GROUP/hpx/pull/3551>

2.10.11 HPX V1.2.0 (Nov 12, 2018)

General changes

Here are some of the main highlights and changes for this release:

- Thanks to the work of our Google Summer of Code student, Nikunj Gupta, we now have a new implementation of `hpx_main.hpp` on supported platforms (Linux, BSD and MacOS). This is intended to be a less fragile drop-in replacement for the old implementation relying on preprocessor macros. The new implementation does not require changes if you are using the [CMake²⁵¹⁹](#) or `pkg-config`. The old behaviour can be restored by setting `HPX_WITH_DYNAMIC_HPX_MAIN=OFF` during [CMake²⁵²⁰](#) configuration. The implementation on Windows is unchanged.
- We have added functionality to allow passing scheduling hints to our schedulers. These will allow us to create executors that for example target a specific NUMA domain or allow for *HPX* threads to be pinned to a particular worker thread.
- We have significantly improved the performance of our futures implementation by making the shared state atomic.
- We have replaced Boostbook by Sphinx for our documentation. This means the documentation is easier to navigate with built-in search and table of contents. We have also added a quick start section and restructured the documentation to be easier to follow for new users.
- We have added a new option to the `--hpx:threads` command line option. It is now possible to use `cores` to tell *HPX* to only use one worker thread per core, unlike the existing option `all` which uses one worker thread per processing unit (processing unit can be a hyperthread if hyperthreads are available). The default value of `--hpx:threads` has also been changed to `cores` as this leads to better performance in most cases.
- All command line options can now be passed alongside configuration options when initializing *HPX*. This means that some options that were previously only available on the command line can now be set as configuration options.
- *HPXMP* is a portable, scalable, and flexible application programming interface using the OpenMP specification that supports multi-platform shared memory multiprocessing programming in C and C++. *HPXMP* can be enabled within *HPX* by setting `DHPX_WITH_HPXMP=ON` during [CMake²⁵²¹](#) configuration.
- Two new performance counters were added for measuring the time spent doing background work. `/threads/time/background-work-duration` returns the time spent doing background on a given thread or locality, while `/threads/time/background-overhead` returns the fraction of time spent doing background work with respect to the overall time spent running the scheduler. The new performance counters are disabled by default and can be turned on by setting `HPX_WITH_BACKGROUND_THREAD_COUNTERS=ON` during [CMake²⁵²²](#) configuration.
- The idling behaviour of *HPX* has been tweaked to allow for faster idling. This is useful in interactive applications where the *HPX* worker threads may not have work all the time. This behaviour can be tweaked and turned off as before with `HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF=OFF` during [CMake²⁵²³](#) configuration.
- It is now possible to register callback functions for *HPX* worker thread events. Callbacks can be registered for starting and stopping worker threads, and for when errors occur.

²⁵¹⁹ <https://www.cmake.org>

²⁵²⁰ <https://www.cmake.org>

²⁵²¹ <https://www.cmake.org>

²⁵²² <https://www.cmake.org>

²⁵²³ <https://www.cmake.org>

Breaking changes

- The implementation of `hpx_main.hpp` has changed. If you are using custom Makefiles you will need to make changes. Please see the documentation on [using Makefiles](#) for more details.
- The default value of `--hpx:threads` has changed from `all` to `cores`. The new option `cores` only starts one worker thread per core.
- We have dropped support for Boost 1.56 and 1.57. The minimal version of Boost we now test is 1.58.
- Our `boost::format`-based formatting implementation has been revised and replaced with a custom implementation. This changes the formatting syntax and requires changes if you are relying on `hpx::util::format` or `hpx::util::format_to`. The pull request for this change contains more information: [PR #3266](#)²⁵²⁴.
- The following deprecated options have now been completely removed: `HPX_WITH_ASYNC_FUNCTION_COMPATIBILITY`, `HPX_WITH_LOCAL_DATAFLOW`, `HPX_WITH_GENERIC_EXECUTION_POLICY`, `HPX_WITH_BOOST_CHRONO_COMPATIBILITY`, `HPX_WITH_EXECUTOR_COMPATIBILITY`, `HPX_WITH_EXECUTION_POLICY_COMPATIBILITY`, and `HPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY`.

Closed issues

- Issue #3538²⁵²⁵ - numa handling incorrect for hwloc 2
- Issue #3533²⁵²⁶ - Cmake version 3.5.1does not work (git ff26b35 2018-11-06)
- Issue #3526²⁵²⁷ - Failed building hpx-1.2.0-rc1 on Ubuntu16.04 x86-64 Virtualbox VM
- Issue #3512²⁵²⁸ - Build on aarch64 fails
- Issue #3475²⁵²⁹ - HPX fails to link if the MPI parcelport is enabled
- Issue #3462²⁵³⁰ - CMake configuration shows a minor and inconsequential failure to create a symlink
- Issue #3461²⁵³¹ - Compilation Problems with the most recent Clang
- Issue #3460²⁵³² - Deadlock when create_partitioner fails (assertion fails) in debug mode
- Issue #3455²⁵³³ - HPX build failing with HWLOC errors on POWER8 with hwloc 1.8
- Issue #3438²⁵³⁴ - HPX no longer builds on IBM POWER8
- Issue #3426²⁵³⁵ - hpx build failed on MacOS
- Issue #3424²⁵³⁶ - CircleCI builds broken for forked repositories
- Issue #3422²⁵³⁷ - Benchmarks in tests.performance.local are not run nightly

²⁵²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3266>

²⁵²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3538>

²⁵²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3533>

²⁵²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3526>

²⁵²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3512>

²⁵²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3475>

²⁵³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3462>

²⁵³¹ <https://github.com/STELLAR-GROUP/hpx/issues/3461>

²⁵³² <https://github.com/STELLAR-GROUP/hpx/issues/3460>

²⁵³³ <https://github.com/STELLAR-GROUP/hpx/issues/3455>

²⁵³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3438>

²⁵³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3426>

²⁵³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3424>

²⁵³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3422>

- Issue #3408²⁵³⁸ - CMake Targets for HPX
- Issue #3399²⁵³⁹ - processing unit out of bounds
- Issue #3395²⁵⁴⁰ - Floating point bug in hpx/runtime/threads/policies/scheduler_base.hpp
- Issue #3378²⁵⁴¹ - compile error with lcos::communicator
- Issue #3376²⁵⁴² - Failed to build HPX with APEX using clang
- Issue #3366²⁵⁴³ - Adapted Safe_Object example fails for -hpx:threads > 1
- Issue #3360²⁵⁴⁴ - Segmentation fault when passing component id as parameter
- Issue #3358²⁵⁴⁵ - HPX runtime hangs after multiple (~thousands) start-stop sequences
- Issue #3352²⁵⁴⁶ - Support TCP provider in libfabric ParcelPort
- Issue #3342²⁵⁴⁷ - undefined reference to __atomic_load_16
- Issue #3339²⁵⁴⁸ - setting command line options/flags from init cfg is not obvious
- Issue #3325²⁵⁴⁹ - AGAS migrates components prematurely
- Issue #3321²⁵⁵⁰ - hpx bad_parameter handling is awful
- Issue #3318²⁵⁵¹ - Benchmarks fail to build with C++11
- Issue #3304²⁵⁵² - hpx::threads::run_as_hpx_thread does not properly handle exceptions
- Issue #3300²⁵⁵³ - Setting pu step or offset results in no threads in default pool
- Issue #3297²⁵⁵⁴ - Crash with APEX when running Phylanx lra_csv with > 1 thread
- Issue #3296²⁵⁵⁵ - Building HPX with APEX configuration gives compiler warnings
- Issue #3290²⁵⁵⁶ - make tests failing at hello_world_component
- Issue #3285²⁵⁵⁷ - possible compilation error when “using namespace std;” is defined before including “hpx” headers files
- Issue #3280²⁵⁵⁸ - HPX fails on OSX
- Issue #3272²⁵⁵⁹ - CircleCI does not upload generated docker image any more
- Issue #3270²⁵⁶⁰ - Error when compiling CUDA examples

²⁵³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3408>

²⁵³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3399>

²⁵⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3395>

²⁵⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/3378>

²⁵⁴² <https://github.com/STELLAR-GROUP/hpx/issues/3376>

²⁵⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/3366>

²⁵⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3360>

²⁵⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3358>

²⁵⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3352>

²⁵⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3342>

²⁵⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3339>

²⁵⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3325>

²⁵⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3321>

²⁵⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/3318>

²⁵⁵² <https://github.com/STELLAR-GROUP/hpx/issues/3304>

²⁵⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/3300>

²⁵⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3297>

²⁵⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3296>

²⁵⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3290>

²⁵⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3285>

²⁵⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3280>

²⁵⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3272>

²⁵⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3270>

- Issue #3267²⁵⁶¹ - tests.unit.host_.block_allocator fails occasionally
- Issue #3264²⁵⁶² - Possible move to Sphinx for documentation
- Issue #3263²⁵⁶³ - Documentation improvements
- Issue #3259²⁵⁶⁴ - set_parcel_write_handler test fails occasionally
- Issue #3258²⁵⁶⁵ - Links to source code in documentation are broken
- Issue #3247²⁵⁶⁶ - Rare tests.unit.host_.block_allocator test failure on 1.1.0-rc1
- Issue #3244²⁵⁶⁷ - Slowing down and speeding up an interval_timer
- Issue #3215²⁵⁶⁸ - Cannot build both tests and examples on MSVC with pseudo-dependencies enabled
- Issue #3195²⁵⁶⁹ - Unnecessary customization point route causing performance penalty
- Issue #3088²⁵⁷⁰ - A strange thing in parallel::sort.
- Issue #2650²⁵⁷¹ - libfabric support for passive endpoints
- Issue #1205²⁵⁷² - TSS is broken

Closed pull requests

- PR #3542²⁵⁷³ - Fix numa lookup from pu when using hwloc 2.x
- PR #3541²⁵⁷⁴ - Fixing the build system of the MPI parcelport
- PR #3540²⁵⁷⁵ - Updating HPX people section
- PR #3539²⁵⁷⁶ - Splitting test to avoid OOM on CircleCI
- PR #3537²⁵⁷⁷ - Fix guided exec
- PR #3536²⁵⁷⁸ - Updating grants which support the LSU team
- PR #3535²⁵⁷⁹ - Fix hiding of docker credentials
- PR #3534²⁵⁸⁰ - Fixing #3533
- PR #3532²⁵⁸¹ - fixing minor doc typo --hpx:print-counter-at arg
- PR #3530²⁵⁸² - Changing APEX default tag to v2.1.0

²⁵⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/3267>

²⁵⁶² <https://github.com/STELLAR-GROUP/hpx/issues/3264>

²⁵⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/3263>

²⁵⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3259>

²⁵⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3258>

²⁵⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3247>

²⁵⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3244>

²⁵⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3215>

²⁵⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3195>

²⁵⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3088>

²⁵⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/2650>

²⁵⁷² <https://github.com/STELLAR-GROUP/hpx/issues/1205>

²⁵⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3542>

²⁵⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3541>

²⁵⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3540>

²⁵⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3539>

²⁵⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3537>

²⁵⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3536>

²⁵⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3535>

²⁵⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3534>

²⁵⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/3532>

²⁵⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3530>

- PR #3529²⁵⁸³ - Remove leftover security options and documentation
- PR #3528²⁵⁸⁴ - Fix hwloc version check
- PR #3524²⁵⁸⁵ - Do not build guided pool examples with older GCC compilers
- PR #3523²⁵⁸⁶ - Fix logging regression
- PR #3522²⁵⁸⁷ - Fix more warnings
- PR #3521²⁵⁸⁸ - Fixing argument handling in induction and reduction clauses for parallel::for_loop
- PR #3520²⁵⁸⁹ - Remove docs symlink and versioned docs folders
- PR #3519²⁵⁹⁰ - hpxMP release
- PR #3518²⁵⁹¹ - Change all steps to use new docker image on CircleCI
- PR #3516²⁵⁹² - Drop usage of deprecated facilities removed in C++17
- PR #3515²⁵⁹³ - Remove remaining uses of Boost.TypeTraits
- PR #3513²⁵⁹⁴ - Fixing a CMake problem when trying to use libfabric
- PR #3508²⁵⁹⁵ - Remove memory_block component
- PR #3507²⁵⁹⁶ - Propagating the MPI compile definitions to all relevant targets
- PR #3503²⁵⁹⁷ - Update documentation colors and logo
- PR #3502²⁵⁹⁸ - Fix bogus `throws` bindings in scheduled_thread_pool_impl
- PR #3501²⁵⁹⁹ - Split parallel::remove_if tests to avoid OOM on CircleCI
- PR #3500²⁶⁰⁰ - Support NONAMEPREFIX in add_hpx_library()
- PR #3497²⁶⁰¹ - Note that cuda support requires cmake 3.9
- PR #3495²⁶⁰² - Fixing dataflow
- PR #3493²⁶⁰³ - Remove deprecated options for 1.2.0 part 2
- PR #3492²⁶⁰⁴ - Add CUDA_LINK_LIBRARIES_KEYWORD to allow PRIVATE keyword in linkage t...
- PR #3491²⁶⁰⁵ - Changing Base docker image

²⁵⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/3529>

²⁵⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3528>

²⁵⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3524>

²⁵⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3523>

²⁵⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3522>

²⁵⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3521>

²⁵⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3520>

²⁵⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3519>

²⁵⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3518>

²⁵⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3516>

²⁵⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3515>

²⁵⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3513>

²⁵⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3508>

²⁵⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3507>

²⁵⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3503>

²⁵⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3502>

²⁵⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3501>

²⁶⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3500>

²⁶⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3497>

²⁶⁰² <https://github.com/STELLAR-GROUP/hpx/pull/3495>

²⁶⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/3493>

²⁶⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3492>

²⁶⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3491>

- PR #3490²⁶⁰⁶ - Don't create tasks immediately with hpx::apply
- PR #3489²⁶⁰⁷ - Remove deprecated options for 1.2.0
- PR #3488²⁶⁰⁸ - Revert "Use BUILD_INTERFACE generator expression to fix cmake flag exports"
- PR #3487²⁶⁰⁹ - Revert "Fixing type attribute warning for transfer_action"
- PR #3485²⁶¹⁰ - Use BUILD_INTERFACE generator expression to fix cmake flag exports
- PR #3483²⁶¹¹ - Fixing type attribute warning for transfer_action
- PR #3481²⁶¹² - Remove unused variables
- PR #3480²⁶¹³ - Towards a more lightweight transfer action
- PR #3479²⁶¹⁴ - Fix FLAGS - Use correct version of target_compile_options
- PR #3478²⁶¹⁵ - Making sure the application's exit code is properly propagated back to the OS
- PR #3476²⁶¹⁶ - Don't print docker credentials as part of the environment.
- PR #3473²⁶¹⁷ - Fixing invalid cmake code if no jemalloc prefix was given
- PR #3472²⁶¹⁸ - Attempting to work around recent clang test compilation failures
- PR #3471²⁶¹⁹ - Enable jemalloc on windows
- PR #3470²⁶²⁰ - Updates readme
- PR #3468²⁶²¹ - Avoid hang if there is an exception thrown during startup
- PR #3467²⁶²² - Add compiler specific fallthrough attributes if C++17 attribute is not available
- PR #3466²⁶²³ - - bugfix : fix compilation with llvm-7.0
- PR #3465²⁶²⁴ - This patch adds various optimizations extracted from the thread_local_allocator work
- PR #3464²⁶²⁵ - Check for forked repos in CircleCI docker push step
- PR #3463²⁶²⁶ - - cmake : create the parent directory before symlinking
- PR #3459²⁶²⁷ - Remove unused/incomplete functionality from util/logging
- PR #3458²⁶²⁸ - Fix a problem with scope of CMAKE_CXX_FLAGS and hpx_add_compile_flag

²⁶⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3490>

²⁶⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3489>

²⁶⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3488>

²⁶⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3487>

²⁶¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3485>

²⁶¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3483>

²⁶¹² <https://github.com/STELLAR-GROUP/hpx/pull/3481>

²⁶¹³ <https://github.com/STELLAR-GROUP/hpx/pull/3480>

²⁶¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3479>

²⁶¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3478>

²⁶¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3476>

²⁶¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3473>

²⁶¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3472>

²⁶¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3471>

²⁶²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3470>

²⁶²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3468>

²⁶²² <https://github.com/STELLAR-GROUP/hpx/pull/3467>

²⁶²³ <https://github.com/STELLAR-GROUP/hpx/pull/3466>

²⁶²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3465>

²⁶²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3464>

²⁶²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3463>

²⁶²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3459>

²⁶²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3458>

- PR #3457²⁶²⁹ - Fixing more size_t -> int16_t (and similar) warnings
- PR #3456²⁶³⁰ - Add #ifdefs to topology.cpp to support old hwloc versions again
- PR #3454²⁶³¹ - Fixing warnings related to silent conversion of size_t -> int16_t
- PR #3451²⁶³² - Add examples as unit tests
- PR #3450²⁶³³ - Constexpr-fying bind and other functional facilities
- PR #3446²⁶³⁴ - Fix some thread suspension timeouts
- PR #3445²⁶³⁵ - Fix various warnings
- PR #3443²⁶³⁶ - Only enable service pool config options if pools are enabled
- PR #3441²⁶³⁷ - Fix missing closing brackets in documentation
- PR #3439²⁶³⁸ - Use correct MPI CXX libraries for MPI parcelport
- PR #3436²⁶³⁹ - Add projection function to find_*(and fix very bad bug)
- PR #3435²⁶⁴⁰ - Fixing 1205
- PR #3434²⁶⁴¹ - Fix threads cores
- PR #3433²⁶⁴² - Add Heise Online to release announcement list
- PR #3432²⁶⁴³ - Don't track task dependencies for distributed runs
- PR #3431²⁶⁴⁴ - Circle CI setting changes for hpxMP
- PR #3430²⁶⁴⁵ - Fix unused params warning
- PR #3429²⁶⁴⁶ - One thread per core
- PR #3428²⁶⁴⁷ - This suppresses a deprecation warning that is being issued by MSVC 19.15.26726
- PR #3427²⁶⁴⁸ - Fixes #3426
- PR #3425²⁶⁴⁹ - Use source cache and workspace between job steps on CircleCI
- PR #3421²⁶⁵⁰ - Add CDash timing output to future overhead test (for graphs)
- PR #3420²⁶⁵¹ - Add guided_pool_executor

²⁶²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3457>

²⁶³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3456>

²⁶³¹ <https://github.com/STELLAR-GROUP/hpx/pull/3454>

²⁶³² <https://github.com/STELLAR-GROUP/hpx/pull/3451>

²⁶³³ <https://github.com/STELLAR-GROUP/hpx/pull/3450>

²⁶³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3446>

²⁶³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3445>

²⁶³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3443>

²⁶³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3441>

²⁶³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3439>

²⁶³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3436>

²⁶⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3435>

²⁶⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/3434>

²⁶⁴² <https://github.com/STELLAR-GROUP/hpx/pull/3433>

²⁶⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/3432>

²⁶⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3431>

²⁶⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3430>

²⁶⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3429>

²⁶⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3428>

²⁶⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3427>

²⁶⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3425>

²⁶⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3421>

²⁶⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3420>

- PR #3419²⁶⁵² - Fix typo in CircleCI config
- PR #3418²⁶⁵³ - Add sphinx documentation
- PR #3415²⁶⁵⁴ - Scheduler NUMA hint and shared priority scheduler
- PR #3414²⁶⁵⁵ - Adding step to synchronize the APEX release
- PR #3413²⁶⁵⁶ - Fixing multiple defines of APEX_HAVE_HPX
- PR #3412²⁶⁵⁷ - Fixes linking with libhpx_wrap error with BSD and Windows based systems
- PR #3410²⁶⁵⁸ - Fix typo in CMakeLists.txt
- PR #3409²⁶⁵⁹ - Fix brackets and indentation in existing_performance_counters.qbk
- PR #3407²⁶⁶⁰ - Fix unused param and extra ; warnings emitted by gcc 8.x
- PR #3406²⁶⁶¹ - Adding thread local allocator and use it for future shared states
- PR #3405²⁶⁶² - Adding DHPX_HAVE_THREAD_LOCAL_STORAGE=ON to builds
- PR #3404²⁶⁶³ - fixing multiple definition of main() in linux
- PR #3402²⁶⁶⁴ - Allow debug option to be enabled only for Linux systems with dynamic main on
- PR #3401²⁶⁶⁵ - Fix cuda_future_helper.h when compiling with C++11
- PR #3400²⁶⁶⁶ - Fix floating point exception scheduler_base idle backoff
- PR #3398²⁶⁶⁷ - Atomic future state
- PR #3397²⁶⁶⁸ - Fixing code for older gcc versions
- PR #3396²⁶⁶⁹ - Allowing to register thread event functions (start/stop/error)
- PR #3394²⁶⁷⁰ - Fix small mistake in primary_namespace_server.cpp
- PR #3393²⁶⁷¹ - Explicitly instantiate configured schedulers
- PR #3392²⁶⁷² - Add performance counters background overhead and background work duration
- PR #3391²⁶⁷³ - Adapt integration of HPXMP to latest build system changes
- PR #3390²⁶⁷⁴ - Make AGAS measurements optional

²⁶⁵² <https://github.com/STELLAR-GROUP/hpx/pull/3419>

²⁶⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3418>

²⁶⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3415>

²⁶⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3414>

²⁶⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3413>

²⁶⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3412>

²⁶⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3410>

²⁶⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3409>

²⁶⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3407>

²⁶⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3406>

²⁶⁶² <https://github.com/STELLAR-GROUP/hpx/pull/3405>

²⁶⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/3404>

²⁶⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3402>

²⁶⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3401>

²⁶⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3400>

²⁶⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3398>

²⁶⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3397>

²⁶⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3396>

²⁶⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3394>

²⁶⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3393>

²⁶⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3392>

²⁶⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3391>

²⁶⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3390>

- PR #3389²⁶⁷⁵ - Fix deadlock during shutdown
- PR #3388²⁶⁷⁶ - Add several functionalities allowing to optimize synchronous action invocation
- PR #3387²⁶⁷⁷ - Add cmake option to opt out of fail-compile tests
- PR #3386²⁶⁷⁸ - Adding support for boost::container::small_vector to dataflow
- PR #3385²⁶⁷⁹ - Adds Debug option for hpx initializing from main
- PR #3384²⁶⁸⁰ - This hopefully fixes two tests that occasionally fail
- PR #3383²⁶⁸¹ - Making sure thread local storage is enable for hpxMP
- PR #3382²⁶⁸² - Fix usage of HPX_CAPTURE together with default value capture [=]
- PR #3381²⁶⁸³ - Replace undefined instantiations of uniform_int_distribution
- PR #3380²⁶⁸⁴ - Add missing semicolons to uses of HPX_COMPILER_FENCE
- PR #3379²⁶⁸⁵ - Fixing #3378
- PR #3377²⁶⁸⁶ - Adding build system support to integrate hpxmp into hpx at the user's machine
- PR #3375²⁶⁸⁷ - Replacing wrapper for __libc_start_main with main
- PR #3374²⁶⁸⁸ - Adds hpx_wrap to HPX_LINK_LIBRARIES which links only when specified.
- PR #3373²⁶⁸⁹ - Forcing cache settings in HPXConfig.cmake to guarantee updated values
- PR #3372²⁶⁹⁰ - Fix some more c++11 build problems
- PR #3371²⁶⁹¹ - Adds HPX_LINKER_FLAGS to HPX applications without editing their source codes
- PR #3370²⁶⁹² - util::format: add typeSpecifier<> specializations for %!s(MISSING) and %!!(MISSING)s
- PR #3369²⁶⁹³ - Adding configuration option to allow explicit disable of the new hpx_main feature on Linux
- PR #3368²⁶⁹⁴ - Updates doc with recent hpx_wrap implementation
- PR #3367²⁶⁹⁵ - Adds Mac OS implementation to hpx_main.hpp
- PR #3365²⁶⁹⁶ - Fix order of hpx libs in HPX_CONF_LIBRARIES.
- PR #3363²⁶⁹⁷ - Apex fixing null wrapper

²⁶⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3389>

²⁶⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3388>

²⁶⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3387>

²⁶⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3386>

²⁶⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3385>

²⁶⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3384>

²⁶⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/3383>

²⁶⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3382>

²⁶⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/3381>

²⁶⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3380>

²⁶⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3379>

²⁶⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3377>

²⁶⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3375>

²⁶⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3374>

²⁶⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3373>

²⁶⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3372>

²⁶⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3371>

²⁶⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3370>

²⁶⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3369>

²⁶⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3368>

²⁶⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3367>

²⁶⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3365>

²⁶⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3363>

- PR #3361²⁶⁹⁸ - Making sure all parcels get destroyed on an HPX thread (TCP pp)
- PR #3359²⁶⁹⁹ - Feature/improveerrorforcompiler
- PR #3357²⁷⁰⁰ - Static/dynamic executable implementation
- PR #3355²⁷⁰¹ - Reverting changes introduced by #3283 as those make applications hang
- PR #3354²⁷⁰² - Add external dependencies to HPX_LIBRARY_DIR
- PR #3353²⁷⁰³ - Fix libfabric tcp
- PR #3351²⁷⁰⁴ - Move obsolete header to tests directory.
- PR #3350²⁷⁰⁵ - Renaming two functions to avoid problem described in #3285
- PR #3349²⁷⁰⁶ - Make idle backoff exponential with maximum sleep time
- PR #3347²⁷⁰⁷ - Replace *simple_component** with *component** in the Documentation
- PR #3346²⁷⁰⁸ - Fix CMakeLists.txt example in quick start
- PR #3345²⁷⁰⁹ - Fix automatic setting of HPX_MORE_THAN_64_THREADS
- PR #3344²⁷¹⁰ - Reduce amount of information printed for unknown command line options
- PR #3343²⁷¹¹ - Safeguard HPX against destruction in global contexts
- PR #3341²⁷¹² - Allowing for all command line options to be used as configuration settings
- PR #3340²⁷¹³ - Always convert inspect results to JUnit XML
- PR #3336²⁷¹⁴ - Only run docker push on master on CircleCI
- PR #3335²⁷¹⁵ - Update description of hpx.os_threads config parameter.
- PR #3334²⁷¹⁶ - Making sure early logging settings don't get mixed with others
- PR #3333²⁷¹⁷ - Update CMake links and versions in documentation
- PR #3332²⁷¹⁸ - Add notes on target suffixes to CMake documentation
- PR #3331²⁷¹⁹ - Add quickstart section to documentation
- PR #3330²⁷²⁰ - Rename resource_partitioner test to avoid conflicts with pseudodependencies

²⁶⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3361>

²⁶⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3359>

²⁷⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3357>

²⁷⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3355>

²⁷⁰² <https://github.com/STELLAR-GROUP/hpx/pull/3354>

²⁷⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/3353>

²⁷⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3351>

²⁷⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3350>

²⁷⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3349>

²⁷⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3347>

²⁷⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3346>

²⁷⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3345>

²⁷¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3344>

²⁷¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3343>

²⁷¹² <https://github.com/STELLAR-GROUP/hpx/pull/3341>

²⁷¹³ <https://github.com/STELLAR-GROUP/hpx/pull/3340>

²⁷¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3336>

²⁷¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3335>

²⁷¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3334>

²⁷¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3333>

²⁷¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3332>

²⁷¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3331>

²⁷²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3330>

- PR #3328²⁷²¹ - Making sure object is pinned while executing actions, even if action returns a future
- PR #3327²⁷²² - Add missing std::forward to tuple.hpp
- PR #3326²⁷²³ - Make sure logging is up and running while modules are being discovered.
- PR #3324²⁷²⁴ - Replace C++14 overload of std::equal with C++11 code.
- PR #3323²⁷²⁵ - Fix a missing apex thread data (wrapper) initialization
- PR #3320²⁷²⁶ - Adding support for -std=c++2a (define *HPX_WITH_CXX2A=On*)
- PR #3319²⁷²⁷ - Replacing C++14 feature with equivalent C++11 code
- PR #3317²⁷²⁸ - Fix compilation with VS 15.7.1 and /std:c++latest
- PR #3316²⁷²⁹ - Fix includes for 1d_stencil_*_omp examples
- PR #3314²⁷³⁰ - Remove some unused parameter warnings
- PR #3313²⁷³¹ - Fix pu-step and pu-offset command line options
- PR #3312²⁷³² - Add conversion of inspect reports to JUnit XML
- PR #3311²⁷³³ - Fix escaping of closing braces in format specification syntax
- PR #3310²⁷³⁴ - Don't overwrite user settings with defaults in registration database
- PR #3309²⁷³⁵ - Fixing potential stack overflow for dataflow
- PR #3308²⁷³⁶ - This updates the .clang-format configuration file to utilize newer features
- PR #3306²⁷³⁷ - Marking migratable objects in their gid to allow not handling migration in AGAS
- PR #3305²⁷³⁸ - Add proper exception handling to run_as_hpx_thread
- PR #3303²⁷³⁹ - Changed std::rand to a better inbuilt PRNG Generator
- PR #3302²⁷⁴⁰ - All non-migratable (simple) components now encode their lva and component type in their gid
- PR #3301²⁷⁴¹ - Add nullptr_t overloads to resource partitioner
- PR #3298²⁷⁴² - Apex task wrapper memory bug
- PR #3295²⁷⁴³ - Fix mistakes after merge of CircleCI config

²⁷²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3328>

²⁷²² <https://github.com/STELLAR-GROUP/hpx/pull/3327>

²⁷²³ <https://github.com/STELLAR-GROUP/hpx/pull/3326>

²⁷²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3324>

²⁷²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3323>

²⁷²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3320>

²⁷²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3319>

²⁷²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3317>

²⁷²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3316>

²⁷³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3314>

2731 <https://github.com/STELLAR-GROUP/hpx/pull/3313>2732 <https://github.com/STELLAR-GROUP/hpx/pull/3312>2733 <https://github.com/STELLAR-GROUP/hpx/pull/3311>2734 <https://github.com/STELLAR-GROUP/hpx/pull/3310>2735 <https://github.com/STELLAR-GROUP/hpx/pull/3309>2736 <https://github.com/STELLAR-GROUP/hpx/pull/3308>2737 <https://github.com/STELLAR-GROUP/hpx/pull/3306>2738 <https://github.com/STELLAR-GROUP/hpx/pull/3305>2739 <https://github.com/STELLAR-GROUP/hpx/pull/3303>2740 <https://github.com/STELLAR-GROUP/hpx/pull/3302>2741 <https://github.com/STELLAR-GROUP/hpx/pull/3301>2742 <https://github.com/STELLAR-GROUP/hpx/pull/3298>2743 <https://github.com/STELLAR-GROUP/hpx/pull/3295>

- PR #3294²⁷⁴⁴ - Fix partitioned vector include in partitioned_vector_find tests
- PR #3293²⁷⁴⁵ - Adding emplace support to promise and make_ready_future
- PR #3292²⁷⁴⁶ - Add new cuda kernel synchronization with hpx::future demo
- PR #3291²⁷⁴⁷ - Fixes #3290
- PR #3289²⁷⁴⁸ - Fixing Docker image creation
- PR #3288²⁷⁴⁹ - Avoid allocating shared state for wait_all
- PR #3287²⁷⁵⁰ - Fixing /scheduler/utilization/instantaneous performance counter
- PR #3286²⁷⁵¹ - dataflow() and future::then() use sync policy where possible
- PR #3284²⁷⁵² - Background thread can use relaxed atomics to manipulate thread state
- PR #3283²⁷⁵³ - Do not unwrap ready future
- PR #3282²⁷⁵⁴ - Fix virtual method override warnings in static schedulers
- PR #3281²⁷⁵⁵ - Disable set_area_membind_nodeset for OSX
- PR #3279²⁷⁵⁶ - Add two variations to the future_overhead benchmark
- PR #3278²⁷⁵⁷ - Fix circleci workspace
- PR #3277²⁷⁵⁸ - Support external plugins
- PR #3276²⁷⁵⁹ - Fix missing parenthesis in hello_compute.cu.
- PR #3274²⁷⁶⁰ - Reinit counters synchronously in reinit_counters test
- PR #3273²⁷⁶¹ - Splitting tests to avoid compiler OOM
- PR #3271²⁷⁶² - Remove leftover code from context_generic_context.hpp
- PR #3269²⁷⁶³ - Fix bulk_construct with count = 0
- PR #3268²⁷⁶⁴ - Replace constexpr with HPX_CXX14_CONSTEXPR and HPX_CONSTEXPR
- PR #3266²⁷⁶⁵ - Replace boost::format with custom sprintf-based implementation
- PR #3265²⁷⁶⁶ - Split parallel tests on CircleCI

²⁷⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3294>

²⁷⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3293>

²⁷⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3292>

²⁷⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3291>

²⁷⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3289>

²⁷⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3288>

²⁷⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3287>

²⁷⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3286>

²⁷⁵² <https://github.com/STELLAR-GROUP/hpx/pull/3284>

²⁷⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3283>

²⁷⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3282>

²⁷⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3281>

²⁷⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3279>

²⁷⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3278>

²⁷⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3277>

²⁷⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3276>

²⁷⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3274>

²⁷⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3273>

²⁷⁶² <https://github.com/STELLAR-GROUP/hpx/pull/3271>

²⁷⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/3269>

²⁷⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3268>

²⁷⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3266>

²⁷⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3265>

- PR #3262²⁷⁶⁷ - Making sure documentation correctly links to source files
- PR #3261²⁷⁶⁸ - Apex refactoring fix rebind
- PR #3260²⁷⁶⁹ - Isolate performance counter parser into a separate TU
- PR #3256²⁷⁷⁰ - Post 1.1.0 version bumps
- PR #3254²⁷⁷¹ - Adding trait for actions allowing to make runtime decision on whether to execute it directly
- PR #3253²⁷⁷² - Bump minimal supported Boost to 1.58.0
- PR #3251²⁷⁷³ - Adds new feature: changing interval used in interval_timer (issue 3244)
- PR #3239²⁷⁷⁴ - Changing std::rand() to a better inbuilt PRNG generator.
- PR #3234²⁷⁷⁵ - Disable background thread when networking is off
- PR #3232²⁷⁷⁶ - Clean up suspension tests
- PR #3230²⁷⁷⁷ - Add optional scheduler mode parameter to create_thread_pool function
- PR #3228²⁷⁷⁸ - Allow suspension also on static schedulers
- PR #3163²⁷⁷⁹ - libfabric parcelport w/o HPX_PARCELPORT_LIBFABRIC_ENDPOINT_RDM
- PR #3036²⁷⁸⁰ - Switching to CircleCI 2.0

2.10.12 HPX V1.1.0 (Mar 24, 2018)

General changes

Here are some of the main highlights and changes for this release (in no particular order):

- We have changed the way *HPX* manages the processing units on a node. We do not longer implicitly bind all available cores to a single thread pool. The user has now full control over what processing units are bound to what thread pool, each with a separate scheduler. It is now also possible to create your own scheduler implementation and control what processing units this scheduler should use. We added the `hpx::resource::partitioner` that manages all available processing units and assigns resources to the used thread pools. Thread pools can be now be suspended/resumed independently. This functionality helps in running *HPX* concurrently to code that is directly relying on OpenMP²⁷⁸¹ and/or MPI²⁷⁸².
- We have continued to implement various parallel algorithms. *HPX* now almost completely implements all of the parallel algorithms as specified by the C++17 Standard²⁷⁸³. We have also continued to implement these algorithms for the distributed use case (for segmented data structures, such as `hpx::partitioned_vector`).

²⁷⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3262>

²⁷⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3261>

²⁷⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3260>

²⁷⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3256>

²⁷⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3254>

²⁷⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3253>

²⁷⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3251>

²⁷⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3239>

²⁷⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3234>

²⁷⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3232>

²⁷⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3230>

²⁷⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3228>

²⁷⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3163>

²⁷⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3036>

²⁷⁸¹ <https://openmp.org/wp/>

²⁷⁸² https://en.wikipedia.org/wiki/Message_Passing_Interface

²⁷⁸³ <http://www.open-std.org/jtc1/sc22/wg21>

- Added a compatibility layer for `std::thread`, `std::mutex`, and `std::condition_variable` allowing for the code to use those facilities where available and to fall back to the corresponding Boost facilities otherwise. The CMake²⁷⁸⁴ configuration option `-DHPX_WITH_THREAD_COMPATIBILITY=On` can be used to force using the Boost equivalents.
- The parameter sequence for the `hpx::parallel::transform_inclusive_scan` overload taking one iterator range has changed (again) to match the changes this algorithm has undergone while being moved to C++17. The old overloads can be still enabled at configure time by passing `-DHPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY=On` to CMake²⁷⁸⁵.
- The parameter sequence for the `hpx::parallel::inclusive_scan` overload taking one iterator range has changed to match the changes this algorithm has undergone while being moved to C++17. The old overloads can be still enabled at configure time by passing `-DHPX_WITH_INCLUSIVE_SCAN_COMPATIBILITY=On` to CMake.
- Added a helper facility `hpx::local_new` which is equivalent to `hpx::new_` except that it creates components locally only. As a consequence, the used component constructor may accept non-serializable argument types and/or non-const references or pointers.
- Removed the (broken) component type `hpx::lcos::queue<T>`. The old type is still available at configure time by passing `-DHPX_WITH_QUEUE_COMPATIBILITY=On` to CMake.
- The parallel algorithms adopted for C++17 restrict the iterator categories usable with those to at least forward iterators. Our implementation of the parallel algorithms was supporting input iterators (and output iterators) as well by simply falling back to sequential execution. We have now made our implementations conforming by requiring at least forward iterators. In order to enable the old behavior use the compatibility option `-DHPX_WITH_ALGORITHM_INPUT_ITERATOR_SUPPORT=On` on the CMake²⁷⁸⁶ command line.
- We have added the functionalities allowing for LCOs being implemented using (simple) components. Before LCOs had to always be implemented using managed components.
- User defined components don't have to be default-constructible anymore. Return types from actions don't have to be default-constructible anymore either. Our serialization layer now in general supports non-default-constructible types.
- We have added a new launch policy `hpx::launch::lazy` that allows one to defer the decision on what launch policy to use to the point of execution. This policy is initialized with a function (object) that – when invoked – is expected to produce the desired launch policy.

Breaking changes

- We have dropped support for the gcc compiler version V4.8. The minimal gcc version we now test on is gcc V4.9. The minimally required version of CMake²⁷⁸⁷ is now V3.3.2.
- We have dropped support for the Visual Studio 2013 compiler version. The minimal Visual Studio version we now test on is Visual Studio 2015.5.
- We have dropped support for the Boost V1.51-V1.54. The minimal version of Boost we now test is Boost V1.55.
- We have dropped support for the `hpx::util::unwrapped` API. `hpx::util::unwrapped` will stay functional to some degree, until it finally gets removed in a later version of HPX. The functional usage of `hpx::util::unwrapped` should be changed to the new `hpx::util::unwrapping` function whereas the immediate usage should be replaced to `hpx::util::unwrap`.

²⁷⁸⁴ <https://www.cmake.org>

²⁷⁸⁵ <https://www.cmake.org>

²⁷⁸⁶ <https://www.cmake.org>

²⁷⁸⁷ <https://www.cmake.org>

- The performance counter names referring to properties as exposed by the threading subsystem have changes as those now additionally have to specify the thread-pool. See the corresponding documentation for more details.
- The overloads of `hpx::async` that invoke an action do not perform implicit unwrapping of the returned future anymore in case the invoked function does return a future in the first place. In this case `hpx::async` now returns a `hpx::future<future<T>>` making its behavior conforming to its local counterpart.
- We have replaced the use of `boost::exception_ptr` in our APIs with the equivalent `std::exception_ptr`. Please change your codes accordingly. No compatibility settings are provided.
- We have removed the compatibility settings for `HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY` and `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY` as their life-cycle has reached its end.
- We have removed the experimental thread schedulers `hierarchy_scheduler`, `periodic_priority_scheduler` and `throttling_scheduler` in an effort to clean up and consolidate our thread schedulers.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- PR #3250²⁷⁸⁸ - Apex refactoring with guids
- PR #3249²⁷⁸⁹ - Updating People.qbk
- PR #3246²⁷⁹⁰ - Assorted fixes for CUDA
- PR #3245²⁷⁹¹ - Apex refactoring with guids
- PR #3242²⁷⁹² - Modify task counting in `thread_queue.hpp`
- PR #3240²⁷⁹³ - Fixed typos
- PR #3238²⁷⁹⁴ - Readding accidentally removed `std::abort`
- PR #3237²⁷⁹⁵ - Adding Pipeline example
- PR #3236²⁷⁹⁶ - Fixing `memory_block`
- PR #3233²⁷⁹⁷ - Make `schedule_thread` take suspended threads into account
- Issue #3226²⁷⁹⁸ - `memory_block` is breaking, signaling SIGSEGV on a thread on creation and freeing
- PR #3225²⁷⁹⁹ - Applying quick fix for hwloc-2.0
- Issue #3224²⁸⁰⁰ - HPX counters crashing the application
- PR #3223²⁸⁰¹ - Fix returns when setting config entries
- Issue #3222²⁸⁰² - Errors linking `libhpx.so`

²⁷⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3250>

²⁷⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3249>

²⁷⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3246>

²⁷⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3245>

²⁷⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3242>

²⁷⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3240>

²⁷⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3238>

²⁷⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3237>

²⁷⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3236>

²⁷⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3233>

²⁷⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3226>

²⁷⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3225>

²⁸⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3224>

²⁸⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3223>

²⁸⁰² <https://github.com/STELLAR-GROUP/hpx/issues/3222>

- Issue #3221²⁸⁰³ - HPX on Mac OS X with HWLoc 2.0.0 fails to run
- PR #3216²⁸⁰⁴ - Reorder a variadic array to satisfy VS 2017 15.6
- PR #3214²⁸⁰⁵ - Changed prerequisites.qbk to avoid confusion while building boost
- PR #3213²⁸⁰⁶ - Relax locks for thread suspension to avoid holding locks when yielding
- PR #3212²⁸⁰⁷ - Fix check in sequenced_executor test
- PR #3211²⁸⁰⁸ - Use preinit_array to set argc/argv in init_globally example
- PR #3210²⁸⁰⁹ - Adapted parallel::{search | search_n} for Ranges TS (see #1668)
- PR #3209²⁸¹⁰ - Fix locking problems during shutdown
- Issue #3208²⁸¹¹ - init_globally throwing a run-time error
- PR #3206²⁸¹² - Addition of new arithmetic performance counter “Count”
- PR #3205²⁸¹³ - Fixing return type calculation for bulk_then_execute
- PR #3204²⁸¹⁴ - Changing std::rand() to a better inbuilt PRNG generator
- PR #3203²⁸¹⁵ - Resolving problems during shutdown for VS2015
- PR #3202²⁸¹⁶ - Making sure resource partitioner is not accessed if its not valid
- PR #3201²⁸¹⁷ - Fixing optional::swap
- Issue #3200²⁸¹⁸ - hpx::util::optional fails
- PR #3199²⁸¹⁹ - Fix sliding_semaphore test
- PR #3198²⁸²⁰ - Set pre_main status before launching run_helper
- PR #3197²⁸²¹ - Update README.rst
- PR #3194²⁸²² - parallel::{fill|fill_n} updated for Ranges TS
- PR #3193²⁸²³ - Updating Runtime.cpp by adding correct description of Performance counters during register
- PR #3191²⁸²⁴ - Fix sliding_semaphore_2338 test
- PR #3190²⁸²⁵ - Topology improvements

²⁸⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/3221>

²⁸⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3216>

²⁸⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3214>

²⁸⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3213>

²⁸⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3212>

²⁸⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3211>

²⁸⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3210>

²⁸¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3209>

²⁸¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/3208>

²⁸¹² <https://github.com/STELLAR-GROUP/hpx/pull/3206>

²⁸¹³ <https://github.com/STELLAR-GROUP/hpx/pull/3205>

²⁸¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3204>

²⁸¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3203>

²⁸¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3202>

²⁸¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3201>

²⁸¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3200>

²⁸¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3199>

²⁸²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3198>

²⁸²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3197>

²⁸²² <https://github.com/STELLAR-GROUP/hpx/pull/3194>

²⁸²³ <https://github.com/STELLAR-GROUP/hpx/pull/3193>

²⁸²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3191>

²⁸²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3190>

- PR #3189²⁸²⁶ - Deleting one include of median from BOOST library to arithmetics_counter file
- PR #3188²⁸²⁷ - Optionally disable printing of diagnostics during terminate
- PR #3187²⁸²⁸ - Suppressing cmake warning issued by cmake > V3.11
- PR #3185²⁸²⁹ - Remove unused scoped_unlock, unlock_guard_try
- PR #3184²⁸³⁰ - Fix nqueen example
- PR #3183²⁸³¹ - Add runtime start/stop, resume/suspend and OpenMP benchmarks
- Issue #3182²⁸³² - bulk_then_execute has unexpected return type/does not compile
- Issue #3181²⁸³³ - hwloc 2.0 breaks topo class and cannot be used
- Issue #3180²⁸³⁴ - Schedulers that don't support suspend/resume are unusable
- PR #3179²⁸³⁵ - Various minor changes to support FLeCSI
- PR #3178²⁸³⁶ - Fix #3124
- PR #3177²⁸³⁷ - Removed allgather
- PR #3176²⁸³⁸ - Fixed Documentation for "using_hpx_pkgconfig"
- PR #3174²⁸³⁹ - Add hpx::iostreams::ostream overload to format_to
- PR #3172²⁸⁴⁰ - Fix lifo queue backend
- PR #3171²⁸⁴¹ - adding the missing unset() function to cpu_mask() for case of more than 64 threads
- PR #3170²⁸⁴² - Add cmake flag -DHPX_WITHFAULT_TOLERANCE=ON (OFF by default)
- PR #3169²⁸⁴³ - Adapted parallel::{countlcount_if} for Ranges TS (see #1668)
- PR #3168²⁸⁴⁴ - Changing used namespace for seq execution policy
- Issue #3167²⁸⁴⁵ - Update GSoC projects
- Issue #3166²⁸⁴⁶ - Application (Octotiger) gets stuck on hpx::finalize when only using one thread
- Issue #3165²⁸⁴⁷ - Compilation of parallel algorithms with HPX_WITH_DATAPAR is broken
- PR #3164²⁸⁴⁸ - Fixing component migration

²⁸²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3189>

²⁸²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3188>

²⁸²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3187>

²⁸²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3185>

²⁸³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3184>

²⁸³¹ <https://github.com/STELLAR-GROUP/hpx/pull/3183>

²⁸³² <https://github.com/STELLAR-GROUP/hpx/issues/3182>

²⁸³³ <https://github.com/STELLAR-GROUP/hpx/issues/3181>

²⁸³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/3180>

²⁸³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3179>

²⁸³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3178>

²⁸³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3177>

²⁸³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3176>

²⁸³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3174>

²⁸⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3172>

²⁸⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/3171>

²⁸⁴² <https://github.com/STELLAR-GROUP/hpx/pull/3170>

²⁸⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/3169>

²⁸⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3168>

²⁸⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3167>

²⁸⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3166>

²⁸⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3165>

²⁸⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3164>

- PR #3162²⁸⁴⁹ - regex_from_pattern: escape regex special characters to avoid misinterpretation
- Issue #3161²⁸⁵⁰ - Building HPX with hwloc 2.0.0 fails
- PR #3160²⁸⁵¹ - Fixing the handling of quoted command line arguments.
- PR #3158²⁸⁵² - Fixing a race with timed suspension (second attempt)
- PR #3157²⁸⁵³ - Revert “Fixing a race with timed suspension”
- PR #3156²⁸⁵⁴ - Fixing serialization of classes with incompatible serialize signature
- PR #3154²⁸⁵⁵ - More refactorings based on clang-tidy reports
- PR #3153²⁸⁵⁶ - Fixing a race with timed suspension
- PR #3152²⁸⁵⁷ - Documentation for runtime suspension
- PR #3151²⁸⁵⁸ - Use small_vector only from boost version 1.59 onwards
- PR #3150²⁸⁵⁹ - Avoiding more stack overflows
- PR #3148²⁸⁶⁰ - Refactoring component_base and base_action/transfer_base_action
- PR #3147²⁸⁶¹ - Move yield_while out of detail namespace and into own file
- PR #3145²⁸⁶² - Remove a leftover of the cxx11 std array cleanup
- PR #3144²⁸⁶³ - Minor changes to how actions are executed
- PR #3143²⁸⁶⁴ - Fix stack overhead
- PR #3142²⁸⁶⁵ - Fix typo in config.hpp
- PR #3141²⁸⁶⁶ - Fixing small_vector compatibility with older boost version
- PR #3140²⁸⁶⁷ - is_heap_text fix
- Issue #3139²⁸⁶⁸ - Error in is_heap_tests.hpp
- PR #3138²⁸⁶⁹ - Partially reverting #3126
- PR #3137²⁸⁷⁰ - Suspend speedup
- PR #3136²⁸⁷¹ - Revert “Fixing #2325”

²⁸⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3162>

²⁸⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3161>

²⁸⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3160>

²⁸⁵² <https://github.com/STELLAR-GROUP/hpx/pull/3158>

²⁸⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3157>

²⁸⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3156>

²⁸⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3154>

²⁸⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3153>

²⁸⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3152>

²⁸⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3151>

²⁸⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3150>

²⁸⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3148>

²⁸⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3147>

²⁸⁶² <https://github.com/STELLAR-GROUP/hpx/pull/3145>

²⁸⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/3144>

²⁸⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3143>

²⁸⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3142>

²⁸⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3141>

²⁸⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3140>

²⁸⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3139>

²⁸⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3138>

²⁸⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3137>

²⁸⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3136>

- PR #3135²⁸⁷² - Improving destruction of threads
- Issue #3134²⁸⁷³ - HPX_SERIALIZATION_SPLIT_FREE does not stop compiler from looking for serialize() method
- PR #3133²⁸⁷⁴ - Make hwloc compulsory
- PR #3132²⁸⁷⁵ - Update CXX14 constexpr feature test
- PR #3131²⁸⁷⁶ - Fixing #2325
- PR #3130²⁸⁷⁷ - Avoid completion handler allocation
- PR #3129²⁸⁷⁸ - Suspend runtime
- PR #3128²⁸⁷⁹ - Make docbook dtd and xsl path names consistent
- PR #3127²⁸⁸⁰ - Add hpx::start nullptr overloads
- PR #3126²⁸⁸¹ - Cleaning up coroutine implementation
- PR #3125²⁸⁸² - Replacing nullptr with hpx::threads::invalid_thread_id
- Issue #3124²⁸⁸³ - Add hello_world_component to CI builds
- PR #3123²⁸⁸⁴ - Add new constructor.
- PR #3122²⁸⁸⁵ - Fixing #3121
- Issue #3121²⁸⁸⁶ - HPX_SMT_PAUSE is broken on non-x86 platforms when __GNUC__ is defined
- PR #3120²⁸⁸⁷ - Don't use boost::intrusive_ptr for thread_id_type
- PR #3119²⁸⁸⁸ - Disable default executor compatibility with V1 executors
- PR #3118²⁸⁸⁹ - Adding performance_counter::reinit to allow for dynamically changing counter sets
- PR #3117²⁸⁹⁰ - Replace uses of boost/experimental::optional with util::optional
- PR #3116²⁸⁹¹ - Moving background thread APEX timer #2980
- PR #3115²⁸⁹² - Fixing race condition in channel test
- PR #3114²⁸⁹³ - Avoid using util::function for thread function wrappers
- PR #3113²⁸⁹⁴ - cmake V3.10.2 has changed the variable names used for MPI

²⁸⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3135>

²⁸⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/3134>

²⁸⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3133>

²⁸⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3132>

²⁸⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3131>

²⁸⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3130>

²⁸⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3129>

²⁸⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3128>

²⁸⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3127>

²⁸⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/3126>

²⁸⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3125>

²⁸⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/3124>

²⁸⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3123>

²⁸⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3122>

²⁸⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3121>

²⁸⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3120>

²⁸⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3119>

²⁸⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3118>

²⁸⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3117>

²⁸⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3116>

²⁸⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3115>

²⁸⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/3114>

²⁸⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3113>

- PR #3112²⁸⁹⁵ - Minor fixes to exclusive_scan algorithm
- PR #3111²⁸⁹⁶ - Revert “fix detection of cxx11_std_atomic”
- PR #3110²⁸⁹⁷ - Suspend thread pool
- PR #3109²⁸⁹⁸ - Fixing thread scheduling when yielding a thread id
- PR #3108²⁸⁹⁹ - Revert “Suspend thread pool”
- PR #3107²⁹⁰⁰ - Remove UB from thread::id relational operators
- PR #3106²⁹⁰¹ - Add cmake test for std::decay_t to fix cuda build
- PR #3105²⁹⁰² - Fixing refcount for async traversal frame
- PR #3104²⁹⁰³ - Local execution of direct actions is now actually performed directly
- PR #3103²⁹⁰⁴ - Adding support for generic counter_raw_values performance counter type
- Issue #3102²⁹⁰⁵ - Introduce generic performance counter type returning an array of values
- PR #3101²⁹⁰⁶ - Revert “Adapting stack overhead limit for gcc 4.9”
- PR #3100²⁹⁰⁷ - Fix #3068 (condition_variable deadlock)
- PR #3099²⁹⁰⁸ - Fixing lock held during suspension in papi counter component
- PR #3098²⁹⁰⁹ - Unbreak broadcast_wait_for_2822 test
- PR #3097²⁹¹⁰ - Adapting stack overhead limit for gcc 4.9
- PR #3096²⁹¹¹ - fix detection of cxx11_std_atomic
- PR #3095²⁹¹² - Add ciso646 header to get _LIBCPP_VERSION for testing inplace merge
- PR #3094²⁹¹³ - Relax atomic operations on performance counter values
- PR #3093²⁹¹⁴ - Short-circuit all_of/any_of/none_of instantiations
- PR #3092²⁹¹⁵ - Take advantage of C++14 lambda capture initialization syntax, where possible
- PR #3091²⁹¹⁶ - Remove more references to Boost from logging code
- PR #3090²⁹¹⁷ - Unify use of yield/yield_k

²⁸⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3112>

²⁸⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3111>

²⁸⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3110>

²⁸⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3109>

²⁸⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3108>

²⁹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3107>

²⁹⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/3106>

²⁹⁰² <https://github.com/STELLAR-GROUP/hpx/pull/3105>

²⁹⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/3104>

²⁹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3103>

²⁹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3102>

²⁹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3101>

²⁹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3100>

²⁹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3099>

2909 <https://github.com/STELLAR-GROUP/hpx/pull/3098>2910 <https://github.com/STELLAR-GROUP/hpx/pull/3097>2911 <https://github.com/STELLAR-GROUP/hpx/pull/3096>2912 <https://github.com/STELLAR-GROUP/hpx/pull/3095>2913 <https://github.com/STELLAR-GROUP/hpx/pull/3094>2914 <https://github.com/STELLAR-GROUP/hpx/pull/3093>2915 <https://github.com/STELLAR-GROUP/hpx/pull/3092>2916 <https://github.com/STELLAR-GROUP/hpx/pull/3091>2917 <https://github.com/STELLAR-GROUP/hpx/pull/3090>

- PR #3089²⁹¹⁸ - Fix a strange thing in parallel::detail::handle_exception. (Fix #2834.)
- Issue #3088²⁹¹⁹ - A strange thing in parallel::sort.
- PR #3087²⁹²⁰ - Fixing assertion in default_distribution_policy
- PR #3086²⁹²¹ - Implement parallel::remove and parallel::remove_if
- PR #3085²⁹²² - Addressing breaking changes in Boost V1.66
- PR #3084²⁹²³ - Ignore build warnings round 2
- PR #3083²⁹²⁴ - Fix typo HPX_WITH_MM_PREFECHT
- PR #3081²⁹²⁵ - Pre-decay template arguments early
- PR #3080²⁹²⁶ - Suspend thread pool
- PR #3079²⁹²⁷ - Ignore build warnings
- PR #3078²⁹²⁸ - Don't test inplace_merge with libc++
- PR #3076²⁹²⁹ - Fixing 3075: Part 1
- PR #3074²⁹³⁰ - Fix more build warnings
- PR #3073²⁹³¹ - Suspend thread cleanup
- PR #3072²⁹³² - Change existing symbol_namespace::iterate to return all data instead of invoking a callback
- PR #3071²⁹³³ - Fixing pack_traversal_async test
- PR #3070²⁹³⁴ - Fix dynamic_counters_loaded_1508 test by adding dependency to memory_component
- PR #3069²⁹³⁵ - Fix scheduling loop exit
- Issue #3068²⁹³⁶ - hpx::lcos::condition_variable could be suspect to deadlocks
- PR #3067²⁹³⁷ - #ifdef out random_shuffle deprecated in later c++
- PR #3066²⁹³⁸ - Make coalescing test depend on coalescing library to ensure it gets built
- PR #3065²⁹³⁹ - Workaround for minimal_timed_async_executor_test compilation failures, attempts to copy a deferred call (in unevaluated context)
- PR #3064²⁹⁴⁰ - Fixing wrong condition in wrapper_heap

²⁹¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3089>

²⁹¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3088>

²⁹²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3087>

²⁹²¹ <https://github.com/STELLAR-GROUP/hpx/pull/3086>

²⁹²² <https://github.com/STELLAR-GROUP/hpx/pull/3085>

²⁹²³ <https://github.com/STELLAR-GROUP/hpx/pull/3084>

²⁹²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3083>

²⁹²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3081>

²⁹²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3080>

²⁹²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3079>

²⁹²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3078>

²⁹²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3076>

²⁹³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3074>

²⁹³¹ <https://github.com/STELLAR-GROUP/hpx/pull/3073>

²⁹³² <https://github.com/STELLAR-GROUP/hpx/pull/3072>

²⁹³³ <https://github.com/STELLAR-GROUP/hpx/pull/3071>

²⁹³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3070>

²⁹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3069>

²⁹³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/3068>

²⁹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3067>

²⁹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3066>

²⁹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3065>

²⁹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3064>

- PR #3062²⁹⁴¹ - Fix exception handling for execution::seq
- PR #3061²⁹⁴² - Adapt MSVC C++ mode handling to VS15.5
- PR #3060²⁹⁴³ - Fix compiler problem in MSVC release mode
- PR #3059²⁹⁴⁴ - Fixing #2931
- Issue #3058²⁹⁴⁵ - minimal_timed_async_executor_test_exe fails to compile on master (d6f505c)
- PR #3057²⁹⁴⁶ - Fix stable_merge_2964 compilation problems
- PR #3056²⁹⁴⁷ - Fix some build warnings caused by unused variables/unnecessary tests
- PR #3055²⁹⁴⁸ - Update documentation for running tests
- Issue #3054²⁹⁴⁹ - Assertion failure when using bulk hpx::new_ in asynchronous mode
- PR #3052²⁹⁵⁰ - Do not bind test running to cmake test build rule
- PR #3051²⁹⁵¹ - Fix HPX-Qt interaction in Qt example.
- Issue #3048²⁹⁵² - nqueen example fails occasionally
- PR #3047²⁹⁵³ - Fixing #3044
- PR #3046²⁹⁵⁴ - Add OS thread suspension
- PR #3042²⁹⁵⁵ - PyCicle - first attempt at a build tool for checking PR's
- PR #3041²⁹⁵⁶ - Fix a problem about asynchronous execution of parallel::merge and parallel::partition.
- PR #3040²⁹⁵⁷ - Fix a mistake about exception handling in asynchronous execution of scan_partitioner.
- PR #3039²⁹⁵⁸ - Consistently use executors to schedule work
- PR #3038²⁹⁵⁹ - Fixing local direct function execution and lambda actions perfect forwarding
- PR #3035²⁹⁶⁰ - Make parallel unit test names match build target/folder names
- PR #3033²⁹⁶¹ - Fix setting of default build type
- Issue #3032²⁹⁶² - Fix partitioner arg copy found in #2982
- Issue #3031²⁹⁶³ - Errors linking libhpx.so due to missing references (master branch, commit 6679a8882)

²⁹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/3062>

²⁹⁴² <https://github.com/STELLAR-GROUP/hpx/pull/3061>

²⁹⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/3060>

²⁹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3059>

²⁹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/3058>

²⁹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3057>

²⁹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3056>

²⁹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3055>

²⁹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/3054>

²⁹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3052>

²⁹⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/3051>

²⁹⁵² <https://github.com/STELLAR-GROUP/hpx/issues/3048>

²⁹⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/3047>

²⁹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3046>

²⁹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3042>

²⁹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3041>

²⁹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3040>

²⁹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3039>

²⁹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3038>

²⁹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3035>

²⁹⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/3033>

²⁹⁶² <https://github.com/STELLAR-GROUP/hpx/issues/3032>

²⁹⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/3031>

- PR #3030²⁹⁶⁴ - Revert “implement executor then interface with && forwarding reference”
- PR #3029²⁹⁶⁵ - Run CI inspect checks before building
- PR #3028²⁹⁶⁶ - Added range version of parallel::move
- Issue #3027²⁹⁶⁷ - Implement all scheduling APIs in terms of executors
- PR #3026²⁹⁶⁸ - implement executor then interface with && forwarding reference
- PR #3025²⁹⁶⁹ - Fix typo uninitialized to uninitialized
- PR #3024²⁹⁷⁰ - Inspect fixes
- PR #3023²⁹⁷¹ - P0356 Simplified partial function application
- PR #3022²⁹⁷² - Master fixes
- PR #3021²⁹⁷³ - Segfault fix
- PR #3020²⁹⁷⁴ - Disable command-line aliasing for applications that use user_main
- PR #3019²⁹⁷⁵ - Adding enable_elasticity option to pool configuration
- PR #3018²⁹⁷⁶ - Fix stack overflow detection configuration in header files
- PR #3017²⁹⁷⁷ - Speed up local action execution
- PR #3016²⁹⁷⁸ - Unify stack-overflow detection options, remove reference to libsigsegv
- PR #3015²⁹⁷⁹ - Speeding up accessing the resource partitioner and the topology info
- Issue #3014²⁹⁸⁰ - HPX does not compile on POWER8 with gcc 5.4
- Issue #3013²⁹⁸¹ - hello_world occasionally prints multiple lines from a single OS-thread
- PR #3012²⁹⁸² - Silence warning about casting away qualifiers in itt_notify.hpp
- PR #3011²⁹⁸³ - Fix cpuset leak in hwloc_topology_info.cpp
- PR #3010²⁹⁸⁴ - Remove useless decay_copy
- PR #3009²⁹⁸⁵ - Fixing 2996
- PR #3008²⁹⁸⁶ - Remove unused internal function

²⁹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3030>

²⁹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3029>

²⁹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3028>

²⁹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/3027>

²⁹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3026>

²⁹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3025>

²⁹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/3024>

²⁹⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/3023>

²⁹⁷² <https://github.com/STELLAR-GROUP/hpx/pull/3022>

²⁹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/3021>

²⁹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3020>

²⁹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3019>

²⁹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3018>

²⁹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3017>

²⁹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/3016>

²⁹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3015>

²⁹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3014>

²⁹⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/3013>

²⁹⁸² <https://github.com/STELLAR-GROUP/hpx/pull/3012>

²⁹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/3011>

²⁹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/3010>

²⁹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/3009>

²⁹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/3008>

- PR #3007²⁹⁸⁷ - Fixing wrapper_heap alignment problems
- Issue #3006²⁹⁸⁸ - hwloc memory leak
- PR #3004²⁹⁸⁹ - Silence C4251 (needs to have dll-interface) for future_data_void
- Issue #3003²⁹⁹⁰ - Suspension of runtime
- PR #3001²⁹⁹¹ - Attempting to avoid data races in async_traversal while evaluating dataflow()
- PR #3000²⁹⁹² - Adding hpx::util::optional as a first step to replace experimental::optional
- PR #2998²⁹⁹³ - Cleanup up and Fixing component creation and deletion
- Issue #2996²⁹⁹⁴ - Build fails with HPX_WITH_HWLOC=OFF
- PR #2995²⁹⁹⁵ - Push more future_data functionality to source file
- PR #2994²⁹⁹⁶ - WIP: Fix throttle test
- PR #2993²⁹⁹⁷ - Making sure -hpx:help does not throw for required (but missing) arguments
- PR #2992²⁹⁹⁸ - Adding non-blocking (on destruction) service executors
- Issue #2991²⁹⁹⁹ - run_as_os_thread locks up
- Issue #2990³⁰⁰⁰ - --help will not work until all required options are provided
- PR #2989³⁰⁰¹ - Improve error messages caused by misuse of dataflow
- PR #2988³⁰⁰² - Improve error messages caused by misuse of .then
- Issue #2987³⁰⁰³ - stack overflow detection producing false positives
- PR #2986³⁰⁰⁴ - Deduplicate non-dependent thread_info logging types
- PR #2985³⁰⁰⁵ - Adapted parallel::{all_oflany_oflnone_of} for Ranges TS (see #1668)
- PR #2984³⁰⁰⁶ - Refactor one_size_heap code to simplify code
- PR #2983³⁰⁰⁷ - Fixing local_new_component
- PR #2982³⁰⁰⁸ - Clang tidy
- PR #2981³⁰⁰⁹ - Simplify allocator rebinding in pack traversal

²⁹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/3007>

²⁹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/3006>

²⁹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/3004>

²⁹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/3003>

²⁹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/3001>

²⁹⁹² <https://github.com/STELLAR-GROUP/hpx/pull/3000>

²⁹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2998>

²⁹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2996>

²⁹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2995>

²⁹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2994>

²⁹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2993>

²⁹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2992>

²⁹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2991>

³⁰⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2990>

³⁰⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2989>

³⁰⁰² <https://github.com/STELLAR-GROUP/hpx/pull/2988>

³⁰⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/2987>

³⁰⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2986>

³⁰⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2985>

³⁰⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2984>

³⁰⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2983>

³⁰⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2982>

³⁰⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2981>

- PR #2979³⁰¹⁰ - Fixing integer overflows
- PR #2978³⁰¹¹ - Implement parallel::inplace_merge
- Issue #2977³⁰¹² - Make hwloc compulsory instead of optional
- PR #2976³⁰¹³ - Making sure client_base instance that registered the component does not unregister it when being destructed
- PR #2975³⁰¹⁴ - Change version of pulled APEX to master
- PR #2974³⁰¹⁵ - Fix domain not being freed at the end of scheduling loop
- PR #2973³⁰¹⁶ - Fix small typos
- PR #2972³⁰¹⁷ - Adding uintstd.h header
- PR #2971³⁰¹⁸ - Fall back to creating local components using local_new
- PR #2970³⁰¹⁹ - Improve is_tuple_like trait
- PR #2969³⁰²⁰ - Fix HPX_WITH_MORE_THAN_64_THREADS default value
- PR #2968³⁰²¹ - Cleaning up dataflow overload set
- PR #2967³⁰²² - Make parallel::merge is stable. (Fix #2964.)
- PR #2966³⁰²³ - Fixing a couple of held locks during exception handling
- PR #2965³⁰²⁴ - Adding missing #include
- Issue #2964³⁰²⁵ - parallel merge is not stable
- PR #2963³⁰²⁶ - Making sure any function object passed to dataflow is released after being invoked
- PR #2962³⁰²⁷ - Partially reverting #2891
- PR #2961³⁰²⁸ - Attempt to fix the gcc 4.9 problem with the async pack traversal
- Issue #2959³⁰²⁹ - Program terminates during error handling
- Issue #2958³⁰³⁰ - HPX_PLAIN_ACTION breaks due to missing include
- PR #2957³⁰³¹ - Fixing errors generated by mixing different attribute syntaxes
- Issue #2956³⁰³² - Mixing attribute syntaxes leads to compiler errors

³⁰¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2979>

³⁰¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2978>

³⁰¹² <https://github.com/STELLAR-GROUP/hpx/issues/2977>

³⁰¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2976>

³⁰¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2975>

³⁰¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2974>

³⁰¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2973>

³⁰¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2972>

³⁰¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2971>

3019 <https://github.com/STELLAR-GROUP/hpx/pull/2970>3020 <https://github.com/STELLAR-GROUP/hpx/pull/2969>3021 <https://github.com/STELLAR-GROUP/hpx/pull/2968>3022 <https://github.com/STELLAR-GROUP/hpx/pull/2967>3023 <https://github.com/STELLAR-GROUP/hpx/pull/2966>3024 <https://github.com/STELLAR-GROUP/hpx/pull/2965>3025 <https://github.com/STELLAR-GROUP/hpx/issues/2964>3026 <https://github.com/STELLAR-GROUP/hpx/pull/2963>3027 <https://github.com/STELLAR-GROUP/hpx/pull/2962>3028 <https://github.com/STELLAR-GROUP/hpx/pull/2961>3029 <https://github.com/STELLAR-GROUP/hpx/issues/2959>3030 <https://github.com/STELLAR-GROUP/hpx/issues/2958>3031 <https://github.com/STELLAR-GROUP/hpx/pull/2957>3032 <https://github.com/STELLAR-GROUP/hpx/issues/2956>

- Issue #2955³⁰³³ - Fix OS-Thread throttling
- PR #2953³⁰³⁴ - Making sure any hpx.os_threads=N supplied through a -hpx::config file is taken into account
- PR #2952³⁰³⁵ - Removing wrong call to cleanup_terminated_locked
- PR #2951³⁰³⁶ - Revert “Make sure the function vtables are initialized before use”
- PR #2950³⁰³⁷ - Fix a namespace compilation error when some schedulers are disabled
- Issue #2949³⁰³⁸ - master branch giving lockups on shutdown
- Issue #2947³⁰³⁹ - hpx.ini is not used correctly at initialization
- PR #2946³⁰⁴⁰ - Adding explicit feature test for thread_local
- PR #2945³⁰⁴¹ - Make sure the function vtables are initialized before use
- PR #2944³⁰⁴² - Attempting to solve affinity problems on CircleCI
- PR #2943³⁰⁴³ - Changing channel actions to be direct
- PR #2942³⁰⁴⁴ - Adding split_future for std::vector
- PR #2941³⁰⁴⁵ - Add a feature test to test for CXX11 override
- Issue #2940³⁰⁴⁶ - Add split_future for future<vector<T>>
- PR #2939³⁰⁴⁷ - Making error reporting during problems with setting affinity masks more verbose
- PR #2938³⁰⁴⁸ - Fix this various executors
- PR #2937³⁰⁴⁹ - Fix some typos in documentation
- PR #2934³⁰⁵⁰ - Remove the need for “complete” SFINAE checks
- PR #2933³⁰⁵¹ - Making sure parallel::for_loop is executed in parallel if requested
- PR #2932³⁰⁵² - Classify chunk_size_iterator to input iterator tag. (Fix #2866)
- Issue #2931³⁰⁵³ - --hpx:help triggers unusual error with clang build
- PR #2930³⁰⁵⁴ - Add #include files needed to set _POSIX_VERSION for debug check
- PR #2929³⁰⁵⁵ - Fix a couple of deprecated c++ features

³⁰³³ <https://github.com/STELLAR-GROUP/hpx/issues/2955>

³⁰³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2953>

³⁰³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2952>

³⁰³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2951>

³⁰³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2950>

³⁰³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2949>

³⁰³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2947>

³⁰⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2946>

³⁰⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/2945>

³⁰⁴² <https://github.com/STELLAR-GROUP/hpx/pull/2944>

³⁰⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/2943>

³⁰⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2942>

³⁰⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2941>

³⁰⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2940>

³⁰⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2939>

³⁰⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2938>

³⁰⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2937>

³⁰⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2934>

³⁰⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/2933>

³⁰⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2932>

³⁰⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/2931>

³⁰⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2930>

³⁰⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2929>

- PR #2928³⁰⁵⁶ - Fixing execution parameters
- Issue #2927³⁰⁵⁷ - CMake warning: ... cycle in constraint graph
- PR #2926³⁰⁵⁸ - Default pool rename
- Issue #2925³⁰⁵⁹ - Default pool cannot be renamed
- Issue #2924³⁰⁶⁰ - hpx:attach-debugger=startup does not work any more
- PR #2923³⁰⁶¹ - Alloc membind
- PR #2922³⁰⁶² - This fixes CircleCI errors when running with -hpx:bind=none
- PR #2921³⁰⁶³ - Custom pool executor was missing priority and stacksize options
- PR #2920³⁰⁶⁴ - Adding test to trigger problem reported in #2916
- PR #2919³⁰⁶⁵ - Make sure the resource_partitioner is properly destructed on hpx::finalize
- Issue #2918³⁰⁶⁶ - hpx::init calls wrong (first) callback when called multiple times
- PR #2917³⁰⁶⁷ - Adding util::checkpoint
- Issue #2916³⁰⁶⁸ - Weird runtime failures when using a channel and chained continuations
- PR #2915³⁰⁶⁹ - Introduce executor parameters customization points
- Issue #2914³⁰⁷⁰ - Task assignment to current Pool has unintended consequences
- PR #2913³⁰⁷¹ - Fix rp hang
- PR #2912³⁰⁷² - Update contributors
- PR #2911³⁰⁷³ - Fixing CUDA problems
- PR #2910³⁰⁷⁴ - Improve error reporting for process component on POSIX systems
- PR #2909³⁰⁷⁵ - Fix typo in include path
- PR #2908³⁰⁷⁶ - Use proper container according to iterator tag in benchmarks of parallel algorithms
- PR #2907³⁰⁷⁷ - Optionally force-delete remaining channel items on close
- PR #2906³⁰⁷⁸ - Making sure generated performance counter names are correct

³⁰⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2928>

³⁰⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2927>

³⁰⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2926>

³⁰⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2925>

³⁰⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2924>

³⁰⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/2923>

³⁰⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2922>

³⁰⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/2921>

³⁰⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2920>

³⁰⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2919>

³⁰⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2918>

³⁰⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2917>

³⁰⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2916>

³⁰⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2915>

3070 <https://github.com/STELLAR-GROUP/hpx/issues/2914>3071 <https://github.com/STELLAR-GROUP/hpx/pull/2913>3072 <https://github.com/STELLAR-GROUP/hpx/pull/2912>3073 <https://github.com/STELLAR-GROUP/hpx/pull/2911>3074 <https://github.com/STELLAR-GROUP/hpx/pull/2910>3075 <https://github.com/STELLAR-GROUP/hpx/pull/2909>3076 <https://github.com/STELLAR-GROUP/hpx/pull/2908>3077 <https://github.com/STELLAR-GROUP/hpx/pull/2907>3078 <https://github.com/STELLAR-GROUP/hpx/pull/2906>

- Issue #2905³⁰⁷⁹ - collecting idle-rate performance counters on multiple localities produces an error
- Issue #2904³⁰⁸⁰ - build broken for Intel 17 compilers
- PR #2903³⁰⁸¹ - Documentation Updates– Adding New People
- PR #2902³⁰⁸² - Fixing service_executor
- PR #2901³⁰⁸³ - Fixing partitioned_vector creation
- PR #2900³⁰⁸⁴ - Add numa-balanced mode to hpx::bind, spread cores over numa domains
- Issue #2899³⁰⁸⁵ - hpx::bind does not have a mode that balances cores over numa domains
- PR #2898³⁰⁸⁶ - Adding missing #include and missing guard for optional code section
- PR #2897³⁰⁸⁷ - Removing dependency on Boost. ICL
- Issue #2896³⁰⁸⁸ - Debug build fails without -fpermissive with GCC 7.1 and Boost 1.65
- PR #2895³⁰⁸⁹ - Fixing SLURM environment parsing
- PR #2894³⁰⁹⁰ - Fix incorrect handling of compile definition with value 0
- Issue #2893³⁰⁹¹ - Disabling schedulers causes build errors
- PR #2892³⁰⁹² - added list serializer
- PR #2891³⁰⁹³ - Resource Partitioner Fixes
- Issue #2890³⁰⁹⁴ - Destroying a non-empty channel causes an assertion failure
- PR #2889³⁰⁹⁵ - Add check for libatomic
- PR #2888³⁰⁹⁶ - Fix compilation problems if HPX_WITH_ITT_NOTIFY=ON
- PR #2887³⁰⁹⁷ - Adapt broadcast() to non-unwrapping async<Action>
- PR #2886³⁰⁹⁸ - Replace Boost.Random with C++11 <random>
- Issue #2885³⁰⁹⁹ - regression in broadcast?
- Issue #2884³¹⁰⁰ - linking -latomic is not portable
- PR #2883³¹⁰¹ - Explicitly set -pthread flag if available

³⁰⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2905>

³⁰⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2904>

³⁰⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2903>

³⁰⁸² <https://github.com/STELLAR-GROUP/hpx/pull/2902>

³⁰⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2901>

³⁰⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2900>

³⁰⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2899>

³⁰⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2898>

³⁰⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2897>

³⁰⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2896>

³⁰⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2895>

³⁰⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2894>

³⁰⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/2893>

³⁰⁹² <https://github.com/STELLAR-GROUP/hpx/pull/2892>

³⁰⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2891>

³⁰⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2890>

³⁰⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2889>

³⁰⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2888>

³⁰⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2887>

³⁰⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2886>

³⁰⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2885>

³¹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2884>

³¹⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2883>

- PR #2882³¹⁰² - Wrap boost::format uses
- Issue #2881³¹⁰³ - hpx not compiling with HPX_WITH_ITTNOTIFY=On
- Issue #2880³¹⁰⁴ - hpx::bind scatter/balanced give wrong pu masks
- PR #2878³¹⁰⁵ - Fix incorrect pool usage masks setup in RP/thread manager
- PR #2877³¹⁰⁶ - Require std::array by default
- PR #2875³¹⁰⁷ - Deprecate use of BOOST_ASSERT
- PR #2874³¹⁰⁸ - Changed serialization of boost.variant to use variadic templates
- Issue #2873³¹⁰⁹ - building with parcelport_mpi fails on cori
- PR #2871³¹¹⁰ - Adding missing support for throttling scheduler
- PR #2870³¹¹¹ - Disambiguate use of base_lco_with_value macros with channel
- Issue #2869³¹¹² - Difficulty compiling HPX_REGISTER_CHANNEL_DECLARATION(double)
- PR #2868³¹¹³ - Removing unneeded assert
- PR #2867³¹¹⁴ - Implement parallel::unique
- Issue #2866³¹¹⁵ - The chunk_size_iterator violates multipass guarantee
- PR #2865³¹¹⁶ - Only use sched_getcpu on linux machines
- PR #2864³¹¹⁷ - Create redistribution archive for successful builds
- PR #2863³¹¹⁸ - Replace casts/assignments with hard-coded memcpy operations
- Issue #2862³¹¹⁹ - sched_getcpu not available on MacOS
- PR #2861³¹²⁰ - Fixing unmatched header defines and recursive inclusion of threadmanager
- Issue #2860³¹²¹ - Master program fails with assertion ‘type == data_type_address’ failed: HPX(assertion_failure)
- Issue #2852³¹²² - Support for ARM64
- PR #2858³¹²³ - Fix misplaced #if #endif's that cause build failure without THREAD_CUMULATIVE_COUNTS

³¹⁰² <https://github.com/STELLAR-GROUP/hpx/pull/2882>

³¹⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/2881>

³¹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2880>

³¹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2878>

³¹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2877>

³¹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2875>

³¹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2874>

³¹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2873>

³¹¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2871>

³¹¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2870>

³¹¹² <https://github.com/STELLAR-GROUP/hpx/issues/2869>

³¹¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2868>

³¹¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2867>

³¹¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2866>

³¹¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2865>

³¹¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2864>

³¹¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2863>

³¹¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2862>

³¹²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2861>

³¹²¹ <https://github.com/STELLAR-GROUP/hpx/issues/2860>

³¹²² <https://github.com/STELLAR-GROUP/hpx/issues/2852>

³¹²³ <https://github.com/STELLAR-GROUP/hpx/pull/2858>

- PR #2857³¹²⁴ - Fix some listing in documentation
- PR #2856³¹²⁵ - Fixing component handling for lcos
- PR #2855³¹²⁶ - Add documentation for coarrays
- PR #2854³¹²⁷ - Support ARM64 in timestamps
- PR #2853³¹²⁸ - Update Table 17. Non-modifying Parallel Algorithms in Documentation
- PR #2851³¹²⁹ - Allowing for non-default-constructible component types
- PR #2850³¹³⁰ - Enable returning future<R> from actions where R is not default-constructible
- PR #2849³¹³¹ - Unify serialization of non-default-constructable types
- Issue #2848³¹³² - Components have to be default constructible
- Issue #2847³¹³³ - Returning a future<R> where R is not default-constructable broken
- Issue #2846³¹³⁴ - Unify serialization of non-default-constructible types
- PR #2845³¹³⁵ - Add Visual Studio 2015 to the tested toolchains in Appveyor
- Issue #2844³¹³⁶ - Change the appveyor build to use the minimal required MSVC version
- Issue #2843³¹³⁷ - multi node hello_world hangs
- PR #2842³¹³⁸ - Correcting Spelling mistake in docs
- PR #2841³¹³⁹ - Fix usage of std::aligned_storage
- PR #2840³¹⁴⁰ - Remove constexpr from a void function
- Issue #2839³¹⁴¹ - memcpy buffer overflow: load_construct_data() and std::complex members
- Issue #2835³¹⁴² - constexpr functions with void return type break compilation with CUDA 8.0
- Issue #2834³¹⁴³ - One suspicion in parallel::detail::handle_exception
- PR #2833³¹⁴⁴ - Implement parallel::merge
- PR #2832³¹⁴⁵ - Fix a strange thing in parallel::util::detail::handle_local_exceptions. (Fix #2818)
- PR #2830³¹⁴⁶ - Break the debugger when a test failed

³¹²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2857>

³¹²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2856>

³¹²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2855>

³¹²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2854>

³¹²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2853>

³¹²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2851>

³¹³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2850>

³¹³¹ <https://github.com/STELLAR-GROUP/hpx/pull/2849>

³¹³² <https://github.com/STELLAR-GROUP/hpx/issues/2848>

³¹³³ <https://github.com/STELLAR-GROUP/hpx/issues/2847>

³¹³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2846>

³¹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2845>

³¹³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2844>

³¹³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2843>

³¹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2842>

³¹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2841>

³¹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2840>

³¹⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/2839>

³¹⁴² <https://github.com/STELLAR-GROUP/hpx/issues/2835>

³¹⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/2834>

³¹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2833>

³¹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2832>

³¹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2830>

- Issue #2831³¹⁴⁷ - parallel/executors/execution_fwd.hpp causes compilation failure in C++11 mode.
- PR #2829³¹⁴⁸ - Implement an API for asynchronous pack traversal
- PR #2828³¹⁴⁹ - Split unit test builds on CircleCI to avoid timeouts
- Issue #2827³¹⁵⁰ - failure to compile hello_world example with -Werror
- PR #2824³¹⁵¹ - Making sure promises are marked as started when used as continuations
- PR #2823³¹⁵² - Add documentation for partitioned_vector_view
- Issue #2822³¹⁵³ - Yet another issue with wait_for similar to #2796
- PR #2821³¹⁵⁴ - Fix bugs and improve that about HPX_HAVE_CXX11_AUTO_RETURN_VALUE of CMake
- PR #2820³¹⁵⁵ - Support C++11 in benchmark codes of parallel::partition and parallel::partition_copy
- PR #2819³¹⁵⁶ - Fix compile errors in unit test of container version of parallel::partition
- Issue #2818³¹⁵⁷ - A strange thing in parallel::util::detail::handle_local_exceptions
- Issue #2815³¹⁵⁸ - HPX fails to compile with HPX_WITH_CUDA=ON and the new CUDA 9.0 RC
- Issue #2814³¹⁵⁹ - Using ‘gmakeN’ after ‘cmake’ produces error in src/CMakeFiles/hpx.dir/runtime/agas/addressing_service.cpp.o
- PR #2813³¹⁶⁰ - Properly support [[noreturn]] attribute if available
- Issue #2812³¹⁶¹ - Compilation fails with gcc 7.1.1
- PR #2811³¹⁶² - Adding hpx::launch::lazy and support for async, dataflow, and future::then
- PR #2810³¹⁶³ - Add option allowing to disable deprecation warning
- PR #2809³¹⁶⁴ - Disable throttling scheduler if HWLOC is not found/used
- PR #2808³¹⁶⁵ - Fix compile errors on some environments of parallel::partition
- Issue #2807³¹⁶⁶ - Difficulty building with HPX_WITH_HWLOC=Off
- PR #2806³¹⁶⁷ - Partitioned vector
- PR #2805³¹⁶⁸ - Serializing collections with non-default constructible data

³¹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2831>

³¹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2829>

³¹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2828>

³¹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2827>

³¹⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/2824>

³¹⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2823>

³¹⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/2822>

³¹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2821>

³¹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2820>

³¹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2819>

³¹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2818>

³¹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2815>

³¹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2814>

³¹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2813>

³¹⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/2812>

³¹⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2811>

³¹⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/2810>

³¹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2809>

³¹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2808>

³¹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2807>

³¹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2806>

³¹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2805>

- PR #2802³¹⁶⁹ - Fix FreeBSD 11
- Issue #2801³¹⁷⁰ - Rate limiting techniques in io_service
- Issue #2800³¹⁷¹ - New Launch Policy: async_if
- PR #2799³¹⁷² - Fix a unit test failure on GCC in tuple_cat
- PR #2798³¹⁷³ - bump minimum required cmake to 3.0 in test
- PR #2797³¹⁷⁴ - Making sure future::wait_for et.al. work properly for action results
- Issue #2796³¹⁷⁵ - wait_for does always in “deferred” state for calls on remote localities
- Issue #2795³¹⁷⁶ - Serialization of types without default constructor
- PR #2794³¹⁷⁷ - Fixing test for partitioned_vector iteration
- PR #2792³¹⁷⁸ - Implemented segmented find and its variations for partitioned vector
- PR #2791³¹⁷⁹ - Circumvent scary warning about placement new
- PR #2790³¹⁸⁰ - Fix OSX build
- PR #2789³¹⁸¹ - Resource partitioner
- PR #2788³¹⁸² - Adapt parallel::is_heap and parallel::is_heap_until to Ranges TS
- PR #2787³¹⁸³ - Unwrap hotfixes
- PR #2786³¹⁸⁴ - Update CMake Minimum Version to 3.3.2 (refs #2565)
- Issue #2785³¹⁸⁵ - Issues with masks and cpuset
- PR #2784³¹⁸⁶ - Error with reduce and transform reduce fixed
- PR #2783³¹⁸⁷ - StackOverflow integration with libsigsegv
- PR #2782³¹⁸⁸ - Replace boost::atomic with std::atomic (where possible)
- PR #2781³¹⁸⁹ - Check for and optionally use [[deprecated]] attribute
- PR #2780³¹⁹⁰ - Adding empty (but non-trivial) destructor to circumvent warnings
- PR #2779³¹⁹¹ - Exception info tweaks

³¹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2802>

³¹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2801>

³¹⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/2800>

³¹⁷² <https://github.com/STELLAR-GROUP/hpx/pull/2799>

³¹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2798>

³¹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2797>

³¹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2796>

³¹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2795>

³¹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2794>

³¹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2792>

³¹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2791>

³¹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2790>

³¹⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2789>

³¹⁸² <https://github.com/STELLAR-GROUP/hpx/pull/2788>

³¹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2787>

³¹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2786>

³¹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2785>

³¹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2784>

³¹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2783>

³¹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2782>

³¹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2781>

³¹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2780>

³¹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2779>

- PR #2778³¹⁹² - Implement parallel::partition
- PR #2777³¹⁹³ - Improve error handling in gather_here/gather_there
- PR #2776³¹⁹⁴ - Fix a bug in compiler version check
- PR #2775³¹⁹⁵ - Fix compilation when HPX_WITH_LOGGING is OFF
- PR #2774³¹⁹⁶ - Removing dependency on Boost.Date_Time
- PR #2773³¹⁹⁷ - Add sync_images() method to spmd_block class
- PR #2772³¹⁹⁸ - Adding documentation for PAPI counters
- PR #2771³¹⁹⁹ - Removing boost preprocessor dependency
- PR #2770³²⁰⁰ - Adding test, fixing deadlock in config registry
- PR #2769³²⁰¹ - Remove some other warnings and errors detected by clang 5.0
- Issue #2768³²⁰² - Is there iterator tag for HPX?
- PR #2767³²⁰³ - Improvements to continuation annotation
- PR #2765³²⁰⁴ - gcc split stack support for HPX threads #620
- PR #2764³²⁰⁵ - Fix some uses of begin/end, remove unnecessary includes
- PR #2763³²⁰⁶ - Bump minimal Boost version to 1.55.0
- PR #2762³²⁰⁷ - hpx::partitioned_vector serializer
- PR #2761³²⁰⁸ - Adding configuration summary to cmake output and -hpx:info
- PR #2760³²⁰⁹ - Removing 1d_hydro example as it is broken
- PR #2758³²¹⁰ - Remove various warnings detected by clang 5.0
- Issue #2757³²¹¹ - In case of a “raw thread” is needed per core for implementing parallel algorithm, what is good practice in HPX?
- PR #2756³²¹² - Allowing for LCOs to be simple components
- PR #2755³²¹³ - Removing make_index_pack_unrolled
- PR #2754³²¹⁴ - Implement parallel::unique_copy

³¹⁹² <https://github.com/STELLAR-GROUP/hpx/pull/2778>

³¹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2777>

³¹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2776>

³¹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2775>

³¹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2774>

³¹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2773>

³¹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2772>

³¹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2771>

³²⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2770>

³²⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2769>

³²⁰² <https://github.com/STELLAR-GROUP/hpx/issues/2768>

³²⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2767>

³²⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2765>

³²⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2764>

³²⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2763>

³²⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2762>

³²⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2761>

³²⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2760>

³²¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2758>

³²¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/2757>

³²¹² <https://github.com/STELLAR-GROUP/hpx/pull/2756>

³²¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2755>

³²¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2754>

- PR #2753³²¹⁵ - Fixing detection of [[fallthrough]] attribute
- PR #2752³²¹⁶ - New thread priority names
- PR #2751³²¹⁷ - Replace boost::exception with proposed exception_info
- PR #2750³²¹⁸ - Replace boost::iterator_range
- PR #2749³²¹⁹ - Fixing hdf5 examples
- Issue #2748³²²⁰ - HPX fails to build with enabled hdf5 examples
- Issue #2747³²²¹ - Inherited task priorities break certain DAG optimizations
- Issue #2746³²²² - HPX segfaulting with valgrind
- PR #2745³²²³ - Adding extended arithmetic performance counters
- PR #2744³²²⁴ - Adding ability to statistics counters to reset base counter
- Issue #2743³²²⁵ - Statistics counter does not support resetting
- PR #2742³²²⁶ - Making sure Vc V2 builds without additional HPX configuration flags
- PR #2741³²²⁷ - Deprecate unwrapped and implement unwrap and unwrapping
- PR #2740³²²⁸ - Coroutine stackoverflow detection for linux/posix; Issue #2408
- PR #2739³²²⁹ - Add files via upload
- PR #2738³²³⁰ - Appveyor support
- PR #2737³²³¹ - Fixing 2735
 - Issue #2736³²³² - 1d_hydro example doesn't work
 - Issue #2735³²³³ - partitioned_vector_subview test failing
- PR #2734³²³⁴ - Add C++11 range utilities
- PR #2733³²³⁵ - Adapting iterator requirements for parallel algorithms
- PR #2732³²³⁶ - Integrate C++ Co-arrays
- PR #2731³²³⁷ - Adding on_migrated event handler to migratable component instances

³²¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2753>

³²¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2752>

³²¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2751>

³²¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2750>

³²¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2749>

³²²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2748>

³²²¹ <https://github.com/STELLAR-GROUP/hpx/issues/2747>

³²²² <https://github.com/STELLAR-GROUP/hpx/issues/2746>

³²²³ <https://github.com/STELLAR-GROUP/hpx/pull/2745>

³²²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2744>

³²²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2743>

³²²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2742>

³²²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2741>

³²²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2740>

³²²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2739>

³²³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2738>

³²³¹ <https://github.com/STELLAR-GROUP/hpx/pull/2737>

³²³² <https://github.com/STELLAR-GROUP/hpx/issues/2736>

³²³³ <https://github.com/STELLAR-GROUP/hpx/issues/2735>

³²³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2734>

³²³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2733>

³²³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2732>

³²³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2731>

- Issue #2729³²³⁸ - Add on_migrated() event handler to migratable components
- Issue #2728³²³⁹ - Why Projection is needed in parallel algorithms?
- PR #2727³²⁴⁰ - Cmake files for StackOverflow Detection
- PR #2726³²⁴¹ - CMake for Stack Overflow Detection
- PR #2725³²⁴² - Implemented segmented algorithms for partitioned vector
- PR #2724³²⁴³ - Fix examples in Action documentation
- PR #2723³²⁴⁴ - Enable lcos::channel<T>::register_as
- Issue #2722³²⁴⁵ - channel register_as() failing on compilation
- PR #2721³²⁴⁶ - Mind map
- PR #2720³²⁴⁷ - reorder forward declarations to get rid of C++14-only auto return types
- PR #2719³²⁴⁸ - Add documentation for partitioned_vector and add features in pack.hpp
- Issue #2718³²⁴⁹ - Some forward declarations in execution_fwd.hpp aren't C++11-compatible
- PR #2717³²⁵⁰ - Config support for fallthrough attribute
- PR #2716³²⁵¹ - Implement parallel::partition_copy
- PR #2715³²⁵² - initial import of icu string serializer
- PR #2714³²⁵³ - initial import of valarray serializer
- PR #2713³²⁵⁴ - Remove slashes before CMAKE_FILES_DIRECTORY variables
- PR #2712³²⁵⁵ - Fixing wait for 1751
- PR #2711³²⁵⁶ - Adjust code for minimal supported GCC having been bumped to 4.9
- PR #2710³²⁵⁷ - Adding code of conduct
- PR #2709³²⁵⁸ - Fixing UB in destroy tests
- PR #2708³²⁵⁹ - Add inline to prevent multiple definition issue
- Issue #2707³²⁶⁰ - Multiple defined symbols for task_block.hpp in VS2015

³²³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2729>

³²³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2728>

³²⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2727>

³²⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/2726>

³²⁴² <https://github.com/STELLAR-GROUP/hpx/pull/2725>

³²⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/2724>

³²⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2723>

³²⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2722>

³²⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2721>

³²⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2720>

³²⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2719>

³²⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2718>

³²⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2717>

³²⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/2716>

³²⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2715>

³²⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/2714>

³²⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2713>

³²⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2712>

³²⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2711>

³²⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2710>

³²⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2709>

³²⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2708>

³²⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2707>

- PR #2706³²⁶¹ - Adding .clang-format file
- PR #2704³²⁶² - Add a synchronous mapping API
- Issue #2703³²⁶³ - Request: Add the .clang-format file to the repository
- Issue #2702³²⁶⁴ - STELLAR-GROUP/Vc slower than VCv1 possibly due to wrong instructions generated
- Issue #2701³²⁶⁵ - Datapar with STELLAR-GROUP/Vc requires obscure flag
- Issue #2700³²⁶⁶ - Naming inconsistency in parallel algorithms
- Issue #2699³²⁶⁷ - Iterator requirements are different from standard in parallel copy_if.
- PR #2698³²⁶⁸ - Properly releasing parcelport write handlers
- Issue #2697³²⁶⁹ - Compile error in addressing_service.cpp
- Issue #2696³²⁷⁰ - Building and using HPX statically: undefined references from runtime_support_server.cpp
- Issue #2695³²⁷¹ - Executor changes cause compilation failures
- PR #2694³²⁷² - Refining C++ language mode detection for MSVC
- PR #2693³²⁷³ - P0443 r2
- PR #2692³²⁷⁴ - Partially reverting changes to parcel_await
- Issue #2689³²⁷⁵ - HPX build fails when HPX_WITH_CUDA is enabled
- PR #2688³²⁷⁶ - Make Cuda Clang builds pass
- PR #2687³²⁷⁷ - Add an is_tuple_like trait for sequenceable type detection
- PR #2686³²⁷⁸ - Allowing throttling scheduler to be used without idle backoff
- PR #2685³²⁷⁹ - Add support of std::array to hpx::util::tuple_size and tuple_element
- PR #2684³²⁸⁰ - Adding new statistics performance counters
- PR #2683³²⁸¹ - Replace boost::exception_ptr with std::exception_ptr
- Issue #2682³²⁸² - HPX does not compile with HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF=OFF
- PR #2681³²⁸³ - Attempt to fix problem in managed_component_base

³²⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/2706>

³²⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2704>

³²⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/2703>

³²⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2702>

³²⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2701>

³²⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2700>

³²⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2699>

³²⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2698>

³²⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2697>

³²⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2696>

³²⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/2695>

³²⁷² <https://github.com/STELLAR-GROUP/hpx/pull/2694>

³²⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2693>

³²⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2692>

³²⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2689>

³²⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2688>

³²⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2687>

³²⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2686>

³²⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2685>

³²⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2684>

³²⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2683>

³²⁸² <https://github.com/STELLAR-GROUP/hpx/issues/2682>

³²⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2681>

- PR #2680³²⁸⁴ - Fix bad size during archive creation
- Issue #2679³²⁸⁵ - Mismatch between size of archive and container
- Issue #2678³²⁸⁶ - In parallel algorithm, other tasks are executed to the end even if an exception occurs in any task.
- PR #2677³²⁸⁷ - Adding include check for std::addressof
- PR #2676³²⁸⁸ - Adding parallel::destroy and destroy_n
- PR #2675³²⁸⁹ - Making sure statistics counters work as expected
- PR #2674³²⁹⁰ - Turning assertions into exceptions
- PR #2673³²⁹¹ - Inhibit direct conversion from future<future<T>> -> future<void>
- PR #2672³²⁹² - C++17 invoke forms
- PR #2671³²⁹³ - Adding uninitialized_value_construct and uninitialized_value_construct_n
- PR #2670³²⁹⁴ - Integrate spmd multidimensional views for partitioned_vectors
- PR #2669³²⁹⁵ - Adding uninitialized_default_construct and uninitialized_default_construct_n
- PR #2668³²⁹⁶ - Fixing documentation index
- Issue #2667³²⁹⁷ - Ambiguity of nested hpx::future<void>'s
- Issue #2666³²⁹⁸ - Statistics Performance counter is not working
- PR #2664³²⁹⁹ - Adding uninitialized_move and uninitialized_move_n
- Issue #2663³³⁰⁰ - Seg fault in managed_component::get_base_gid, possibly cause by util::reinitializable_static
- Issue #2662³³⁰¹ - Crash in managed_component::get_base_gid due to problem with util::reinitializable_static
- PR #2665³³⁰² - Hide the detail namespace in doxygen per default
- PR #2660³³⁰³ - Add documentation to hpx::util::unwrapped and hpx::util::unwrapped2
- PR #2659³³⁰⁴ - Improve integration with vcpkg
- PR #2658³³⁰⁵ - Unify access_data trait for use in both, serialization and de-serialization
- PR #2657³³⁰⁶ - Removing hpx::lcos::queue<T>

³²⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2680>

³²⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2679>

³²⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2678>

³²⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2677>

³²⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2676>

³²⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2675>

³²⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2674>

³²⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2673>

³²⁹² <https://github.com/STELLAR-GROUP/hpx/pull/2672>

³²⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2671>

³²⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2670>

³²⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2669>

³²⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2668>

³²⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2667>

³²⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2666>

³²⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2664>

³³⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2663>

³³⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/2662>

³³⁰² <https://github.com/STELLAR-GROUP/hpx/pull/2665>

³³⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2660>

³³⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2659>

³³⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2658>

³³⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2657>

- PR #2656³³⁰⁷ - Reduce MAX_TERMINATED_THREADS default, improve memory use on manycore cpus
- PR #2655³³⁰⁸ - Maintenance for emulate-deleted macros
- PR #2654³³⁰⁹ - Implement parallel is_heap and is_heap_until
- PR #2653³³¹⁰ - Drop support for VS2013
- PR #2652³³¹¹ - This patch makes sure that all parcels in a batch are properly handled
- PR #2649³³¹² - Update docs (Table 18) - move transform to end
- Issue #2647³³¹³ - hpx::parcelset::detail::parcel_data::has_continuation_ is uninitialized
- Issue #2644³³¹⁴ - Some .vcxproj in the HPX.sln fail to build
- Issue #2641³³¹⁵ - hpx::lcos::queue should be deprecated
- PR #2640³³¹⁶ - A new throttling policy with public APIs to suspend/resume
- PR #2639³³¹⁷ - Fix a tiny typo in tutorial.
- Issue #2638³³¹⁸ - Invalid return type ‘void’ of constexpr function
- PR #2636³³¹⁹ - Add and use HPX_MSVC_WARNING_PRAGMA for #pragma warning
- PR #2633³³²⁰ - Distributed define_spmd_block
- PR #2632³³²¹ - Making sure container serialization uses size-compatible types
- PR #2631³³²² - Add lcos::local::one_element_channel
- PR #2629³³²³ - Move unordered_map out of parcelport into hpx/concurrent
- PR #2628³³²⁴ - Making sure that shutdown does not hang
- PR #2627³³²⁵ - Fix serialization
- PR #2626³³²⁶ - Generate cmake_variables.qbk and cmake_toolchains.qbk outside of the source tree
- PR #2625³³²⁷ - Supporting -std=c++17 flag
- PR #2624³³²⁸ - Fixing a small cmake typo
- PR #2622³³²⁹ - Update CMake minimum required version to 3.0.2 (closes #2621)

³³⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2656>

³³⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2655>

³³⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2654>

³³¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2653>

³³¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2652>

³³¹² <https://github.com/STELLAR-GROUP/hpx/pull/2649>

³³¹³ <https://github.com/STELLAR-GROUP/hpx/issues/2647>

³³¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2644>

³³¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2641>

³³¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2640>

³³¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2639>

³³¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2638>

³³¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2636>

³³²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2633>

³³²¹ <https://github.com/STELLAR-GROUP/hpx/pull/2632>

³³²² <https://github.com/STELLAR-GROUP/hpx/pull/2631>

³³²³ <https://github.com/STELLAR-GROUP/hpx/pull/2629>

³³²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2628>

³³²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2627>

³³²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2626>

³³²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2625>

³³²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2624>

³³²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2622>

- Issue #2621³³³⁰ - Compiling hpx master fails with /usr/bin/ld: final link failed: Bad value
- PR #2620³³³¹ - Remove warnings due to some captured variables
- PR #2619³³³² - LF multiple parcels
- PR #2618³³³³ - Some fixes to libfabric that didn't get caught before the merge
- PR #2617³³³⁴ - Adding hpx::local_new
- PR #2616³³³⁵ - Documentation: Extract all entities in order to autolink functions correctly
- Issue #2615³³³⁶ - Documentation: Linking functions is broken
- PR #2614³³³⁷ - Adding serialization for std::deque
- PR #2613³³³⁸ - We need to link with boost.thread and boost.chrono if we use boost.context
- PR #2612³³³⁹ - Making sure for_loop_n(par, ...) is actually executed in parallel
- PR #2611³³⁴⁰ - Add documentation to invoke_fused and friends NFC
- PR #2610³³⁴¹ - Added reduction templates using an identity value
- PR #2608³³⁴² - Fixing some unused vars in inspect
- PR #2607³³⁴³ - Fixed build for mingw
- PR #2606³³⁴⁴ - Supporting generic context for boost >= 1.61
- PR #2605³³⁴⁵ - Parcelport libfabric3
- PR #2604³³⁴⁶ - Adding allocator support to promise and friends
- PR #2603³³⁴⁷ - Barrier hang
- PR #2602³³⁴⁸ - Changes to scheduler to steal from one high-priority queue
- Issue #2601³³⁴⁹ - High priority tasks are not executed first
- PR #2600³³⁵⁰ - Compat fixes
- PR #2599³³⁵¹ - Compatibility layer for threading support
- PR #2598³³⁵² - V1.1

³³³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2621>

³³³¹ <https://github.com/STELLAR-GROUP/hpx/pull/2620>

³³³² <https://github.com/STELLAR-GROUP/hpx/pull/2619>

³³³³ <https://github.com/STELLAR-GROUP/hpx/pull/2618>

³³³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2617>

³³³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2616>

³³³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2615>

³³³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2614>

³³³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2613>

3339 <https://github.com/STELLAR-GROUP/hpx/pull/2612>3340 <https://github.com/STELLAR-GROUP/hpx/pull/2611>3341 <https://github.com/STELLAR-GROUP/hpx/pull/2610>3342 <https://github.com/STELLAR-GROUP/hpx/pull/2608>3343 <https://github.com/STELLAR-GROUP/hpx/pull/2607>3344 <https://github.com/STELLAR-GROUP/hpx/pull/2606>3345 <https://github.com/STELLAR-GROUP/hpx/pull/2605>3346 <https://github.com/STELLAR-GROUP/hpx/pull/2604>3347 <https://github.com/STELLAR-GROUP/hpx/pull/2603>3348 <https://github.com/STELLAR-GROUP/hpx/pull/2602>3349 <https://github.com/STELLAR-GROUP/hpx/issues/2601>3350 <https://github.com/STELLAR-GROUP/hpx/pull/2600>3351 <https://github.com/STELLAR-GROUP/hpx/pull/2599>3352 <https://github.com/STELLAR-GROUP/hpx/pull/2598>

- PR #2597³³⁵³ - Release V1.0
- PR #2592³³⁵⁴ - First attempt to introduce spmd_block in hpx
- PR #2586³³⁵⁵ - local_segment in segmented_iterator_traits
- Issue #2584³³⁵⁶ - Add allocator support to promise, packaged_task and friends
- PR #2576³³⁵⁷ - Add missing dependencies of cuda based tests
- PR #2575³³⁵⁸ - Remove warnings due to some captured variables
- Issue #2574³³⁵⁹ - MSVC 2015 Compiler crash when building HPX
- Issue #2568³³⁶⁰ - Remove throttle_scheduler as it has been abandoned
- Issue #2566³³⁶¹ - Add an inline versioning namespace before 1.0 release
- Issue #2565³³⁶² - Raise minimal cmake version requirement
- PR #2556³³⁶³ - Fixing scan partitioner
- PR #2546³³⁶⁴ - Broadcast async
- Issue #2543³³⁶⁵ - make install fails due to a non-existing .so file
- PR #2495³³⁶⁶ - wait_or_add_new returning thread_id_type
- Issue #2480³³⁶⁷ - Unable to register new performance counter
- Issue #2471³³⁶⁸ - no type named ‘fcontext_t’ in namespace
- Issue #2456³³⁶⁹ - Re-implement hpx::util::unwrapped
- Issue #2455³³⁷⁰ - Add more arithmetic performance counters
- PR #2454³³⁷¹ - Fix a couple of warnings and compiler errors
- PR #2453³³⁷² - Timed executor support
- PR #2447³³⁷³ - Implementing new executor API (P0443)
- Issue #2439³³⁷⁴ - Implement executor proposal
- Issue #2408³³⁷⁵ - Stackoverflow detection for linux, e.g. based on libsigsev

³³⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/2597>

³³⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2592>

³³⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2586>

³³⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2584>

³³⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2576>

³³⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2575>

³³⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2574>

³³⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2568>

³³⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/2566>

³³⁶² <https://github.com/STELLAR-GROUP/hpx/issues/2565>

³³⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/2556>

³³⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2546>

³³⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2543>

³³⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2495>

³³⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2480>

³³⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2471>

³³⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2456>

³³⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2455>

³³⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/2454>

³³⁷² <https://github.com/STELLAR-GROUP/hpx/pull/2453>

³³⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2447>

³³⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2439>

³³⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2408>

- PR #2377³³⁷⁶ - Add a customization point for put_parcel so we can override actions
- Issue #2368³³⁷⁷ - HPX_ASSERT problem
- Issue #2324³³⁷⁸ - Change default number of threads used to the maximum of the system
- Issue #2266³³⁷⁹ - hpx_0.9.99 make tests fail
- PR #2195³³⁸⁰ - Support for code completion in VIM
- Issue #2137³³⁸¹ - Hpx does not compile over osx
- Issue #2092³³⁸² - make tests should just build the tests
- Issue #2026³³⁸³ - Build HPX with Apple's clang
- Issue #1932³³⁸⁴ - hpx with PBS fails on multiple localities
- PR #1914³³⁸⁵ - Parallel heap algorithm implementations WIP
- Issue #1598³³⁸⁶ - Disconnecting a locality results in segfault using heartbeat example
- Issue #1404³³⁸⁷ - unwrapped doesn't work with movable only types
- Issue #1400³³⁸⁸ - hpx::util::unwrapped doesn't work with non-future types
- Issue #1205³³⁸⁹ - TSS is broken
- Issue #1126³³⁹⁰ - vector<future<T>> does not work gracefully with dataflow, when_all and unwrapped
- Issue #1056³³⁹¹ - Thread manager cleanup
- Issue #863³³⁹² - Futures should not require a default constructor
- Issue #856³³⁹³ - Allow runtimemode_connect to be used with security enabled
- Issue #726³³⁹⁴ - Valgrind
- Issue #701³³⁹⁵ - Add RCR performance counter component
- Issue #528³³⁹⁶ - Add support for known failures and warning count/comparisons to hpx_run_tests.py

³³⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2377>

³³⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2368>

³³⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2324>

³³⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2266>

³³⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2195>

³³⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/2137>

³³⁸² <https://github.com/STELLAR-GROUP/hpx/issues/2092>

³³⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/2026>

³³⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1932>

³³⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1914>

3386 <https://github.com/STELLAR-GROUP/hpx/issues/1598>3387 <https://github.com/STELLAR-GROUP/hpx/issues/1404>3388 <https://github.com/STELLAR-GROUP/hpx/issues/1400>3389 <https://github.com/STELLAR-GROUP/hpx/issues/1205>3390 <https://github.com/STELLAR-GROUP/hpx/issues/1126>3391 <https://github.com/STELLAR-GROUP/hpx/issues/1056>3392 <https://github.com/STELLAR-GROUP/hpx/issues/863>3393 <https://github.com/STELLAR-GROUP/hpx/issues/856>3394 <https://github.com/STELLAR-GROUP/hpx/issues/726>3395 <https://github.com/STELLAR-GROUP/hpx/issues/701>3396 <https://github.com/STELLAR-GROUP/hpx/issues/528>

2.10.13 HPX V1.0.0 (Apr 24, 2017)

General changes

Here are some of the main highlights and changes for this release (in no particular order):

- Added the facility `hpx::split_future` which allows one to convert a `future<tuple<Ts...>>` into a `tuple<future<Ts>...>`. This functionality is not available when compiling *HPX* with VS2012.
- Added a new type of performance counter which allows one to return a list of values for each invocation. We also added a first counter of this type which collects a histogram of the times between parcels being created.
- Added new LCOs: `hpx::lcos::channel` and `hpx::lcos::local::channel` which are very similar to the well known channel constructs used in the Go language.
- Added new performance counters reporting the amount of data handled by the networking layer on a action-by-action basis (please see [PR #2289³³⁹⁷](#) for more details).
- Added a new facility `hpx::lcos::barrier`, replacing the equally named older one. The new facility has a slightly changed API and is much more efficient. Most notable, the new facility exposes a (global) function `hpx::lcos::barrier::synchronize()` which represents a global barrier across all localities.
- We have started to add support for vectorization to our parallel algorithm implementations. This support depends on using an external library, currently either Vc Library or [libboost_simd](#). Please see [Issue #2333³³⁹⁸](#) for a list of currently supported algorithms. This is an experimental feature and its implementation and/or API might change in the future. Please see this [blog-post³³⁹⁹](#) for more information.
- The parameter sequence for the `hpx::parallel::transform_reduce` overload taking one iterator range has changed to match the changes this algorithm has undergone while being moved to C++17. The old overload can be still enabled at configure time by specifying `-DHPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY=On` to CMake.
- The algorithm `hpx::parallel::inner_product` has been renamed to `hpx::parallel::transform_reduce` to match the changes this algorithm has undergone while being moved to C++17. The old `inner_product` names can be still enabled at configure time by specifying `-DHPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY=On` to CMake.
- Added versions of `hpx::get_ptr` taking client side representations for component instances as their parameter (instead of a global id).
- Added the helper utility `hpx::performance_counters::performance_counter_set` helping to encapsulate a set of performance counters to be managed concurrently.
- All execution policies and related classes have been renamed to be consistent with the naming changes applied for C++17. All policies now live in the namespace `hpx::parallel::execution`. The old names can be still enabled at configure time by specifying `-DHPX_WITH_EXECUTION_POLICY_COMPATIBILITY=On` to CMake.
- The thread scheduling subsystem has undergone a major refactoring which results in significant performance improvements. We have also improved the performance of creating `hpx::future` and of various facilities handling those.
- We have consolidated all of the code in `HPX.Compute` related to the integration of CUDA. `hpx::partitioned_vector` has been enabled to be usable with `hpx::compute::vector` which allows one to place the partitions on one or more GPU devices.
- Added new performance counters exposing various internals of the thread scheduling subsystem, such as the current idle- and busy-loop counters and instantaneous scheduler utilization.

³³⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2289>

³³⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2333>

³³⁹⁹ <http://stellar-group.org/2016/09/vectorized-cpp-parallel-algorithms-with-hpx/>

- Extended and improved the use of the ITTNotify hooks allowing to collect performance counter data and function annotation information from within the Intel Amplifier tool.

Breaking changes

- We have dropped support for the gcc compiler versions V4.6 and 4.7. The minimal gcc version we now test on is gcc V4.8.
- We have removed (default) support for `boost::chrono` in interfaces, uses of it have been replaced with `std::chrono`. This facility can be still enabled at configure time by specifying `-DHPX_WITH_BOOST_CHRONO_COMPATIBILITY=On` to CMake.
- The parameter sequence for the `hpx::parallel::transform_reduce` overload taking one iterator range has changed to match the changes this algorithm has undergone while being moved to C++17.
- The algorithm `hpx::parallel::inner_product` has been renamed to `hpx::parallel::transform_reduce` to match the changes this algorithm has undergone while being moved to C++17.
- the build options `HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY` and `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY` are now disabled by default. Please change your code still depending on the deprecated interfaces.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- PR #2596³⁴⁰⁰ - Adding apex data
- PR #2595³⁴⁰¹ - Remove obsolete file
- Issue #2594³⁴⁰² - FindOpenCL.cmake mismatch with the official cmake module
- PR #2592³⁴⁰³ - First attempt to introduce spmd_block in hpx
- Issue #2591³⁴⁰⁴ - Feature request: continuation (then) which does not require the callable object to take a future<R> as parameter
- PR #2588³⁴⁰⁵ - Daint fixes
- PR #2587³⁴⁰⁶ - Fixing transfer_(continuation)_action::schedule
- PR #2585³⁴⁰⁷ - Work around MSVC having an ICE when compiling with -Ob2
- PR #2583³⁴⁰⁸ - changing 7zip command to 7za in roll_release.sh
- PR #2582³⁴⁰⁹ - First attempt to introduce spmd_block in hpx
- PR #2581³⁴¹⁰ - Enable annotated function for parallel algorithms

³⁴⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2596>

³⁴⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2595>

³⁴⁰² <https://github.com/STELLAR-GROUP/hpx/issues/2594>

³⁴⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2592>

³⁴⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2591>

³⁴⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2588>

³⁴⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2587>

³⁴⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2585>

³⁴⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2583>

³⁴⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2582>

³⁴¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2581>

- PR #2580³⁴¹¹ - First attempt to introduce spmd_block in hpx
- PR #2579³⁴¹² - Make thread NICE level setting an option
- PR #2578³⁴¹³ - Implementing enqueue instead of busy wait when no sender is available
- PR #2577³⁴¹⁴ - Retrieve -std=c++11 consistent nvcc flag
- PR #2576³⁴¹⁵ - Add missing dependencies of cuda based tests
- PR #2575³⁴¹⁶ - Remove warnings due to some captured variables
- PR #2573³⁴¹⁷ - Attempt to resolve resolve_locality
- PR #2572³⁴¹⁸ - Adding APEX hooks to background thread
- PR #2571³⁴¹⁹ - Pick up hpx.ignore_batch_env from config map
- PR #2570³⁴²⁰ - Add commandline options –hpx:print-counters-locally
- PR #2569³⁴²¹ - Fix computeapi unit tests
- PR #2567³⁴²² - This adds another barrier::synchronize before registering performance counters
- PR #2564³⁴²³ - Cray static toolchain support
- PR #2563³⁴²⁴ - Fixed unhandled exception during startup
- PR #2562³⁴²⁵ - Remove partitioned_vector.cu from build tree when nvcc is used
- Issue #2561³⁴²⁶ - octo-tiger crash with commit 6e921495ff6c26f125d62629cbaad0525f14f7ab
- PR #2560³⁴²⁷ - Prevent -Wundef warnings on Vc version checks
- PR #2559³⁴²⁸ - Allowing CUDA callback to set the future directly from an OS thread
- PR #2558³⁴²⁹ - Remove warnings due to float precisions
- PR #2557³⁴³⁰ - Removing bogus handling of compile flags for CUDA
- PR #2556³⁴³¹ - Fixing scan partitioner
- PR #2554³⁴³² - Add more diagnostics to error thrown from find_appropriate_destination
- Issue #2555³⁴³³ - No valid parcelport configured

³⁴¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2580>

³⁴¹² <https://github.com/STELLAR-GROUP/hpx/pull/2579>

³⁴¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2578>

³⁴¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2577>

³⁴¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2576>

³⁴¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2575>

³⁴¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2573>

³⁴¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2572>

³⁴¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2571>

³⁴²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2570>

³⁴²¹ <https://github.com/STELLAR-GROUP/hpx/pull/2569>

³⁴²² <https://github.com/STELLAR-GROUP/hpx/pull/2567>

³⁴²³ <https://github.com/STELLAR-GROUP/hpx/pull/2564>

³⁴²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2563>

³⁴²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2562>

³⁴²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2561>

³⁴²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2560>

³⁴²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2559>

3429 <https://github.com/STELLAR-GROUP/hpx/pull/2558>3430 <https://github.com/STELLAR-GROUP/hpx/pull/2557>3431 <https://github.com/STELLAR-GROUP/hpx/pull/2556>3432 <https://github.com/STELLAR-GROUP/hpx/pull/2554>3433 <https://github.com/STELLAR-GROUP/hpx/issues/2555>

- PR #2553³⁴³⁴ - Add cmake cuda_arch option
- PR #2552³⁴³⁵ - Remove incomplete datapar bindings to libflatarray
- PR #2551³⁴³⁶ - Rename hwloc_topology to hwloc_topology_info
- PR #2550³⁴³⁷ - Apex api updates
- PR #2549³⁴³⁸ - Pre-include defines.hpp to get the macro HPX_HAVE_CUDA value
- PR #2548³⁴³⁹ - Fixing issue with disconnect
- PR #2546³⁴⁴⁰ - Some fixes around cuda clang partitioned_vector example
- PR #2545³⁴⁴¹ - Fix uses of the Vc2 datapar flags; the value, not the type, should be passed to functions
- PR #2542³⁴⁴² - Make HPX_WITH_MALLOC easier to use
- PR #2541³⁴⁴³ - avoid recompiles when enabling/disabling examples
- PR #2540³⁴⁴⁴ - Fixing usage of target_link_libraries()
- PR #2539³⁴⁴⁵ - fix RPATH behaviour
- Issue #2538³⁴⁴⁶ - HPX_WITH_CUDA corrupts compilation flags
- PR #2537³⁴⁴⁷ - Add output of a Bazel Skylark extension for paths and compile options
- PR #2536³⁴⁴⁸ - Add counter exposing total available memory to Windows as well
- PR #2535³⁴⁴⁹ - Remove obsolete support for security
- Issue #2534³⁴⁵⁰ - Remove command line option --hpx:run-agas-server
- PR #2533³⁴⁵¹ - Pre-cache locality endpoints during bootstrap
- PR #2532³⁴⁵² - Fixing handling of GIDs during serialization preprocessing
- PR #2531³⁴⁵³ - Amend uses of the term “functor”
- PR #2529³⁴⁵⁴ - added counter for reading available memory
- PR #2527³⁴⁵⁵ - Facilities to create actions from lambdas
- PR #2526³⁴⁵⁶ - Updated docs: HPX_WITH_EXAMPLES

³⁴³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2553>

³⁴³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2552>

³⁴³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2551>

³⁴³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2550>

³⁴³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2549>

³⁴³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2548>

³⁴⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2546>

³⁴⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/2545>

³⁴⁴² <https://github.com/STELLAR-GROUP/hpx/pull/2542>

³⁴⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/2541>

3444 <https://github.com/STELLAR-GROUP/hpx/pull/2540>3445 <https://github.com/STELLAR-GROUP/hpx/pull/2539>3446 <https://github.com/STELLAR-GROUP/hpx/issues/2538>3447 <https://github.com/STELLAR-GROUP/hpx/pull/2537>3448 <https://github.com/STELLAR-GROUP/hpx/pull/2536>3449 <https://github.com/STELLAR-GROUP/hpx/pull/2535>3450 <https://github.com/STELLAR-GROUP/hpx/issues/2534>3451 <https://github.com/STELLAR-GROUP/hpx/pull/2533>3452 <https://github.com/STELLAR-GROUP/hpx/pull/2532>3453 <https://github.com/STELLAR-GROUP/hpx/pull/2531>3454 <https://github.com/STELLAR-GROUP/hpx/pull/2529>3455 <https://github.com/STELLAR-GROUP/hpx/pull/2527>3456 <https://github.com/STELLAR-GROUP/hpx/pull/2526>

- PR #2525³⁴⁵⁷ - Remove warnings related to unused captured variables
- Issue #2524³⁴⁵⁸ - CMAKE failed because it is missing: TCMALLOC_LIBRARY TCMALLOC_INCLUDE_DIR
- PR #2523³⁴⁵⁹ - Fixing compose_cb stack overflow
- PR #2522³⁴⁶⁰ - Instead of unlocking, ignore the lock while creating the message handler
- PR #2521³⁴⁶¹ - Create LPROGRESS_ logging macro to simplify progress tracking and timings
- PR #2520³⁴⁶² - Intel 17 support
- PR #2519³⁴⁶³ - Fix components example
- PR #2518³⁴⁶⁴ - Fixing parcel scheduling
- Issue #2517³⁴⁶⁵ - Race condition during Parcel Coalescing Handler creation
- Issue #2516³⁴⁶⁶ - HPX locks up when using at least 256 localities
- Issue #2515³⁴⁶⁷ - error: Install cannot find “/lib/hpx/libparcel_coalescing.so.0.9.99” but I can see that file
- PR #2514³⁴⁶⁸ - Making sure that all continuations of a shared_future are invoked in order
- PR #2513³⁴⁶⁹ - Fixing locks held during suspension
- PR #2512³⁴⁷⁰ - MPI Parcelport improvements and fixes related to the background work changes
- PR #2511³⁴⁷¹ - Fixing bit-wise (zero-copy) serialization
- Issue #2509³⁴⁷² - Linking errors in hwloc_topology
- PR #2508³⁴⁷³ - Added documentation for debugging with core files
- PR #2506³⁴⁷⁴ - Fixing background work invocations
- PR #2505³⁴⁷⁵ - Fix tuple serialization
- Issue #2504³⁴⁷⁶ - Ensure continuations are called in the order they have been attached
- PR #2503³⁴⁷⁷ - Adding serialization support for Vc v2 (datapar)
- PR #2502³⁴⁷⁸ - Resolve various, minor compiler warnings
- PR #2501³⁴⁷⁹ - Some other fixes around cuda examples

³⁴⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2525>

³⁴⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2524>

³⁴⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2523>

³⁴⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2522>

³⁴⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/2521>

³⁴⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2520>

³⁴⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/2519>

³⁴⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2518>

³⁴⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2517>

³⁴⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2516>

³⁴⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2515>

³⁴⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2514>

³⁴⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2513>

³⁴⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2512>

³⁴⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/2511>

³⁴⁷² <https://github.com/STELLAR-GROUP/hpx/issues/2509>

³⁴⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2508>

³⁴⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2506>

³⁴⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2505>

³⁴⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2504>

³⁴⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2503>

³⁴⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2502>

³⁴⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2501>

- Issue #2500³⁴⁸⁰ - nvcc / cuda clang issue due to a missing -DHPX_WITH_CUDA flag
- PR #2499³⁴⁸¹ - Adding support for std::array to wait_all and friends
- PR #2498³⁴⁸² - Execute background work as HPX thread
- PR #2497³⁴⁸³ - Fixing configuration options for spinlock-deadlock detection
- PR #2496³⁴⁸⁴ - Accounting for different compilers in CrayKNL toolchain file
- PR #2494³⁴⁸⁵ - Adding component base class which ties a component instance to a given executor
- PR #2493³⁴⁸⁶ - Enable controlling amount of pending threads which must be available to allow thread stealing
- PR #2492³⁴⁸⁷ - Adding new command line option –hpx:print-counter-reset
- PR #2491³⁴⁸⁸ - Resolve ambiguities when compiling with APEX
- PR #2490³⁴⁸⁹ - Resuming threads waiting on future with higher priority
- Issue #2489³⁴⁹⁰ - nvcc issue because -std=c++11 appears twice
- PR #2488³⁴⁹¹ - Adding performance counters exposing the internal idle and busy-loop counters
- PR #2487³⁴⁹² - Allowing for plain suspend to reschedule thread right away
- PR #2486³⁴⁹³ - Only flag HPX code for CUDA if HPX_WITH_CUDA is set
- PR #2485³⁴⁹⁴ - Making thread-queue parameters runtime-configurable
- PR #2484³⁴⁹⁵ - Added atomic counter for parcel-destinations
- PR #2483³⁴⁹⁶ - Added priority-queue lifo scheduler
- PR #2482³⁴⁹⁷ - Changing scheduler to steal only if more than a minimal number of tasks are available
- PR #2481³⁴⁹⁸ - Extending command line option –hpx:print-counter-destination to support value ‘none’
- PR #2479³⁴⁹⁹ - Added option to disable signal handler
- PR #2478³⁵⁰⁰ - Making sure the sine performance counter module gets loaded only for the corresponding example
- Issue #2477³⁵⁰¹ - Breaking at a throw statement
- PR #2476³⁵⁰² - Annotated function

3480 <https://github.com/STELLAR-GROUP/hpx/issues/2500>

3481 <https://github.com/STELLAR-GROUP/hpx/pull/2499>

3482 <https://github.com/STELLAR-GROUP/hpx/pull/2498>

3483 <https://github.com/STELLAR-GROUP/hpx/pull/2497>

3484 <https://github.com/STELLAR-GROUP/hpx/pull/2496>

3485 <https://github.com/STELLAR-GROUP/hpx/pull/2494>

3486 <https://github.com/STELLAR-GROUP/hpx/pull/2493>

3487 <https://github.com/STELLAR-GROUP/hpx/pull/2492>

3488 <https://github.com/STELLAR-GROUP/hpx/pull/2491>

3489 <https://github.com/STELLAR-GROUP/hpx/pull/2490>

3490 <https://github.com/STELLAR-GROUP/hpx/issues/2489>

3491 <https://github.com/STELLAR-GROUP/hpx/pull/2488>

3492 <https://github.com/STELLAR-GROUP/hpx/pull/2487>

3493 <https://github.com/STELLAR-GROUP/hpx/pull/2486>

3494 <https://github.com/STELLAR-GROUP/hpx/pull/2485>

3495 <https://github.com/STELLAR-GROUP/hpx/pull/2484>

3496 <https://github.com/STELLAR-GROUP/hpx/pull/2483>

3497 <https://github.com/STELLAR-GROUP/hpx/pull/2482>

3498 <https://github.com/STELLAR-GROUP/hpx/pull/2481>

3499 <https://github.com/STELLAR-GROUP/hpx/pull/2479>

3500 <https://github.com/STELLAR-GROUP/hpx/pull/2478>

3501 <https://github.com/STELLAR-GROUP/hpx/issues/2477>

3502 <https://github.com/STELLAR-GROUP/hpx/pull/2476>

- PR #2475³⁵⁰³ - Ensure that using %osthread% during logging will not throw for non-hpx threads
- PR #2474³⁵⁰⁴ - Remove now superficial non_direct actions from base_lco and friends
- PR #2473³⁵⁰⁵ - Refining support for ITTNotify
- PR #2472³⁵⁰⁶ - Some fixes around hpx compute
- Issue #2470³⁵⁰⁷ - redefinition of boost::detail::spinlock
- Issue #2469³⁵⁰⁸ - Dataflow performance issue
- PR #2468³⁵⁰⁹ - Perf docs update
- PR #2466³⁵¹⁰ - Guarantee to execute remote direct actions on HPX-thread
- PR #2465³⁵¹¹ - Improve demo : Async copy and fixed device handling
- PR #2464³⁵¹² - Adding performance counter exposing instantaneous scheduler utilization
- PR #2463³⁵¹³ - Downcast to future<void>
- PR #2462³⁵¹⁴ - Fixed usage of ITT-Notify API with Intel Amplifier
- PR #2461³⁵¹⁵ - Cublas demo
- PR #2460³⁵¹⁶ - Fixing thread bindings
- PR #2459³⁵¹⁷ - Make -std=c++11 nvcc flag consistent for in-build and installed versions
- Issue #2457³⁵¹⁸ - Segmentation fault when registering a partitioned vector
- PR #2452³⁵¹⁹ - Properly releasing global barrier for unhandled exceptions
- PR #2451³⁵²⁰ - Fixing long shutdown times
- PR #2450³⁵²¹ - Attempting to fix initialization errors on newer platforms (Boost V1.63)
- PR #2449³⁵²² - Replace BOOST_COMPILER_FENCE with an HPX version
- PR #2448³⁵²³ - This fixes a possible race in the migration code
- PR #2445^{Page 1092, 3524} - **Fixing dataflow et.al. for futures or future-ranges wrapped into ref()**
- PR #2444³⁵²⁵ - Fix segfaults

³⁵⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2475>

³⁵⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2474>

³⁵⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2473>

³⁵⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2472>

³⁵⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2470>

³⁵⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2469>

³⁵⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2468>

³⁵¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2466>

³⁵¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2465>

³⁵¹² <https://github.com/STELLAR-GROUP/hpx/pull/2464>

³⁵¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2463>

³⁵¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2462>

³⁵¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2461>

³⁵¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2460>

³⁵¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2459>

³⁵¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2457>

³⁵¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2452>

³⁵²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2451>

³⁵²¹ <https://github.com/STELLAR-GROUP/hpx/pull/2450>

³⁵²² <https://github.com/STELLAR-GROUP/hpx/pull/2449>

³⁵²³ <https://github.com/STELLAR-GROUP/hpx/pull/2448>

³⁵²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2445>

³⁵²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2444>

- PR #2443³⁵²⁶ - Issue 2442
- Issue #2442³⁵²⁷ - Mismatch between #if/#endif and namespace scope brackets in this_thread_executors.hpp
- Issue #2441³⁵²⁸ - undeclared identifier BOOST_COMPILER_FENCE
- PR #2440³⁵²⁹ - Knl build
- PR #2438³⁵³⁰ - Datapar backend
- PR #2437³⁵³¹ - Adapt algorithm parameter sequence changes from C++17
- PR #2436³⁵³² - Adapt execution policy name changes from C++17
- Issue #2435³⁵³³ - Trunk broken, undefined reference to hpx::thread::interrupt(hpx::thread::id, bool)
- PR #2434³⁵³⁴ - More fixes to resource manager
- PR #2433³⁵³⁵ - Added versions of hpx::get_ptr taking client side representations
- PR #2432³⁵³⁶ - Warning fixes
- PR #2431³⁵³⁷ - Adding facility representing set of performance counters
- PR #2430³⁵³⁸ - Fix parallel_executor thread spawning
- PR #2429³⁵³⁹ - Fix attribute warning for gcc
- Issue #2427³⁵⁴⁰ - Seg fault running octo-tiger with latest HPX commit
- Issue #2426³⁵⁴¹ - Bug in 9592f5c0bc29806fce0dbe73f35b6ca7e027edcb causes immediate crash in Octo-tiger
- PR #2425³⁵⁴² - Fix nvcc errors due to constexpr specifier
- Issue #2424³⁵⁴³ - Async action on component present on hpx::find_here is executing synchronously
- PR #2423³⁵⁴⁴ - Fix nvcc errors due to constexpr specifier
- PR #2422³⁵⁴⁵ - Implementing hpx::this_thread thread data functions
- PR #2421³⁵⁴⁶ - Adding benchmark for wait_all
- Issue #2420³⁵⁴⁷ - Returning object of a component client from another component action fails
- PR #2419³⁵⁴⁸ - Infiniband parcelport

³⁵²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2443>

³⁵²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2442>

³⁵²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2441>

³⁵²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2440>

³⁵³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2438>

³⁵³¹ <https://github.com/STELLAR-GROUP/hpx/pull/2437>

³⁵³² <https://github.com/STELLAR-GROUP/hpx/pull/2436>

³⁵³³ <https://github.com/STELLAR-GROUP/hpx/issues/2435>

³⁵³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2434>

³⁵³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2433>

³⁵³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2432>

³⁵³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2431>

³⁵³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2430>

³⁵³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2429>

³⁵⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2427>

³⁵⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/2426>

³⁵⁴² <https://github.com/STELLAR-GROUP/hpx/pull/2425>

³⁵⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/2424>

³⁵⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2423>

³⁵⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2422>

³⁵⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2421>

³⁵⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2420>

³⁵⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2419>

- Issue #2418³⁵⁴⁹ - gcc + nvcc fails to compile code that uses partitioned_vector
- PR #2417³⁵⁵⁰ - Fixing context switching
- PR #2416³⁵⁵¹ - Adding fixes and workarounds to allow compilation with nvcc/msvc (VS2015up3)
- PR #2415³⁵⁵² - Fix errors coming from hpx compute examples
- PR #2414³⁵⁵³ - Fixing msvc12
- PR #2413³⁵⁵⁴ - Enable cuda/nvcc or cuda/clang when using add_hpx_executable()
- PR #2412³⁵⁵⁵ - Fix issue in HPX_SetupTarget.cmake when cuda is used
- PR #2411³⁵⁵⁶ - This fixes the core compilation issues with MSVC12
- Issue #2410³⁵⁵⁷ - undefined reference to opal_hwloc191_hwloc_.....
- PR #2409³⁵⁵⁸ - Fixing locking for channel and receive_buffer
- PR #2407³⁵⁵⁹ - Solving #2402 and #2403
- PR #2406³⁵⁶⁰ - Improve guards
- PR #2405³⁵⁶¹ - Enable parallel::for_each for iterators returning proxy types
- PR #2404³⁵⁶² - Forward the explicitly given result_type in the hpx invoke
- Issue #2403³⁵⁶³ - datapar_execution + zip iterator: lambda arguments aren't references
- Issue #2402³⁵⁶⁴ - datapar algorithm instantiated with wrong type #2402
- PR #2401³⁵⁶⁵ - Added support for imported libraries to HPX_Libraries.cmake
- PR #2400³⁵⁶⁶ - Use CMake policy CMP0060
- Issue #2399³⁵⁶⁷ - Error trying to push back vector of futures to vector
- PR #2398³⁵⁶⁸ - Allow config #defines to be written out to custom config/defines.hpp
- Issue #2397³⁵⁶⁹ - CMake generated config defines can cause tedious rebuilds category
- Issue #2396³⁵⁷⁰ - BOOST_ROOT paths are not used at link time
- PR #2395³⁵⁷¹ - Fix target_link_libraries() issue when HPX Cuda is enabled

³⁵⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2418>

³⁵⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2417>

³⁵⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/2416>

³⁵⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2415>

³⁵⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/2414>

³⁵⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2413>

³⁵⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2412>

³⁵⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2411>

³⁵⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2410>

³⁵⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2409>

³⁵⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2407>

³⁵⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2406>

³⁵⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/2405>

³⁵⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2404>

³⁵⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/2403>

³⁵⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2402>

³⁵⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2401>

³⁵⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2400>

³⁵⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2399>

³⁵⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2398>

³⁵⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2397>

³⁵⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2396>

³⁵⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/2395>

- Issue #2394³⁵⁷² - Template compilation error using HPX_WITH_DATAPAR_LIBFLATARRAY
- PR #2393³⁵⁷³ - Fixing lock registration for recursive mutex
- PR #2392³⁵⁷⁴ - Add keywords in target_link_libraries in hpx_setup_target
- PR #2391³⁵⁷⁵ - Clang goroutines
- Issue #2390³⁵⁷⁶ - Adapt execution policy name changes from C++17
- PR #2389³⁵⁷⁷ - Chunk allocator and pool are not used and are obsolete
- PR #2388³⁵⁷⁸ - Adding functionalities to datapar needed by octotiger
- PR #2387³⁵⁷⁹ - Fixing race condition for early parcels
- Issue #2386³⁵⁸⁰ - Lock registration broken for recursive_mutex
- PR #2385³⁵⁸¹ - Datapar zip iterator
- PR #2384³⁵⁸² - Fixing race condition in for_loop_reduction
- PR #2383³⁵⁸³ - Continuations
- PR #2382³⁵⁸⁴ - add LibFlatArray-based backend for datapar
- PR #2381³⁵⁸⁵ - remove unused typedef to get rid of compiler warnings
- PR #2380³⁵⁸⁶ - Tau cleanup
- PR #2379³⁵⁸⁷ - Can send immediate
- PR #2378³⁵⁸⁸ - Renaming copy_helper/copy_n_helper/move_helper/move_n_helper
- Issue #2376³⁵⁸⁹ - Boost trunk's spinlock initializer fails to compile
- PR #2375³⁵⁹⁰ - Add support for minimal thread local data
- PR #2374³⁵⁹¹ - Adding API functions set_config_entry_callback
- PR #2373³⁵⁹² - Add a simple utility for debugging that gives suspended task backtraces
- PR #2372³⁵⁹³ - Barrier Fixes
- Issue #2370³⁵⁹⁴ - Can't wait on a wrapped future

³⁵⁷² <https://github.com/STELLAR-GROUP/hpx/issues/2394>

³⁵⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2393>

³⁵⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2392>

³⁵⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2391>

³⁵⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2390>

³⁵⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2389>

³⁵⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2388>

³⁵⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2387>

³⁵⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2386>

³⁵⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2385>

³⁵⁸² <https://github.com/STELLAR-GROUP/hpx/pull/2384>

³⁵⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2383>

³⁵⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2382>

³⁵⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2381>

³⁵⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2380>

³⁵⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2379>

³⁵⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2378>

³⁵⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2376>

³⁵⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2375>

³⁵⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2374>

³⁵⁹² <https://github.com/STELLAR-GROUP/hpx/pull/2373>

³⁵⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2372>

³⁵⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2370>

- PR #2369³⁵⁹⁵ - Fixing stable_partition
- PR #2367³⁵⁹⁶ - Fixing find_prefixes for Windows platforms
- PR #2366³⁵⁹⁷ - Testing for experimental/optional only in C++14 mode
- PR #2364³⁵⁹⁸ - Adding set_config_entry
- PR #2363³⁵⁹⁹ - Fix papi
- PR #2362³⁶⁰⁰ - Adding missing macros for new non-direct actions
- PR #2361³⁶⁰¹ - Improve cmake output to help debug compiler incompatibility check
- PR #2360³⁶⁰² - Fixing race condition in condition_variable
- PR #2359³⁶⁰³ - Fixing shutdown when parcels are still in flight
- Issue #2357³⁶⁰⁴ - failed to insert console_print_action into typename_to_id_t registry
- PR #2356³⁶⁰⁵ - Fixing return type of get_iterator_tuple
- PR #2355³⁶⁰⁶ - Fixing compilation against Boost 1.62
- PR #2354³⁶⁰⁷ - Adding serialization for mask_type if CPU_COUNT > 64
- PR #2353³⁶⁰⁸ - Adding hooks to tie in APEX into the parcel layer
- Issue #2352³⁶⁰⁹ - Compile errors when using intel 17 beta (for KNL) on edison
- PR #2351³⁶¹⁰ - Fix function vtable get_function_address implementation
- Issue #2350³⁶¹¹ - Build failure - master branch (4de09f5) with Intel Compiler v17
- PR #2349³⁶¹² - Enabling zero-copy serialization support for std::vector<>
- PR #2348³⁶¹³ - Adding test to verify #2334 is fixed
- PR #2347³⁶¹⁴ - Bug fixes for hpx.compute and hpx::lcos::channel
- PR #2346³⁶¹⁵ - Removing cmake “find” files that are in the APEX cmake Modules
- PR #2345³⁶¹⁶ - Implemented parallel::stable_partition
- PR #2344³⁶¹⁷ - Making hpx::lcos::channel usable with basename registration

³⁵⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2369>

³⁵⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2367>

³⁵⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2366>

³⁵⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2364>

³⁵⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2363>

³⁶⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2362>

³⁶⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2361>

³⁶⁰² <https://github.com/STELLAR-GROUP/hpx/pull/2360>

³⁶⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2359>

³⁶⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2357>

3605 <https://github.com/STELLAR-GROUP/hpx/pull/2356>3606 <https://github.com/STELLAR-GROUP/hpx/pull/2355>3607 <https://github.com/STELLAR-GROUP/hpx/pull/2354>3608 <https://github.com/STELLAR-GROUP/hpx/pull/2353>3609 <https://github.com/STELLAR-GROUP/hpx/issues/2352>3610 <https://github.com/STELLAR-GROUP/hpx/pull/2351>3611 <https://github.com/STELLAR-GROUP/hpx/issues/2350>3612 <https://github.com/STELLAR-GROUP/hpx/pull/2349>3613 <https://github.com/STELLAR-GROUP/hpx/pull/2348>3614 <https://github.com/STELLAR-GROUP/hpx/pull/2347>3615 <https://github.com/STELLAR-GROUP/hpx/pull/2346>3616 <https://github.com/STELLAR-GROUP/hpx/pull/2345>3617 <https://github.com/STELLAR-GROUP/hpx/pull/2344>

- PR #2343³⁶¹⁸ - Fix a couple of examples that failed to compile after recent api changes
- Issue #2342³⁶¹⁹ - Enabling APEX causes link errors
- PR #2341³⁶²⁰ - Removing cmake “find” files that are in the APEX cmake Modules
- PR #2340³⁶²¹ - Implemented all existing datapar algorithms using Boost.SIMD
- PR #2339³⁶²² - Fixing 2338
- PR #2338³⁶²³ - Possible race in sliding semaphore
- PR #2337³⁶²⁴ - Adjust osu_latency test to measure window_size parcels in flight at once
- PR #2336³⁶²⁵ - Allowing remote direct actions to be executed without spawning a task
- PR #2335³⁶²⁶ - Making sure multiple components are properly initialized from arguments
- Issue #2334³⁶²⁷ - Cannot construct component with large vector on a remote locality
- PR #2332³⁶²⁸ - Fixing hpx::lcos::local::barrier
- PR #2331³⁶²⁹ - Updating APEX support to include OTF2
- PR #2330³⁶³⁰ - Support for data-parallelism for parallel algorithms
- Issue #2329³⁶³¹ - Coordinate settings in cmake
- PR #2328³⁶³² - fix LibGeoDecomp builds with HPX + GCC 5.3.0 + CUDA 8RC
- PR #2326³⁶³³ - Making scan_partitioner work (for now)
- Issue #2323³⁶³⁴ - Constructing a vector of components only correctly initializes the first component
- PR #2322³⁶³⁵ - Fix problems that bubbled up after merging #2278
- PR #2321³⁶³⁶ - Scalable barrier
- PR #2320³⁶³⁷ - Std flag fixes
- Issue #2319³⁶³⁸ - -std=c++14 and -std=c++1y with Intel can't build recent Boost builds due to insufficient C++14 support; don't enable these flags by default for Intel
- PR #2318³⁶³⁹ - Improve handling of -hpx:bind=<bind-spec>
- PR #2317³⁶⁴⁰ - Making sure command line warnings are printed once only

3618 <https://github.com/STELLAR-GROUP/hpx/pull/2343>

3619 <https://github.com/STELLAR-GROUP/hpx/issues/2342>

3620 <https://github.com/STELLAR-GROUP/hpx/pull/2341>

3621 <https://github.com/STELLAR-GROUP/hpx/pull/2340>

3622 <https://github.com/STELLAR-GROUP/hpx/pull/2339>

3623 <https://github.com/STELLAR-GROUP/hpx/pull/2338>

3624 <https://github.com/STELLAR-GROUP/hpx/pull/2337>

3625 <https://github.com/STELLAR-GROUP/hpx/pull/2336>

3626 <https://github.com/STELLAR-GROUP/hpx/pull/2335>

3627 <https://github.com/STELLAR-GROUP/hpx/issues/2334>

3628 <https://github.com/STELLAR-GROUP/hpx/pull/2332>

3629 <https://github.com/STELLAR-GROUP/hpx/pull/2331>

3630 <https://github.com/STELLAR-GROUP/hpx/pull/2330>

3631 <https://github.com/STELLAR-GROUP/hpx/issues/2329>

3632 <https://github.com/STELLAR-GROUP/hpx/pull/2328>

3633 <https://github.com/STELLAR-GROUP/hpx/pull/2326>

3634 <https://github.com/STELLAR-GROUP/hpx/issues/2323>

3635 <https://github.com/STELLAR-GROUP/hpx/pull/2322>

3636 <https://github.com/STELLAR-GROUP/hpx/pull/2321>

3637 <https://github.com/STELLAR-GROUP/hpx/pull/2320>

3638 <https://github.com/STELLAR-GROUP/hpx/issues/2319>

3639 <https://github.com/STELLAR-GROUP/hpx/pull/2318>

3640 <https://github.com/STELLAR-GROUP/hpx/pull/2317>

- PR #2316³⁶⁴¹ - Fixing command line handling for default bind mode
- PR #2315³⁶⁴² - Set id_retrieved if set_id is present
- Issue #2314³⁶⁴³ - Warning for requested/allocated thread discrepancy is printed twice
- Issue #2313³⁶⁴⁴ - –hpx:print-bind doesn't work with –hpx:pu-step
- Issue #2312³⁶⁴⁵ - –hpx:bind range specifier restrictions are overly restrictive
- Issue #2311³⁶⁴⁶ - hpx_0.9.99 out of project build fails
- PR #2310³⁶⁴⁷ - Simplify function registration
- PR #2309³⁶⁴⁸ - Spelling and grammar revisions in documentation (and some code)
- PR #2306³⁶⁴⁹ - Correct minor typo in the documentation
- PR #2305³⁶⁵⁰ - Cleaning up and fixing parcel coalescing
- PR #2304³⁶⁵¹ - Inspect checks for stream related includes
- PR #2303³⁶⁵² - Add functionality allowing to enumerate threads of given state
- PR #2301³⁶⁵³ - Algorithm overloads fix for VS2013
- PR #2300³⁶⁵⁴ - Use <cstdint>, add inspect checks
- PR #2299³⁶⁵⁵ - Replace boost::[c]ref with std::[c]ref, add inspect checks
- PR #2297³⁶⁵⁶ - Fixing compilation with no hw_loc
- PR #2296³⁶⁵⁷ - HpX compute
- PR #2295³⁶⁵⁸ - Making sure for_loop(execution::par, 0, N, ...) is actually executed in parallel
- PR #2294³⁶⁵⁹ - Throwing exceptions if the runtime is not up and running
- PR #2293³⁶⁶⁰ - Removing unused parcel port code
- PR #2292³⁶⁶¹ - Refactor function vtables
- PR #2291³⁶⁶² - Fixing 2286
- PR #2290³⁶⁶³ - Simplify algorithm overloads

³⁶⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/2316>

³⁶⁴² <https://github.com/STELLAR-GROUP/hpx/pull/2315>

³⁶⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/2314>

³⁶⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2313>

³⁶⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2312>

³⁶⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2311>

³⁶⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2310>

³⁶⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2309>

³⁶⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2306>

³⁶⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2305>

³⁶⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/2304>

³⁶⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2303>

³⁶⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/2301>

³⁶⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2300>

³⁶⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2299>

³⁶⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2297>

³⁶⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2296>

³⁶⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2295>

³⁶⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2294>

³⁶⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2293>

³⁶⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/2292>

³⁶⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2291>

³⁶⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/2290>

- PR #2289³⁶⁶⁴ - Adding performance counters reporting parcel related data on a per-action basis
- Issue #2288³⁶⁶⁵ - Remove dormant parcelports
- Issue #2286³⁶⁶⁶ - adjustments to parcel handling to support parcelports that do not need a connection cache
- PR #2285³⁶⁶⁷ - add CMake option to disable package export
- PR #2283³⁶⁶⁸ - Add more inspect checks for use of deprecated components
- Issue #2282³⁶⁶⁹ - Arithmetic exception in executor static chunker
- Issue #2281³⁶⁷⁰ - For loop doesn't parallelize
- PR #2280³⁶⁷¹ - Fixing 2277: build failure with PAPI
- PR #2279³⁶⁷² - Child vs parent stealing
- Issue #2277³⁶⁷³ - master branch build failure (53c5b4f) with papi
- PR #2276³⁶⁷⁴ - Compile time launch policies
- PR #2275³⁶⁷⁵ - Replace boost::chrono with std::chrono in interfaces
- PR #2274³⁶⁷⁶ - Replace most uses of Boost.Assign with initializer list
- PR #2273³⁶⁷⁷ - Fixed typos
- PR #2272³⁶⁷⁸ - Inspect checks
- PR #2270³⁶⁷⁹ - Adding test verifying -Ihpx.os_threads=all
- PR #2269³⁶⁸⁰ - Added inspect check for now obsolete boost type traits
- PR #2268³⁶⁸¹ - Moving more code into source files
- Issue #2267³⁶⁸² - Add inspect support to deprecate Boost.TypeTraits
- PR #2265³⁶⁸³ - Adding channel LCO
- PR #2264³⁶⁸⁴ - Make support for std::ref mandatory
- PR #2263³⁶⁸⁵ - Constrain tuple_member forwarding constructor
- Issue #2262³⁶⁸⁶ - Test hpx.os_threads=all

³⁶⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2289>

³⁶⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2288>

³⁶⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2286>

³⁶⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2285>

³⁶⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2283>

³⁶⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2282>

³⁶⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2281>

³⁶⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/2280>

³⁶⁷² <https://github.com/STELLAR-GROUP/hpx/pull/2279>

³⁶⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/2277>

³⁶⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2276>

³⁶⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2275>

³⁶⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2274>

³⁶⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2273>

³⁶⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2272>

³⁶⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2270>

³⁶⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2269>

³⁶⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2268>

³⁶⁸² <https://github.com/STELLAR-GROUP/hpx/issues/2267>

³⁶⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2265>

³⁶⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2264>

³⁶⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2263>

³⁶⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2262>

- Issue #2261³⁶⁸⁷ - OS X: Error: no matching constructor for initialization of ‘hpx::lcos::local::condition_variable_any’
- Issue #2260³⁶⁸⁸ - Make support for std::ref mandatory
- PR #2259³⁶⁸⁹ - Remove most of Boost.MPL, Boost.EnableIf and Boost.TypeTraits
- PR #2258³⁶⁹⁰ - Fixing #2256
- PR #2257³⁶⁹¹ - Fixing launch process
- Issue #2256³⁶⁹² - Actions are not registered if not invoked
- PR #2255³⁶⁹³ - Coalescing histogram
- PR #2254³⁶⁹⁴ - Silence explicit initialization in copy-constructor warnings
- PR #2253³⁶⁹⁵ - Drop support for GCC 4.6 and 4.7
- PR #2252³⁶⁹⁶ - Prepare V1.0
- PR #2251³⁶⁹⁷ - Convert to 0.9.99
- PR #2249³⁶⁹⁸ - Adding iterator_facade and iterator_adaptor
- Issue #2248³⁶⁹⁹ - Need a feature to yield to a new task immediately
- PR #2246³⁷⁰⁰ - Adding split_future
- PR #2245³⁷⁰¹ - Add an example for handing over a component instance to a dynamically launched locality
- Issue #2243³⁷⁰² - Add example demonstrating AGAS symbolic name registration
- Issue #2242³⁷⁰³ - pkgconfig test broken on CentOS 7 / Boost 1.61
- Issue #2241³⁷⁰⁴ - Compilation error for partitioned vector in hpx_compute branch
- PR #2240³⁷⁰⁵ - Fixing termination detection on one locality
- Issue #2239³⁷⁰⁶ - Create a new facility lcos::split_all
- Issue #2236³⁷⁰⁷ - hpx::cout vs. std::cout
- PR #2232³⁷⁰⁸ - Implement local-only primary namespace service
- Issue #2147³⁷⁰⁹ - would like to know how much data is being routed by particular actions

³⁶⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2261>

³⁶⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2260>

³⁶⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2259>

³⁶⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2258>

³⁶⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2257>

³⁶⁹² <https://github.com/STELLAR-GROUP/hpx/issues/2256>

³⁶⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2255>

³⁶⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2254>

³⁶⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2253>

³⁶⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2252>

³⁶⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2251>

³⁶⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2249>

³⁶⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2248>

³⁷⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2246>

³⁷⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2245>

³⁷⁰² <https://github.com/STELLAR-GROUP/hpx/issues/2243>

³⁷⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/2242>

³⁷⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2241>

³⁷⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2240>

³⁷⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2239>

³⁷⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2236>

³⁷⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2232>

³⁷⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2147>

- Issue #2109³⁷¹⁰ - Warning while compiling hpx
- Issue #1973³⁷¹¹ - Setting INTERFACE_COMPILE_OPTIONS for hpx_init in CMake taints Fortran_FLAGS
- Issue #1864³⁷¹² - run_guarded using bound function ignores reference
- Issue #1754³⁷¹³ - Running with TCP parcelport causes immediate crash or freeze
- Issue #1655³⁷¹⁴ - Enable zip_iterator to be used with Boost traversal iterator categories
- Issue #1591³⁷¹⁵ - Optimize AGAS for shared memory only operation
- Issue #1401³⁷¹⁶ - Need an efficient infiniband parcelport
- Issue #1125³⁷¹⁷ - Fix the IPC parcelport
- Issue #839³⁷¹⁸ - Refactor ibverbs and shmem parcelport
- Issue #702³⁷¹⁹ - Add instrumentation of parcel layer
- Issue #668³⁷²⁰ - Implement ispc task interface
- Issue #533³⁷²¹ - Thread queue/deque internal parameters should be runtime configurable
- Issue #475³⁷²² - Create a means of combining performance counters into querysets

2.10.14 HPX V0.9.99 (Jul 15, 2016)

General changes

As the version number of this release hints, we consider this release to be a preview for the upcoming *HPX* V1.0. All of the functionalities we set out to implement for V1.0 are in place; all of the features we wanted to have exposed are ready. We are very happy with the stability and performance of *HPX* and we would like to present this release to the community in order for us to gather broad feedback before releasing V1.0. We still expect for some minor details to change, but on the whole this release represents what we would like to have in a V1.0.

Overall, since the last release we have had almost 1600 commits while closing almost 400 tickets. These numbers reflect the incredible development activity we have seen over the last couple of months. We would like to express a big ‘Thank you!’ to all contributors and those who helped to make this release happen.

The most notable addition in terms of new functionality available with this release is the full implementation of object migration (i.e. the ability to transparently move *HPX* components to a different compute node). Additionally, this release of *HPX* cleans up many minor issues and some API inconsistencies.

Here are some of the main highlights and changes for this release (in no particular order):

- We have fixed a couple of issues in AGAS and the parcel layer which have caused hangs, segmentation faults at exit, and a slowdown of applications over time. Fixing those has significantly increased the overall stability and performance of distributed runs.

³⁷¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2109>

³⁷¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1973>

³⁷¹² <https://github.com/STELLAR-GROUP/hpx/issues/1864>

³⁷¹³ <https://github.com/STELLAR-GROUP/hpx/issues/1754>

³⁷¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1655>

³⁷¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1591>

³⁷¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1401>

³⁷¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1125>

³⁷¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/839>

³⁷¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/702>

³⁷²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/668>

³⁷²¹ <https://github.com/STELLAR-GROUP/hpx/issues/533>

³⁷²² <https://github.com/STELLAR-GROUP/hpx/issues/475>

- We have started to add parallel algorithm overloads based on the C++ Extensions for Ranges ([N4560³⁷²³](#)) proposal. This also includes the addition of projections to the existing algorithms. Please see [Issue #1668³⁷²⁴](#) for a list of algorithms which have been adapted to [N4560³⁷²⁵](#).
- We have implemented index-based parallel for-loops based on a corresponding standardization proposal ([P0075R1³⁷²⁶](#)). Please see [Issue #2016³⁷²⁷](#) for a list of available algorithms.
- We have added implementations for more parallel algorithms as proposed for the upcoming C++ 17 Standard. See [Issue #1141³⁷²⁸](#) for an overview of which algorithms are available by now.
- We have started to implement a new prototypical functionality with *HPX.Compute* which uniformly exposes some of the higher level APIs to heterogeneous architectures (currently CUDA). This functionality is an early preview and should not be considered stable. It may change considerably in the future.
- We have pervasively added (optional) executor arguments to all API functions which schedule new work. Executors are now used throughout the code base as the main means of executing tasks.
- Added `hpx::make_future<R>(future<T> &&)` allowing to convert a future of any type T into a future of any other type R, either based on default conversion rules of the embedded types or using a given explicit conversion function.
- We finally finished the implementation of transparent migration of components to another locality. It is now possible to trigger a migration operation without ‘stopping the world’ for the object to migrate. *HPX* will make sure that no work is being performed on an object before it is migrated and that all subsequently scheduled work for the migrated object will be transparently forwarded to the new locality. Please note that the global id of the migrated object does not change, thus the application will not have to be changed in any way to support this new functionality. Please note that this feature is currently considered experimental. See [Issue #559³⁷²⁹](#) and [PR #1966³⁷³⁰](#) for more details.
- The `hpx::dataflow` facility is now usable with actions. Similarly to `hpx::async`, actions can be specified as an explicit template argument (`hpx::dataflow<Action>(target, ...)`) or as the first argument (`hpx::dataflow(Action(), target, ...)`). We have also enabled the use of distribution policies as the target for dataflow invocations. Please see [Issue #1265³⁷³¹](#) and [PR #1912³⁷³²](#) for more information.
- Adding overloads of `gather_here` and `gather_there` to accept the plain values of the data to gather (in addition to the existing overloads expecting futures).
- We have cleaned up and refactored large parts of the code base. This helped reducing compile and link times of *HPX* itself and also of applications depending on it. We have further decreased the dependency of *HPX* on the Boost libraries by replacing part of those with facilities available from the standard libraries.
- Wherever possible we have removed dependencies of our API on Boost by replacing those with the equivalent facility from the C++11 standard library.
- We have added new performance counters for parcel coalescing, file-IO, the AGAS cache, and overall scheduler time. Resetting performance counters has been overhauled and fixed.
- We have introduced a generic client type `hpx::components::client<>` and added support for using it with `hpx::async`. This removes the necessity to implement specific client types for every component type without losing type safety. This deemphasizes the need for using the low level `hpx::id_type` for referencing

³⁷²³ <http://wg21.link/n4560>³⁷²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1668>³⁷²⁵ <http://wg21.link/n4560>³⁷²⁶ <http://wg21.link/p0075r1>³⁷²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2016>³⁷²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1141>³⁷²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/559>³⁷³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1966>³⁷³¹ <https://github.com/STELLAR-GROUP/hpx/issues/1265>³⁷³² <https://github.com/STELLAR-GROUP/hpx/pull/1912>

(possibly remote) component instances. The plan is to deprecate the direct use of `hpx::id_type` in user code in the future.

- We have added a special iterator which supports automatic prefetching of one or more arrays for speeding up loop-like code (see `hpx::parallel::util::make_prefetcher_context()`).
- We have extended the interfaces exposed from executors (as proposed by [N4406³⁷³³](#)) to accept an arbitrary number of arguments.

Breaking changes

- In order to move the dataflow facility to namespace `hpx` we added a definition of `hpx::dataflow` which might create ambiguities in existing codes. The previous definition of this facility (`hpx::lcos::local::dataflow`) has been deprecated and is available only if the constant `-DHPX_WITH_LOCAL_DATAFLOW_COMPATIBILITY=On` to [CMake³⁷³⁴](#) is defined at configuration time. Please explicitly qualify all uses of the dataflow facility if you enable this compatibility setting and encounter ambiguities.
- The adaptation of the C++ Extensions for Ranges ([N4560³⁷³⁵](#)) proposal imposes some breaking changes related to the return types of some of the parallel algorithms. Please see [Issue #1668³⁷³⁶](#) for a list of algorithms which have already been adapted.
- The facility `hpx::lcos::make_future_void()` has been replaced by `hpx::make_future<void>()`.
- We have removed support for Intel V13 and gcc 4.4.x.
- We have removed (default) support for the generic `hpx::parallel::execution_policy` because it was removed from the Parallelism TS ([_cpp11_n4104_](#)) while it was being added to the upcoming C++17 Standard. This facility can be still enabled at configure time by specifying `-DHPX_WITH_GENERIC_EXECUTION_POLICY=On` to CMake.
- Uses of `boost::shared_ptr` and related facilities have been replaced with `std::shared_ptr` and friends. Uses of `boost::unique_lock`, `boost::lock_guard` etc. have also been replaced by the equivalent (and equally named) tools available from the C++11 standard library.
- Facilities that used to expect an explicit `boost::unique_lock` now take an `std::unique_lock`. Additionally, `condition_variable` no longer aliases `condition_variable_any`; its interface now only works with `std::unique_lock<local::mutex>`.
- Uses of `boost::function`, `boost::bind`, `boost::tuple` have been replaced by the corresponding facilities in *HPX* (`hpx::util::function`, `hpx::util::bind`, and `hpx::util::tuple`, respectively).

³⁷³³ <http://wg21.link/n4406>

³⁷³⁴ <https://www.cmake.org>

³⁷³⁵ <http://wg21.link/n4560>

³⁷³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1668>

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- PR #2250³⁷³⁷ - change default chunker of parallel executor to static one
- PR #2247³⁷³⁸ - HPX on ppc64le
- PR #2244³⁷³⁹ - Fixing MSVC problems
- PR #2238³⁷⁴⁰ - Fixing small typos
- PR #2237³⁷⁴¹ - Fixing small typos
- PR #2234³⁷⁴² - Fix broken add test macro when extra args are passed in
- PR #2231³⁷⁴³ - Fixing possible race during future awaiting in serialization
- PR #2230³⁷⁴⁴ - Fix stream nvcc
- PR #2229³⁷⁴⁵ - Fixed run_as_hpx_thread
- PR #2228³⁷⁴⁶ - On prefetching_test branch : adding prefetching_iterator and related tests used for prefetching containers within lambda functions
- PR #2227³⁷⁴⁷ - Support for HPXCL's opencl::event
- PR #2226³⁷⁴⁸ - Preparing for release of V0.9.99
- PR #2225³⁷⁴⁹ - fix issue when compiling components with hpxcxx
- PR #2224³⁷⁵⁰ - Compute alloc fix
- PR #2223³⁷⁵¹ - Simplify promise
- PR #2222³⁷⁵² - Replace last uses of boost::function by util::function_nonser
- PR #2221³⁷⁵³ - Fix config tests
- PR #2220³⁷⁵⁴ - Fixing gcc 4.6 compilation issues
- PR #2219³⁷⁵⁵ - nullptr support for [unique_]function
- PR #2218³⁷⁵⁶ - Introducing clang tidy
- PR #2216³⁷⁵⁷ - Replace NULL with nullptr

³⁷³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2250>

³⁷³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2247>

³⁷³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2244>

³⁷⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2238>

³⁷⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/2237>

³⁷⁴² <https://github.com/STELLAR-GROUP/hpx/pull/2234>

³⁷⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/2231>

³⁷⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2230>

³⁷⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2229>

³⁷⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2228>

³⁷⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2227>

³⁷⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2226>

³⁷⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2225>

³⁷⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2224>

³⁷⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/2223>

³⁷⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2222>

³⁷⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/2221>

³⁷⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2220>

³⁷⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2219>

³⁷⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2218>

³⁷⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2216>

- Issue #2214³⁷⁵⁸ - Let inspect flag use of NULL, suggest nullptr instead
- PR #2213³⁷⁵⁹ - Require support for nullptr
- PR #2212³⁷⁶⁰ - Properly find jemalloc through pkg-config
- PR #2211³⁷⁶¹ - Disable a couple of warnings reported by Intel on Windows
- PR #2210³⁷⁶² - Fixed host::block_allocator::bulk_construct
- PR #2209³⁷⁶³ - Started to clean up new sort algorithms, made things compile for sort_by_key
- PR #2208³⁷⁶⁴ - A couple of fixes that were exposed by a new sort algorithm
- PR #2207³⁷⁶⁵ - Adding missing includes in /hpx/include/serialization.hpp
- PR #2206³⁷⁶⁶ - Call package_action::get_future before package_action::apply
- PR #2205³⁷⁶⁷ - The indirect_packaged_task::operator() needs to be run on a HPX thread
- PR #2204³⁷⁶⁸ - Variadic executor parameters
- PR #2203³⁷⁶⁹ - Delay-initialize members of partitioned iterator
- PR #2202³⁷⁷⁰ - Added segmented fill for hpx::vector
- Issue #2201³⁷⁷¹ - Null Thread id encountered on partitioned_vector
- PR #2200³⁷⁷² - Fix hangs
- PR #2199³⁷⁷³ - Deprecating hpx/traits.hpp
- PR #2198³⁷⁷⁴ - Making explicit inclusion of external libraries into build
- PR #2197³⁷⁷⁵ - Fix typo in QT CMakeLists
- PR #2196³⁷⁷⁶ - Fixing a gcc warning about attributes being ignored
- PR #2194³⁷⁷⁷ - Fixing partitioned_vector_spmd_FOREACH example
- Issue #2193³⁷⁷⁸ - partitioned_vector_spmd_FOREACH seg faults
- PR #2192³⁷⁷⁹ - Support Boost.Thread v4
- PR #2191³⁷⁸⁰ - HPX.Compute prototype

³⁷⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2214>

³⁷⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2213>

³⁷⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2212>

³⁷⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/2211>

³⁷⁶² <https://github.com/STELLAR-GROUP/hpx/pull/2210>

³⁷⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/2209>

³⁷⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2208>

³⁷⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2207>

³⁷⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2206>

³⁷⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2205>

³⁷⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2204>

³⁷⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2203>

³⁷⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2202>

³⁷⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/2201>

³⁷⁷² <https://github.com/STELLAR-GROUP/hpx/pull/2200>

³⁷⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2199>

³⁷⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2198>

³⁷⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2197>

³⁷⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2196>

³⁷⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2194>

³⁷⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2193>

³⁷⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2192>

³⁷⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2191>

- PR #2190³⁷⁸¹ - Spawning operation on new thread if remaining stack space becomes too small
- PR #2189³⁷⁸² - Adding callback taking index and future to when_each
- PR #2188³⁷⁸³ - Adding new example demonstrating receive_buffer
- PR #2187³⁷⁸⁴ - Mask 128-bit ints if CUDA is being used
- PR #2186³⁷⁸⁵ - Make startup & shutdown functions unique_function
- PR #2185³⁷⁸⁶ - Fixing logging output not to cause hang on shutdown
- PR #2184³⁷⁸⁷ - Allowing component clients as action return types
- Issue #2183³⁷⁸⁸ - Enabling logging output causes hang on shutdown
- Issue #2182³⁷⁸⁹ - 1d_stencil seg fault
- Issue #2181³⁷⁹⁰ - Setting small stack size does not change default
- PR #2180³⁷⁹¹ - Changing default bind mode to balanced
- PR #2179³⁷⁹² - adding prefetching_iterator and related tests used for prefetching containers within lambda functions
- PR #2177³⁷⁹³ - Fixing 2176
- Issue #2176³⁷⁹⁴ - Launch process test fails on OSX
- PR #2175³⁷⁹⁵ - Fix unbalanced config/warnings includes, add some new ones
- PR #2174³⁷⁹⁶ - Fix test categorization : regression not unit
- Issue #2172³⁷⁹⁷ - Different performance results
- Issue #2171³⁷⁹⁸ - “negative entry in reference count table” running octotiger on 32 nodes on queenbee
- Issue #2170³⁷⁹⁹ - Error while compiling on Mac + boost 1.60
- PR #2168³⁸⁰⁰ - Fixing problems with is_bitwise_serializable
- Issue #2167³⁸⁰¹ - startup & shutdown function should accept unique_function
- Issue #2166³⁸⁰² - Simple receive_buffer example
- PR #2165³⁸⁰³ - Fix wait all

³⁷⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2190>

³⁷⁸² <https://github.com/STELLAR-GROUP/hpx/pull/2189>

³⁷⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2188>

³⁷⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2187>

³⁷⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2186>

³⁷⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2185>

³⁷⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2184>

³⁷⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2183>

³⁷⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2182>

³⁷⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2181>

³⁷⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2180>

³⁷⁹² <https://github.com/STELLAR-GROUP/hpx/pull/2179>

³⁷⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2177>

³⁷⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2176>

³⁷⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2175>

³⁷⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2174>

³⁷⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2172>

³⁷⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2171>

³⁷⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/2170>

³⁸⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2168>

³⁸⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/2167>

³⁸⁰² <https://github.com/STELLAR-GROUP/hpx/issues/2166>

³⁸⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2165>

- PR #2164³⁸⁰⁴ - Fix wait all
- PR #2163³⁸⁰⁵ - Fix some typos in config tests
- PR #2162³⁸⁰⁶ - Improve #includes
- PR #2160³⁸⁰⁷ - Add inspect check for missing #include <list>
- PR #2159³⁸⁰⁸ - Add missing finalize call to stop test hanging
- PR #2158³⁸⁰⁹ - Algo fixes
- PR #2157³⁸¹⁰ - Stack check
- Issue #2156³⁸¹¹ - OSX reports stack space incorrectly (generic context coroutines)
- Issue #2155³⁸¹² - Race condition suspected in runtime
- PR #2154³⁸¹³ - Replace boost::detail::atomic_count with the new util::atomic_count
- PR #2153³⁸¹⁴ - Fix stack overflow on OSX
- PR #2152³⁸¹⁵ - Define is_bitwise_serializable as is_trivially_copyable when available
- PR #2151³⁸¹⁶ - Adding missing <cstring> for std::mem* functions
- Issue #2150³⁸¹⁷ - Unable to use component clients as action return types
- PR #2149³⁸¹⁸ - std::memmove copies bytes, use bytes* sizeof(type) when copying larger types
- PR #2146³⁸¹⁹ - Adding customization point for parallel copy/move
- PR #2145³⁸²⁰ - Applying changes to address warnings issued by latest version of PVS Studio
- Issue #2148³⁸²¹ - hpx::parallel::copy is broken after trivially copyable changes
- PR #2144³⁸²² - Some minor tweaks to compute prototype
- PR #2143³⁸²³ - Added Boost version support information over OSX platform
- PR #2142³⁸²⁴ - Fixing memory leak in example
- PR #2141³⁸²⁵ - Add missing specializations in execution policies
- PR #2139³⁸²⁶ - This PR fixes a few problems reported by Clang's Undefined Behavior sanitizer

³⁸⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2164>

³⁸⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2163>

³⁸⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2162>

³⁸⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2160>

³⁸⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2159>

³⁸⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2158>

³⁸¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2157>

³⁸¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/2156>

³⁸¹² <https://github.com/STELLAR-GROUP/hpx/issues/2155>

³⁸¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2154>

³⁸¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2153>

³⁸¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2152>

³⁸¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2151>

³⁸¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2150>

³⁸¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2149>

³⁸¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2146>

³⁸²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2145>

³⁸²¹ <https://github.com/STELLAR-GROUP/hpx/issues/2148>

³⁸²² <https://github.com/STELLAR-GROUP/hpx/pull/2144>

³⁸²³ <https://github.com/STELLAR-GROUP/hpx/pull/2143>

³⁸²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2142>

³⁸²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2141>

³⁸²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2139>

- PR #2138³⁸²⁷ - Revert “Adding fedora docs”
- PR #2136³⁸²⁸ - Removed double semicolon
- PR #2135³⁸²⁹ - Add deprecated #include check for hpx_fwd.hpp
- PR #2134³⁸³⁰ - Resolved memory leak in stencil_8
- PR #2133³⁸³¹ - Replace uses of boost pointer containers
- PR #2132³⁸³² - Removing unused typedef
- PR #2131³⁸³³ - Add several include checks for std facilities
- PR #2130³⁸³⁴ - Fixing parcel compression, adding test
- PR #2129³⁸³⁵ - Fix invalid attribute warnings
- Issue #2128³⁸³⁶ - hpx::init seems to segfault
- PR #2127³⁸³⁷ - Making executor_traits N-nary
- PR #2126³⁸³⁸ - GCC 4.6 fails to deduce the correct type in lambda
- PR #2125³⁸³⁹ - Making parcel coalescing test actually test something
- Issue #2124³⁸⁴⁰ - Make a testcase for parcel compression
- Issue #2123³⁸⁴¹ - hpx/hpx/runtime/applier_fwd.hpp - Multiple defined types
- Issue #2122³⁸⁴² - Exception in primary_namespace::resolve_free_list
- Issue #2121³⁸⁴³ - Possible memory leak in 1d_stencil_8
- PR #2120³⁸⁴⁴ - Fixing 2119
- Issue #2119³⁸⁴⁵ - reduce_by_key compilation problems
- Issue #2118³⁸⁴⁶ - Premature unwrapping of boost::ref’ed arguments
- PR #2117³⁸⁴⁷ - Added missing initializer on last constructor for thread_description
- PR #2116³⁸⁴⁸ - Use a lightweight bind implementation when no placeholders are given
- PR #2115³⁸⁴⁹ - Replace boost::shared_ptr with std::shared_ptr

³⁸²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2138>

³⁸²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2136>

³⁸²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2135>

³⁸³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2134>

³⁸³¹ <https://github.com/STELLAR-GROUP/hpx/pull/2133>

³⁸³² <https://github.com/STELLAR-GROUP/hpx/pull/2132>

³⁸³³ <https://github.com/STELLAR-GROUP/hpx/pull/2131>

³⁸³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2130>

³⁸³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2129>

³⁸³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2128>

3837 <https://github.com/STELLAR-GROUP/hpx/pull/2127>3838 <https://github.com/STELLAR-GROUP/hpx/pull/2126>3839 <https://github.com/STELLAR-GROUP/hpx/pull/2125>3840 <https://github.com/STELLAR-GROUP/hpx/issues/2124>3841 <https://github.com/STELLAR-GROUP/hpx/issues/2123>3842 <https://github.com/STELLAR-GROUP/hpx/issues/2122>3843 <https://github.com/STELLAR-GROUP/hpx/issues/2121>3844 <https://github.com/STELLAR-GROUP/hpx/pull/2120>3845 <https://github.com/STELLAR-GROUP/hpx/issues/2119>3846 <https://github.com/STELLAR-GROUP/hpx/issues/2118>3847 <https://github.com/STELLAR-GROUP/hpx/pull/2117>3848 <https://github.com/STELLAR-GROUP/hpx/pull/2116>3849 <https://github.com/STELLAR-GROUP/hpx/pull/2115>

- PR #2114³⁸⁵⁰ - Adding hook functions for executor_parameter_traits supporting timers
- Issue #2113³⁸⁵¹ - Compilation error with gcc version 4.9.3 (MacPorts gcc49 4.9.3_0)
- PR #2112³⁸⁵² - Replace uses of safe_bool with explicit operator bool
- Issue #2111³⁸⁵³ - Compilation error on QT example
- Issue #2110³⁸⁵⁴ - Compilation error when passing non-future argument to unwrapped continuation in dataflow
- Issue #2109³⁸⁵⁵ - Warning while compiling hpx
- Issue #2109³⁸⁵⁶ - Stack trace of last bug causing issues with octotiger
- Issue #2108³⁸⁵⁷ - Stack trace of last bug causing issues with octotiger
- PR #2107³⁸⁵⁸ - Making sure that a missing parcel_coalescing module does not cause startup exceptions
- PR #2106³⁸⁵⁹ - Stop using hpx_fwd.hpp
- Issue #2105³⁸⁶⁰ - coalescing plugin handler is not optional any more
- Issue #2104³⁸⁶¹ - Make executor_traits N-nary
- Issue #2103³⁸⁶² - Build error with octotiger and hpx commit e657426d
- PR #2102³⁸⁶³ - Combining thread data storage
- PR #2101³⁸⁶⁴ - Added repartition version of 1d stencil that uses any performance counter
- PR #2100³⁸⁶⁵ - Drop obsolete TR1 result_of protocol
- PR #2099³⁸⁶⁶ - Replace uses of boost::bind with util::bind
- PR #2098³⁸⁶⁷ - Deprecated inspect checks
- PR #2097³⁸⁶⁸ - Reduce by key, extends #1141
- PR #2096³⁸⁶⁹ - Moving local cache from external to hpx/util
- PR #2095³⁸⁷⁰ - Bump minimum required Boost to 1.50.0
- PR #2094³⁸⁷¹ - Add include checks for several Boost utilities
- Issue #2093³⁸⁷² - ./local_cache.hpp(89): error #303: explicit type is missing (“int” assumed)

³⁸⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2114>

³⁸⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/2113>

³⁸⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2112>

³⁸⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/2111>

³⁸⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2110>

³⁸⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2109>

3856 <https://github.com/STELLAR-GROUP/hpx/issues/2109>3857 <https://github.com/STELLAR-GROUP/hpx/issues/2108>3858 <https://github.com/STELLAR-GROUP/hpx/pull/2107>3859 <https://github.com/STELLAR-GROUP/hpx/pull/2106>3860 <https://github.com/STELLAR-GROUP/hpx/issues/2105>3861 <https://github.com/STELLAR-GROUP/hpx/issues/2104>3862 <https://github.com/STELLAR-GROUP/hpx/issues/2103>3863 <https://github.com/STELLAR-GROUP/hpx/pull/2102>3864 <https://github.com/STELLAR-GROUP/hpx/pull/2101>3865 <https://github.com/STELLAR-GROUP/hpx/pull/2100>3866 <https://github.com/STELLAR-GROUP/hpx/pull/2099>3867 <https://github.com/STELLAR-GROUP/hpx/pull/2098>3868 <https://github.com/STELLAR-GROUP/hpx/pull/2097>3869 <https://github.com/STELLAR-GROUP/hpx/pull/2096>3870 <https://github.com/STELLAR-GROUP/hpx/pull/2095>3871 <https://github.com/STELLAR-GROUP/hpx/pull/2094>3872 <https://github.com/STELLAR-GROUP/hpx/issues/2093>

- PR #2091³⁸⁷³ - Fix for Raspberry pi build
- PR #2090³⁸⁷⁴ - Fix storage size for util::function<>
- PR #2089³⁸⁷⁵ - Fix #2088
- Issue #2088³⁸⁷⁶ - More verbose output from cmake configuration
- PR #2087³⁸⁷⁷ - Making sure init_globally always executes hpx_main
- Issue #2086³⁸⁷⁸ - Race condition with recent HPX
- PR #2085³⁸⁷⁹ - Adding #include checker
- PR #2084³⁸⁸⁰ - Replace boost lock types with standard library ones
- PR #2083³⁸⁸¹ - Simplify packaged task
- PR #2082³⁸⁸² - Updating APEX version for testing
- PR #2081³⁸⁸³ - Cleanup exception headers
- PR #2080³⁸⁸⁴ - Make call_once variadic
- Issue #2079³⁸⁸⁵ - With GNU C++, line 85 of hpx/config/version.hpp causes link failure when linking application
- Issue #2078³⁸⁸⁶ - Simple test fails with _GLIBCXX_DEBUG defined
- PR #2077³⁸⁸⁷ - Instantiate board in nqueen client
- PR #2076³⁸⁸⁸ - Moving coalescing registration to TUs
- PR #2075³⁸⁸⁹ - Fixed some documentation typos
- PR #2074³⁸⁹⁰ - Adding flush-mode to message handler flush
- PR #2073³⁸⁹¹ - Fixing performance regression introduced lately
- PR #2072³⁸⁹² - Refactor local::condition_variable
- PR #2071³⁸⁹³ - Timer based on boost::asio::deadline_timer
- PR #2070³⁸⁹⁴ - Refactor tuple based functionality
- PR #2069³⁸⁹⁵ - Fixed typos

³⁸⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/2091>

³⁸⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2090>

³⁸⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2089>

³⁸⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2088>

³⁸⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2087>

³⁸⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2086>

³⁸⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2085>

³⁸⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2084>

³⁸⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/2083>

³⁸⁸² <https://github.com/STELLAR-GROUP/hpx/pull/2082>

³⁸⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/2081>

³⁸⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2080>

³⁸⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/2079>

³⁸⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2078>

³⁸⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2077>

³⁸⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2076>

³⁸⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2075>

³⁸⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2074>

³⁸⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2073>

³⁸⁹² <https://github.com/STELLAR-GROUP/hpx/pull/2072>

³⁸⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/2071>

³⁸⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2070>

³⁸⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2069>

- Issue #2068³⁸⁹⁶ - Seg fault with octotiger
- PR #2067³⁸⁹⁷ - Algorithm cleanup
- PR #2066³⁸⁹⁸ - Split credit fixes
- PR #2065³⁸⁹⁹ - Rename HPX_MOVABLE_BUT_NOT_COPYABLE to HPX_MOVABLE_ONLY
- PR #2064³⁹⁰⁰ - Fixed some typos in docs
- PR #2063³⁹⁰¹ - Adding example demonstrating template components
- Issue #2062³⁹⁰² - Support component templates
- PR #2061³⁹⁰³ - Replace some uses of lexical_cast<string> with C++11 std::to_string
- PR #2060³⁹⁰⁴ - Replace uses of boost::noncopyable with HPX_NON_COPYABLE
- PR #2059³⁹⁰⁵ - Adding missing for_loop algorithms
- PR #2058³⁹⁰⁶ - Move several definitions to more appropriate headers
- PR #2057³⁹⁰⁷ - Simplify assert_owns_lock and ignore_while_checking
- PR #2056³⁹⁰⁸ - Replacing std::result_of with util::result_of
- PR #2055³⁹⁰⁹ - Fix process launching/connecting back
- PR #2054³⁹¹⁰ - Add a forwarding coroutine header
- PR #2053³⁹¹¹ - Replace uses of boost::unordered_map with std::unordered_map
- PR #2052³⁹¹² - Rewrite tuple unwrap
- PR #2050³⁹¹³ - Replace uses of BOOST_SCOPED_ENUM with C++11 scoped enums
- PR #2049³⁹¹⁴ - Attempt to narrow down split_credit problem
- PR #2048³⁹¹⁵ - Fixing gcc startup hangs
- PR #2047³⁹¹⁶ - Fixing when_xxx and wait_xxx for MSVC12
- PR #2046³⁹¹⁷ - adding persistent_auto_chunk_size and related tests for for_each
- PR #2045³⁹¹⁸ - Fixing HPX_HAVE_THREAD_BACKTRACE_DEPTH build time configuration

³⁸⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2068>

³⁸⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2067>

³⁸⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2066>

³⁸⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2065>

³⁹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2064>

³⁹⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/2063>

³⁹⁰² <https://github.com/STELLAR-GROUP/hpx/issues/2062>

³⁹⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/2061>

³⁹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2060>

³⁹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2059>

³⁹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2058>

³⁹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2057>

³⁹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2056>

³⁹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2055>

³⁹¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2054>

³⁹¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/2053>

³⁹¹² <https://github.com/STELLAR-GROUP/hpx/pull/2052>

³⁹¹³ <https://github.com/STELLAR-GROUP/hpx/pull/2050>

³⁹¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2049>

³⁹¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2048>

³⁹¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2047>

³⁹¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2046>

³⁹¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2045>

- PR #2044³⁹¹⁹ - Adding missing service executor types
- PR #2043³⁹²⁰ - Removing ambiguous definitions for is_future_range and future_range_traits
- PR #2042³⁹²¹ - Clarify that HPX builds can use (much) more than 2GB per process
- PR #2041³⁹²² - Changing future_iterator_traits to support pointers
- Issue #2040³⁹²³ - Improve documentation memory usage warning?
- PR #2039³⁹²⁴ - Coroutine cleanup
- PR #2038³⁹²⁵ - Fix cmake policy CMP0042 warning MACOSX_RPATH
- PR #2037³⁹²⁶ - Avoid redundant specialization of [**unique**]function_nonser
- PR #2036³⁹²⁷ - nvcc dies with an internal error upon pushing/popping warnings inside templates
- Issue #2035³⁹²⁸ - Use a less restrictive iterator definition in hpx::lcos::detail::future_iterator_traits
- PR #2034³⁹²⁹ - Fixing compilation error with thread queue wait time performance counter
- Issue #2033³⁹³⁰ - Compilation error when compiling with thread queue waittime performance counter
- Issue #2032³⁹³¹ - Ambiguous template instantiation for is_future_range and future_range_traits.
- PR #2031³⁹³² - Don't restart timer on every incoming parcel
- PR #2030³⁹³³ - Unify handling of execution policies in parallel algorithms
- PR #2029³⁹³⁴ - Make pkg-config .pc files use .dylib on OSX
- PR #2028³⁹³⁵ - Adding process component
- PR #2027³⁹³⁶ - Making check for compiler compatibility independent on compiler path
- PR #2025³⁹³⁷ - Fixing inspect tool
- PR #2024³⁹³⁸ - Intel13 removal
- PR #2023³⁹³⁹ - Fix errors related to older boost versions and parameter pack expansions in lambdas
- Issue #2022³⁹⁴⁰ - gmake fail: “No rule to make target /usr/lib46/libboost_context-mt.so”
- PR #2021³⁹⁴¹ - Added Sudoku example

³⁹¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2044>

³⁹²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2043>

³⁹²¹ <https://github.com/STELLAR-GROUP/hpx/pull/2042>

³⁹²² <https://github.com/STELLAR-GROUP/hpx/pull/2041>

³⁹²³ <https://github.com/STELLAR-GROUP/hpx/issues/2040>

³⁹²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2039>

³⁹²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2038>

³⁹²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2037>

³⁹²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2036>

³⁹²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/2035>

³⁹²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2034>

³⁹³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2033>

³⁹³¹ <https://github.com/STELLAR-GROUP/hpx/issues/2032>

³⁹³² <https://github.com/STELLAR-GROUP/hpx/pull/2031>

³⁹³³ <https://github.com/STELLAR-GROUP/hpx/pull/2030>

³⁹³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2029>

³⁹³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2028>

³⁹³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2027>

³⁹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2025>

³⁹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2024>

³⁹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2023>

³⁹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/2022>

³⁹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/2021>

- Issue #2020³⁹⁴² - Make errors related to init_globally.cpp example while building HPX out of the box
- PR #2019³⁹⁴³ - Fixed some compilation and cmake errors encountered in nqueen example
- PR #2018³⁹⁴⁴ - For loop algorithms
- PR #2017³⁹⁴⁵ - Non-recursive at_index implementation
- Issue #2016³⁹⁴⁶ - Add index-based for-loops
- Issue #2015³⁹⁴⁷ - Change default bind-mode to balanced
- PR #2014³⁹⁴⁸ - Fixed dataflow if invoked action returns a future
- PR #2013³⁹⁴⁹ - Fixing compilation issues with external example
- PR #2012³⁹⁵⁰ - Added Sierpinski Triangle example
- Issue #2011³⁹⁵¹ - Compilation error while running sample hello_world_component code
- PR #2010³⁹⁵² - Segmented move implemented for hpx::vector
- Issue #2009³⁹⁵³ - pkg-config order incorrect on 14.04 / GCC 4.8
- Issue #2008³⁹⁵⁴ - Compilation error in dataflow of action returning a future
- PR #2007³⁹⁵⁵ - Adding new performance counter exposing overall scheduler time
- PR #2006³⁹⁵⁶ - Function includes
- PR #2005³⁹⁵⁷ - Adding an example demonstrating how to initialize HPX from a global object
- PR #2004³⁹⁵⁸ - Fixing 2000
- PR #2003³⁹⁵⁹ - Adding generation parameter to gather to enable using it more than once
- PR #2002³⁹⁶⁰ - Turn on position independent code to solve link problem with hpx_init
- Issue #2001³⁹⁶¹ - Gathering more than once segfaults
- Issue #2000³⁹⁶² - Undefined reference to hpx::assertion_failed
- Issue #1999³⁹⁶³ - Seg fault in hpx::lcos::base_lco_with_value<*>::set_value_nonvirt() when running octo-tiger
- PR #1998³⁹⁶⁴ - Detect unknown command line options

³⁹⁴² <https://github.com/STELLAR-GROUP/hpx/issues/2020>

³⁹⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/2019>

³⁹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/2018>

³⁹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2017>

³⁹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/2016>

³⁹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/2015>

³⁹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2014>

³⁹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2013>

³⁹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2012>

³⁹⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/2011>

³⁹⁵² <https://github.com/STELLAR-GROUP/hpx/pull/2010>

³⁹⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/2009>

³⁹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/2008>

³⁹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/2007>

³⁹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/2006>

³⁹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/2005>

³⁹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/2004>

³⁹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/2003>

³⁹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/2002>

³⁹⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/2001>

³⁹⁶² <https://github.com/STELLAR-GROUP/hpx/issues/2000>

³⁹⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/1999>

³⁹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1998>

- PR #1997³⁹⁶⁵ - Extending thread description
- PR #1996³⁹⁶⁶ - Adding natvis files to solution (MSVC only)
- Issue #1995³⁹⁶⁷ - Command line handling does not produce error
- PR #1994³⁹⁶⁸ - Possible missing include in test_utils.hpp
- PR #1993³⁹⁶⁹ - Add missing LANGUAGES tag to a hpx_add_compile_flag_if_available() call in CMake-Lists.txt
- PR #1992³⁹⁷⁰ - Fixing shared_executor_test
- PR #1991³⁹⁷¹ - Making sure the winsock library is properly initialized
- PR #1990³⁹⁷² - Fixing bind_test placeholder ambiguity coming from boost-1.60
- PR #1989³⁹⁷³ - Performance tuning
- PR #1987³⁹⁷⁴ - Make configurable size of internal storage in util::function
- PR #1986³⁹⁷⁵ - AGAS Refactoring+1753 Cache mods
- PR #1985³⁹⁷⁶ - Adding missing task_block::run() overload taking an executor
- PR #1984³⁹⁷⁷ - Adding an optimized LRU Cache implementation (for AGAS)
- PR #1983³⁹⁷⁸ - Avoid invoking migration table look up for all objects
- PR #1981³⁹⁷⁹ - Replacing uintptr_t (which is not defined everywhere) with std::size_t
- PR #1980³⁹⁸⁰ - Optimizing LCO continuations
- PR #1979³⁹⁸¹ - Fixing Cori
- PR #1978³⁹⁸² - Fix test check that got broken in hasty fix to memory overflow
- PR #1977³⁹⁸³ - Refactor action traits
- PR #1976³⁹⁸⁴ - Fixes typo in README.rst
- PR #1975³⁹⁸⁵ - Reduce size of benchmark timing arrays to fix test failures
- PR #1974³⁹⁸⁶ - Add action to update data owned by the partitioned_vector component
- PR #1972³⁹⁸⁷ - Adding partitioned_vector SPMD example

³⁹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1997>

³⁹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1996>

³⁹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1995>

³⁹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1994>

³⁹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1993>

³⁹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1992>

³⁹⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/1991>

³⁹⁷² <https://github.com/STELLAR-GROUP/hpx/pull/1990>

³⁹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/1989>

³⁹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1987>

³⁹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1986>

³⁹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1985>

³⁹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1984>

3978 <https://github.com/STELLAR-GROUP/hpx/pull/1983>3979 <https://github.com/STELLAR-GROUP/hpx/pull/1981>3980 <https://github.com/STELLAR-GROUP/hpx/pull/1980>3981 <https://github.com/STELLAR-GROUP/hpx/pull/1979>3982 <https://github.com/STELLAR-GROUP/hpx/pull/1978>3983 <https://github.com/STELLAR-GROUP/hpx/pull/1977>3984 <https://github.com/STELLAR-GROUP/hpx/pull/1976>3985 <https://github.com/STELLAR-GROUP/hpx/pull/1975>3986 <https://github.com/STELLAR-GROUP/hpx/pull/1974>3987 <https://github.com/STELLAR-GROUP/hpx/pull/1972>

- PR #1971³⁹⁸⁸ - Fixing 1965
- PR #1970³⁹⁸⁹ - Papi fixes
- PR #1969³⁹⁹⁰ - Fixing continuation recursions to not depend on fixed amount of recursions
- PR #1968³⁹⁹¹ - More segmented algorithms
- Issue #1967³⁹⁹² - Simplify component implementations
- PR #1966³⁹⁹³ - Migrate components
- Issue #1964³⁹⁹⁴ - fatal error: ‘boost/lockfree/detail/branch_hints.hpp’ file not found
- Issue #1962³⁹⁹⁵ - parallel:copy_if has race condition when used on in place arrays
- PR #1963³⁹⁹⁶ - Fixing Static Parcelport initialization
- PR #1961³⁹⁹⁷ - Fix function target
- Issue #1960³⁹⁹⁸ - Papi counters don’t reset
- PR #1959³⁹⁹⁹ - Fixing 1958
- Issue #1958⁴⁰⁰⁰ - inclusive_scan gives incorrect results with non-commutative operator
- PR #1957⁴⁰⁰¹ - Fixing #1950
- PR #1956⁴⁰⁰² - Sort by key example
- PR #1955⁴⁰⁰³ - Adding regression test for #1946: Hang in wait_all() in distributed run
- Issue #1954⁴⁰⁰⁴ - HPX releases should not use -Werror
- PR #1953⁴⁰⁰⁵ - Adding performance analysis for AGAS cache
- PR #1952⁴⁰⁰⁶ - Adapting test for explicit variadics to fail for gcc 4.6
- PR #1951⁴⁰⁰⁷ - Fixing memory leak
- Issue #1950⁴⁰⁰⁸ - Simplify external builds
- PR #1949⁴⁰⁰⁹ - Fixing yet another lock that is being held during suspension
- PR #1948⁴⁰¹⁰ - Fixed container algorithms for Intel

³⁹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1971>

³⁹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1970>

³⁹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1969>

³⁹⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1968>

³⁹⁹² <https://github.com/STELLAR-GROUP/hpx/issues/1967>

³⁹⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/1966>

³⁹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1964>

³⁹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1962>

³⁹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1963>

³⁹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1961>

³⁹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1960>

³⁹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1959>

⁴⁰⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1958>

⁴⁰⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/1957>

⁴⁰⁰² <https://github.com/STELLAR-GROUP/hpx/pull/1956>

⁴⁰⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/1955>

⁴⁰⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1954>

⁴⁰⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1953>

⁴⁰⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1952>

⁴⁰⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1951>

⁴⁰⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1950>

⁴⁰⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1949>

⁴⁰¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1948>

- PR #1947⁴⁰¹¹ - Adding workaround for tagged_tuple
- Issue #1946⁴⁰¹² - Hang in wait_all() in distributed run
- PR #1945⁴⁰¹³ - Fixed container algorithm tests
- Issue #1944⁴⁰¹⁴ - assertion ‘p.destination_locality() == hpx::get_locality()’ failed
- PR #1943⁴⁰¹⁵ - Fix a couple of compile errors with clang
- PR #1942⁴⁰¹⁶ - Making parcel coalescing functional
- Issue #1941⁴⁰¹⁷ - Re-enable parcel coalescing
- PR #1940⁴⁰¹⁸ - Touching up make_future
- PR #1939⁴⁰¹⁹ - Fixing problems in over-subscription management in the resource manager
- PR #1938⁴⁰²⁰ - Removing use of unified Boost.Thread header
- PR #1937⁴⁰²¹ - Cleaning up the use of Boost.Accumulator headers
- PR #1936⁴⁰²² - Making sure interval timer is started for aggregating performance counters
- PR #1935⁴⁰²³ - Tagged results
- PR #1934⁴⁰²⁴ - Fix remote async with deferred launch policy
- Issue #1933⁴⁰²⁵ - Floating point exception in statistics_counter<boost::accumulators::tag::mean>::get_c...
- PR #1932⁴⁰²⁶ - Removing superfluous includes of boost/lockfree/detail/branch_hints.hpp
- PR #1931⁴⁰²⁷ - fix compilation with clang 3.8.0
- Issue #1930⁴⁰²⁸ - Missing online documentation for HPX 0.9.11
- PR #1929⁴⁰²⁹ - LWG2485: get() should be overloaded for const tuple&&
- PR #1928⁴⁰³⁰ - Revert “Using ninja for circle-ci builds”
- PR #1927⁴⁰³¹ - Using ninja for circle-ci builds
- PR #1926⁴⁰³² - Fixing serialization of std::array
- Issue #1925⁴⁰³³ - Issues with static HPX libraries

⁴⁰¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1947>

⁴⁰¹² <https://github.com/STELLAR-GROUP/hpx/issues/1946>

⁴⁰¹³ <https://github.com/STELLAR-GROUP/hpx/pull/1945>

⁴⁰¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1944>

⁴⁰¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1943>

⁴⁰¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1942>

⁴⁰¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1941>

⁴⁰¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1940>

⁴⁰¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1939>

⁴⁰²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1938>

⁴⁰²¹ <https://github.com/STELLAR-GROUP/hpx/pull/1937>

⁴⁰²² <https://github.com/STELLAR-GROUP/hpx/pull/1936>

⁴⁰²³ <https://github.com/STELLAR-GROUP/hpx/pull/1935>

⁴⁰²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1934>

⁴⁰²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1933>

⁴⁰²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1932>

⁴⁰²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1931>

⁴⁰²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1930>

⁴⁰²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1929>

⁴⁰³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1928>

⁴⁰³¹ <https://github.com/STELLAR-GROUP/hpx/pull/1927>

⁴⁰³² <https://github.com/STELLAR-GROUP/hpx/pull/1926>

⁴⁰³³ <https://github.com/STELLAR-GROUP/hpx/issues/1925>

- Issue #1924⁴⁰³⁴ - Performance degrading over time
- Issue #1923⁴⁰³⁵ - serialization of std::array appears broken in latest commit
- PR #1922⁴⁰³⁶ - Container algorithms
- PR #1921⁴⁰³⁷ - Tons of smaller quality improvements
- Issue #1920⁴⁰³⁸ - Seg fault in hpx::serialization::output_archive::add_gid when running octotiger
- Issue #1919⁴⁰³⁹ - Intel 15 compiler bug preventing HPX build
- PR #1918⁴⁰⁴⁰ - Address sanitizer fixes
- PR #1917⁴⁰⁴¹ - Fixing compilation problems of parallel::sort with Intel compilers
- PR #1916⁴⁰⁴² - Making sure code compiles if HPX_WITH_HWLOC=Off
- Issue #1915⁴⁰⁴³ - max_cores undefined if HPX_WITH_HWLOC=Off
- PR #1913⁴⁰⁴⁴ - Add utility member functions for partitioned_vector
- PR #1912⁴⁰⁴⁵ - Adding support for invoking actions to dataflow
- PR #1911⁴⁰⁴⁶ - Adding first batch of container algorithms
- PR #1910⁴⁰⁴⁷ - Keep cmake_module_path
- PR #1909⁴⁰⁴⁸ - Fix mpirun with pbs
- PR #1908⁴⁰⁴⁹ - Changing parallel::sort to return the last iterator as proposed by N4560
- PR #1907⁴⁰⁵⁰ - Adding a minimum version for Open MPI
- PR #1906⁴⁰⁵¹ - Updates to the Release Procedure
- PR #1905⁴⁰⁵² - Fixing #1903
- PR #1904⁴⁰⁵³ - Making sure std containers are cleared before serialization loads data
- Issue #1903⁴⁰⁵⁴ - When running octotiger, I get: assertion '(*new_gids_)[gid].size() == 1' failed: HPX(assertion_failure)
- Issue #1902⁴⁰⁵⁵ - Immediate crash when running hpx/octotiger with _GLIBCXX_DEBUG defined.
- PR #1901⁴⁰⁵⁶ - Making non-serializable classes non-serializable

⁴⁰³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1924>

⁴⁰³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1923>

⁴⁰³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1922>

⁴⁰³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1921>

⁴⁰³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1920>

⁴⁰³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1919>

⁴⁰⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1918>

⁴⁰⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/1917>

⁴⁰⁴² <https://github.com/STELLAR-GROUP/hpx/pull/1916>

⁴⁰⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/1915>

⁴⁰⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1913>

⁴⁰⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1912>

⁴⁰⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1911>

⁴⁰⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1910>

⁴⁰⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1909>

⁴⁰⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1908>

⁴⁰⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1907>

⁴⁰⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/1906>

⁴⁰⁵² <https://github.com/STELLAR-GROUP/hpx/pull/1905>

⁴⁰⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/1904>

⁴⁰⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1903>

⁴⁰⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1902>

⁴⁰⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1901>

- Issue #1900⁴⁰⁵⁷ - Two possible issues with std::list serialization
- PR #1899⁴⁰⁵⁸ - Fixing a problem with credit splitting as revealed by #1898
- Issue #1898⁴⁰⁵⁹ - Accessing component from locality where it was not created segfaults
- PR #1897⁴⁰⁶⁰ - Changing parallel::sort to return the last iterator as proposed by N4560
- Issue #1896⁴⁰⁶¹ - version 1.0?
- Issue #1895⁴⁰⁶² - Warning comment on numa_allocator is not very clear
- PR #1894⁴⁰⁶³ - Add support for compilers that have thread_local
- PR #1893⁴⁰⁶⁴ - Fixing 1890
- PR #1892⁴⁰⁶⁵ - Adds typed future_type for executor_traits
- PR #1891⁴⁰⁶⁶ - Fix wording in certain parallel algorithm docs
- Issue #1890⁴⁰⁶⁷ - Invoking papi counters give segfault
- PR #1889⁴⁰⁶⁸ - Fixing problems as reported by clang-check
- PR #1888⁴⁰⁶⁹ - WIP parallel is_heap
- PR #1887⁴⁰⁷⁰ - Fixed resetting performance counters related to idle-rate, etc
- Issue #1886⁴⁰⁷¹ - Run hpx with qsub does not work
- PR #1885⁴⁰⁷² - Warning cleaning pass
- PR #1884⁴⁰⁷³ - Add missing parallel algorithm header
- PR #1883⁴⁰⁷⁴ - Add feature test for thread_local on Clang for TLS
- PR #1882⁴⁰⁷⁵ - Fix some redundant qualifiers
- Issue #1881⁴⁰⁷⁶ - Unable to compile Octotiger using HPX and Intel MPI on SuperMIC
- Issue #1880⁴⁰⁷⁷ - clang with libc++ on Linux needs TLS case
- PR #1879⁴⁰⁷⁸ - Doc fixes for #1868
- PR #1878⁴⁰⁷⁹ - Simplify functions

⁴⁰⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1900>

⁴⁰⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1899>

⁴⁰⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1898>

⁴⁰⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1897>

⁴⁰⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/1896>

⁴⁰⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1895>

⁴⁰⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/1894>

⁴⁰⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1893>

⁴⁰⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1892>

⁴⁰⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1891>

⁴⁰⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1890>

⁴⁰⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1889>

⁴⁰⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1888>

⁴⁰⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1887>

⁴⁰⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/1886>

⁴⁰⁷² <https://github.com/STELLAR-GROUP/hpx/pull/1885>

⁴⁰⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/1884>

⁴⁰⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1883>

⁴⁰⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1882>

⁴⁰⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1881>

⁴⁰⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1880>

⁴⁰⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1879>

⁴⁰⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1878>

- PR #1877⁴⁰⁸⁰ - Removing most usage of Boost.Config
- PR #1876⁴⁰⁸¹ - Add missing parallel algorithms to algorithm.hpp
- PR #1875⁴⁰⁸² - Simplify callables
- PR #1874⁴⁰⁸³ - Address long standing FIXME on using std::unique_ptr with incomplete types
- PR #1873⁴⁰⁸⁴ - Fixing 1871
- PR #1872⁴⁰⁸⁵ - Making sure PBS environment uses specified node list even if no PBS_NODEFILE env is available
- Issue #1871⁴⁰⁸⁶ - Fortran checks should be optional
- PR #1870⁴⁰⁸⁷ - Touch local::mutex
- PR #1869⁴⁰⁸⁸ - Documentation refactoring based off #1868
- PR #1867⁴⁰⁸⁹ - Embrace static_assert
- PR #1866⁴⁰⁹⁰ - Fix #1803 with documentation refactoring
- PR #1865⁴⁰⁹¹ - Setting OUTPUT_NAME as target properties
- PR #1863⁴⁰⁹² - Use SYSTEM for boost includes
- PR #1862⁴⁰⁹³ - Minor cleanups
- PR #1861⁴⁰⁹⁴ - Minor Corrections for Release
- PR #1860⁴⁰⁹⁵ - Fixing hpx gdb script
- Issue #1859⁴⁰⁹⁶ - reset_active_counters resets times and thread counts before some of the counters are evaluated
- PR #1858⁴⁰⁹⁷ - Release V0.9.11
- PR #1857⁴⁰⁹⁸ - removing diskperf example from 9.11 release
- PR #1856⁴⁰⁹⁹ - fix return in packaged_task_base::reset()
- Issue #1842⁴¹⁰⁰ - Install error: file INSTALL cannot find libhpx_parcel_coalescing.so.0.9.11
- PR #1839⁴¹⁰¹ - Adding fedora docs
- PR #1824⁴¹⁰² - Changing version on master to V0.9.12

4080 <https://github.com/STELLAR-GROUP/hpx/pull/1877>
4081 <https://github.com/STELLAR-GROUP/hpx/pull/1876>
4082 <https://github.com/STELLAR-GROUP/hpx/pull/1875>
4083 <https://github.com/STELLAR-GROUP/hpx/pull/1874>
4084 <https://github.com/STELLAR-GROUP/hpx/pull/1873>
4085 <https://github.com/STELLAR-GROUP/hpx/pull/1872>
4086 <https://github.com/STELLAR-GROUP/hpx/issues/1871>
4087 <https://github.com/STELLAR-GROUP/hpx/pull/1870>
4088 <https://github.com/STELLAR-GROUP/hpx/pull/1869>
4089 <https://github.com/STELLAR-GROUP/hpx/pull/1867>
4090 <https://github.com/STELLAR-GROUP/hpx/pull/1866>
4091 <https://github.com/STELLAR-GROUP/hpx/pull/1865>
4092 <https://github.com/STELLAR-GROUP/hpx/pull/1863>
4093 <https://github.com/STELLAR-GROUP/hpx/pull/1862>
4094 <https://github.com/STELLAR-GROUP/hpx/pull/1861>
4095 <https://github.com/STELLAR-GROUP/hpx/pull/1860>
4096 <https://github.com/STELLAR-GROUP/hpx/issues/1859>
4097 <https://github.com/STELLAR-GROUP/hpx/pull/1858>
4098 <https://github.com/STELLAR-GROUP/hpx/pull/1857>
4099 <https://github.com/STELLAR-GROUP/hpx/pull/1856>
4100 <https://github.com/STELLAR-GROUP/hpx/issues/1842>
4101 <https://github.com/STELLAR-GROUP/hpx/pull/1839>
4102 <https://github.com/STELLAR-GROUP/hpx/pull/1824>

- PR #1818⁴¹⁰³ - Fixing #1748
- Issue #1815⁴¹⁰⁴ - seg fault in AGAS
- Issue #1803⁴¹⁰⁵ - wait_all documentation
- Issue #1796⁴¹⁰⁶ - Outdated documentation to be revised
- Issue #1759⁴¹⁰⁷ - glibc munmap_chunk or free(): invalid pointer on SuperMIC
- Issue #1753⁴¹⁰⁸ - HPX performance degrades with time since execution begins
- Issue #1748⁴¹⁰⁹ - All public HPX headers need to be self contained
- PR #1719⁴¹¹⁰ - How to build HPX with Visual Studio
- Issue #1684⁴¹¹¹ - Race condition when using --hpx:connect?
- PR #1658⁴¹¹² - Add serialization for std::set (as there is for std::vector and std::map)
- PR #1641⁴¹¹³ - Generic client
- Issue #1632⁴¹¹⁴ - heartbeat example fails on separate nodes
- PR #1603⁴¹¹⁵ - Adds preferred namespace check to inspect tool
- Issue #1559⁴¹¹⁶ - Extend inspect tool
- Issue #1523⁴¹¹⁷ - Remote async with deferred launch policy never executes
- Issue #1472⁴¹¹⁸ - Serialization issues
- Issue #1457⁴¹¹⁹ - Implement N4392: C++ Latches and Barriers
- PR #1444⁴¹²⁰ - Enabling usage of moveonly types for component construction
- Issue #1407⁴¹²¹ - The Intel 13 compiler has failing unit tests
- Issue #1405⁴¹²² - Allow component constructors to take movable only types
- Issue #1265⁴¹²³ - Enable dataflow() to be usable with actions
- Issue #1236⁴¹²⁴ - NUMA aware allocators
- Issue #802⁴¹²⁵ - Fix Broken Examples

⁴¹⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/1818>

⁴¹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1815>

⁴¹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1803>

⁴¹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1796>

⁴¹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1759>

⁴¹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1753>

⁴¹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1748>

⁴¹¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1719>

⁴¹¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1684>

⁴¹¹² <https://github.com/STELLAR-GROUP/hpx/pull/1658>

⁴¹¹³ <https://github.com/STELLAR-GROUP/hpx/pull/1641>

⁴¹¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1632>

⁴¹¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1603>

⁴¹¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1559>

⁴¹¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1523>

⁴¹¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1472>

⁴¹¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1457>

⁴¹²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1444>

⁴¹²¹ <https://github.com/STELLAR-GROUP/hpx/issues/1407>

⁴¹²² <https://github.com/STELLAR-GROUP/hpx/issues/1405>

⁴¹²³ <https://github.com/STELLAR-GROUP/hpx/issues/1265>

⁴¹²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1236>

⁴¹²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/802>

- Issue #559⁴¹²⁶ - Add hpx::migrate facility
- Issue #449⁴¹²⁷ - Make actions with template arguments usable and add documentation
- Issue #279⁴¹²⁸ - Refactor addressing_service into a base class and two derived classes
- Issue #224⁴¹²⁹ - Changing thread state metadata is not thread safe
- Issue #55⁴¹³⁰ - Uniform syntax for enums should be implemented

2.10.15 HPX V0.9.11 (Nov 11, 2015)

Our main focus for this release was the design and development of a coherent set of higher-level APIs exposing various types of parallelism to the application programmer. We introduced the concepts of an `executor`, which can be used to customize the `where` and `when` of execution of tasks in the context of parallelizing codes. We extended all APIs related to managing parallel tasks to support executors which gives the user the choice of either using one of the predefined executor types or to provide its own, possibly application specific, executor. We paid very close attention to align all of these changes with the existing C++ Standards documents or with the ongoing proposals for standardization.

This release is the first after our change to a new development policy. We switched all development to be strictly performed on branches only, all direct commits to our main branch (`master`) are prohibited. Any change has to go through a peer review before it will be merged to `master`. As a result the overall stability of our code base has significantly increased, the development process itself has been simplified. This change manifests itself in a large number of pull-requests which have been merged (please see below for a full list of closed issues and pull-requests). All in all for this release, we closed almost 100 issues and merged over 290 pull-requests. There have been over 1600 commits to the `master` branch since the last release.

General changes

- We are moving into the direction of unifying managed and simple components. As such, the classes `hpx::components::component` and `hpx::components::component_base` have been added which currently just forward to the currently existing simple component facilities. The examples have been converted to only use those two classes.
- Added integration with the [CircleCI](#)⁴¹³¹ hosted continuous integration service. This gives us constant and immediate feedback on the health of our `master` branch.
- The compiler configuration subsystem in the build system has been reimplemented. Instead of using Boost.Config we now use our own lightweight set of cmake scripts to determine the available language and library features supported by the used compiler.
- The API for creating instances of components has been consolidated. All component instances should be created using the `hpx::new_`. It allows one to instantiate both, single component instances and multiple component instances. The placement of the created components can be controlled by special distribution policies. Please see the corresponding documentation outlining the use of `hpx::new_`.
- Introduced four new distribution policies which can be used with many API functions which traditionally expected to be used with a locality id. The new distribution policies are:
 - `hpx::components::default_distribution_policy` which tries to place multiple component instances as evenly as possible.

⁴¹²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/559>

⁴¹²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/449>

⁴¹²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/279>

⁴¹²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/224>

⁴¹³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/55>

⁴¹³¹ <https://circleci.com/gh/STELLAR-GROUP/hpx>

- `hpx::components::colocating_distribution_policy` which will refer to the locality where a given component instance is currently placed.
 - `hpx::components::binpacking_distribution_policy` which will place multiple component instances as evenly as possible based on any performance counter.
 - `hpx::components::target_distribution_policy` which allows one to represent a given locality in the context of a distribution policy.
- The new distribution policies can now be also used with `hpx::async`. This change also deprecates `hpx::async_colocated(id, ...)` which now is replaced by a distribution policy: `hpx::async(hpx::colocated(id), ...)`.
 - The `hpx::vector` and `hpx::unordered_map` data structures can now be used with the new distribution policies as well.
 - The parallel facility `hpx::parallel::task_region` has been renamed to `hpx::parallel::task_block` based on the changes in the corresponding standardization proposal [N4411⁴¹³²](#).
 - Added extensions to the parallel facility `hpx::parallel::task_block` allowing to combine a `task_block` with an execution policy. This implies a minor breaking change as the `hpx::parallel::task_block` is now a template.
 - Added new LCOs: `hpx::lcos::latch` and `hpx::lcos::local::latch` which semantically conform to the proposed `std::latch` (see [N4399⁴¹³³](#)).
 - Added performance counters exposing data related to data transferred by input/output (filesystem) operations (thanks to Maciej Brodowicz).
 - Added performance counters allowing to track the number of action invocations (local and remote invocations).
 - Added new command line options `-hpx:print-counter-at` and `-hpx:reset-counters`.
 - The `hpx::vector` component has been renamed to `hpx::partitioned_vector` to make it explicit that the underlying memory is not contiguous.
 - Introduced a completely new and uniform higher-level parallelism API which is based on executors. All existing parallelism APIs have been adapted to this. We have added a large number of different executor types, such as a numa-aware executor, a this-thread executor, etc.
 - Added support for the MingW toolchain on Windows (thanks to Eric Lemanissier).
 - HPX now includes support for APEX, (Autonomic Performance Environment for eXascale). APEX is an instrumentation and software adaptation library that provides an interface to TAU profiling / tracing as well as runtime adaptation of HPX applications through policy definitions. For more information and documentation, please see <https://github.com/UO-OACISS/xpress-apex>. To enable APEX at configuration time, specify `-DHPX_WITH_APEX=On`. To also include support for TAU profiling, specify `-DHPX_WITH_TAU=On` and specify the `-DTAU_ROOT`, `-DTAU_ARCH` and `-DTAU_OPTIONS` cmake parameters.
 - We have implemented many more of the *Using parallel algorithms*. Please see [Issue #1141⁴¹³⁴](#) for the list of all available parallel algorithms (thanks to Daniel Bourgeois and John Biddiscombe for contributing their work).

⁴¹³² <http://wg21.link/n4411>⁴¹³³ <http://wg21.link/n4399>⁴¹³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1141>

Breaking changes

- We are moving into the direction of unifying managed and simple components. In order to stop exposing the old facilities, all examples have been converted to use the new classes. The breaking change in this release is that performance counters are now a `hpx::components::component_base` instead of `hpx::components::managed_component_base`.
- We removed the support for stackless threads. It turned out that there was no performance benefit when using stackless threads. As such, we decided to clean up our codebase. This feature was not documented.
- The CMake project name has changed from ‘hpx’ to ‘HPX’ for consistency and compatibility with naming conventions and other CMake projects. Generated config files go into `<prefix>/lib/cmake/HPX` and not `<prefix>/lib/cmake/hpx`.
- The macro `HPX_REGISTER_MINIMAL_COMPONENT_FACTORY` has been deprecated. Please use `HPX_REGISTER_COMPONENT` instead. The old macro will be removed in the next release.
- The obsolete `distributing_factory` and `binpacking_factory` components have been removed. The corresponding functionality is now provided by the `hpx::new_` API function in conjunction with the `hpx::default_layout` and `hpx::binpacking` distribution policies (`hpx::components::default_distribution_policy` and `hpx::components::binpacking_distribution_policy`)
- The API function `hpx::new_colocated` has been deprecated. Please use the consolidated API `hpx::new_` in conjunction with the new `hpx::colocated` distribution policy (`hpx::components::colocating_distribution_policy`) instead. The old API function will still be available for at least one release of *HPX* if the configuration variable `HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY` is enabled.
- The API function `hpx::async_colocated` has been deprecated. Please use the consolidated API `hpx::async` in conjunction with the new `hpx::colocated` distribution policy (`hpx::components::colocating_distribution_policy`) instead. The old API function will still be available for at least one release of *HPX* if the configuration variable `HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY` is enabled.
- The obsolete `remote_object` component has been removed.
- Replaced the use of Boost.Serialization with our own solution. While the new version is mostly compatible with Boost.Serialization, this change requires some minor code modifications in user code. For more information, please see the corresponding announcement⁴¹³⁵ on the `hpx-users@stellar.cct.lsu.edu` mailing list.
- The names used by cmake to influence various configuration options have been unified. The new naming scheme relies on all configuration constants to start with `HPX_WITH_...`, while the preprocessor constant which is used at build time starts with `HPX_HAVE_...`. For instance, the former cmake command line `-DHPX_MALLOC=...` now has to be specified a `-DHPX_WITH_MALLOC=...` and will cause the preprocessor constant `HPX_HAVE_MALLOC` to be defined. The actual name of the constant (i.e. `MALLOC`) has not changed. Please see the corresponding documentation for more details (*CMake variables used to configure HPX*).
- The `get_gid()` functions exposed by the component base classes `hpx::components::server::simple_component_base`, `hpx::components::server::managed_component` and `hpx::components::server::fixed_component_base` have been replaced by two new functions: `get_unmanaged_id()` and `get_id()`. To enable the old function name for backwards compatibility, use the cmake configuration option `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY=On`.
- All functions which were named `get_gid()` but were returning `hpx::id_type` have been renamed to `get_id()`. To enable the old function names for backwards compatibility, use the cmake configuration option `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY=On`.

⁴¹³⁵ <http://thread.gmane.org/gmane.comp.lib.hpx.devel/196>

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- PR #1855⁴¹³⁶ - Completely removing external/endian
- PR #1854⁴¹³⁷ - Don't pollute CMAKE_CXX_FLAGS through find_package()
- PR #1853⁴¹³⁸ - Updating CMake configuration to get correct version of TAU library
- PR #1852⁴¹³⁹ - Fixing Performance Problems with MPI Parcelport
- PR #1851⁴¹⁴⁰ - Fixing hpx_add_link_flag() and hpx_remove_link_flag()
- PR #1850⁴¹⁴¹ - Fixing 1836, adding parallel::sort
- PR #1849⁴¹⁴² - Fixing configuration for use of more than 64 cores
- PR #1848⁴¹⁴³ - Change default APEX version for release
- PR #1847⁴¹⁴⁴ - Fix client_base::then on release
- PR #1846⁴¹⁴⁵ - Removing broken lcos::local::channel from release
- PR #1845⁴¹⁴⁶ - Adding example demonstrating a possible safe-object implementation to release
- PR #1844⁴¹⁴⁷ - Removing stubs from accumulator examples
- PR #1843⁴¹⁴⁸ - Don't pollute CMAKE_CXX_FLAGS through find_package()
- PR #1841⁴¹⁴⁹ - Fixing client_base<>::then
- PR #1840⁴¹⁵⁰ - Adding example demonstrating a possible safe-object implementation
- PR #1838⁴¹⁵¹ - Update version rc1
- PR #1837⁴¹⁵² - Removing broken lcos::local::channel
- PR #1835⁴¹⁵³ - Adding explicit move constructor and assignment operator to hpx::lcos::promise
- PR #1834⁴¹⁵⁴ - Making hpx::lcos::promise move-only
- PR #1833⁴¹⁵⁵ - Adding fedora docs
- Issue #1832⁴¹⁵⁶ - hpx::lcos::promise<> must be move-only

⁴¹³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1855>

⁴¹³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1854>

⁴¹³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1853>

⁴¹³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1852>

⁴¹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1851>

⁴¹⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/1850>

⁴¹⁴² <https://github.com/STELLAR-GROUP/hpx/pull/1849>

⁴¹⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/1848>

⁴¹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1847>

⁴¹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1846>

⁴¹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1845>

⁴¹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1844>

⁴¹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1843>

⁴¹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1841>

⁴¹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1840>

⁴¹⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/1838>

⁴¹⁵² <https://github.com/STELLAR-GROUP/hpx/pull/1837>

⁴¹⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/1835>

⁴¹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1834>

⁴¹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1833>

⁴¹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1832>

- PR #1831⁴¹⁵⁷ - Fixing resource manager gcc5.2
- PR #1830⁴¹⁵⁸ - Fix intel13
- PR #1829⁴¹⁵⁹ - Unbreaking thread test
- PR #1828⁴¹⁶⁰ - Fixing #1620
- PR #1827⁴¹⁶¹ - Fixing a memory management issue for the Parquet application
- Issue #1826⁴¹⁶² - Memory management issue in hpx::lcos::promise
- PR #1825⁴¹⁶³ - Adding hpx::components::component and hpx::components::component_base
- PR #1823⁴¹⁶⁴ - Adding git commit id to circleci build
- PR #1822⁴¹⁶⁵ - applying fixes suggested by clang 3.7
- PR #1821⁴¹⁶⁶ - Hyperlink fixes
- PR #1820⁴¹⁶⁷ - added parallel multi-locality sanity test
- PR #1819⁴¹⁶⁸ - Fixing #1667
- Issue #1817⁴¹⁶⁹ - Hyperlinks generated by inspect tool are wrong
- PR #1816⁴¹⁷⁰ - Support hpxrx
- PR #1814⁴¹⁷¹ - Fix async to dispatch to the correct locality in all cases
- Issue #1813⁴¹⁷² - async(launch:..., action(), ...) always invokes locally
- PR #1812⁴¹⁷³ - fixed syntax error in CMakeLists.txt
- PR #1811⁴¹⁷⁴ - Agas optimizations
- PR #1810⁴¹⁷⁵ - drop superfluous typedefs
- PR #1809⁴¹⁷⁶ - Allow HPX to be used as an optional package in 3rd party code
- PR #1808⁴¹⁷⁷ - Fixing #1723
- PR #1807⁴¹⁷⁸ - Making sure resolve_localities does not hang during normal operation
- Issue #1806⁴¹⁷⁹ - Spinlock no longer movable and deletes operator ‘=’, breaks MiniGhost

⁴¹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1831>

⁴¹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1830>

⁴¹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1829>

⁴¹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1828>

⁴¹⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/1827>

⁴¹⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1826>

⁴¹⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/1825>

⁴¹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1823>

⁴¹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1822>

⁴¹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1821>

⁴¹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1820>

⁴¹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1819>

⁴¹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1817>

⁴¹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1816>

⁴¹⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/1814>

⁴¹⁷² <https://github.com/STELLAR-GROUP/hpx/issues/1813>

⁴¹⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/1812>

⁴¹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1811>

⁴¹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1810>

⁴¹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1809>

⁴¹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1808>

⁴¹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1807>

⁴¹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1806>

- Issue #1804⁴¹⁸⁰ - register_with_basename causes hangs
- PR #1801⁴¹⁸¹ - Enhanced the inspect tool to take user directly to the problem with hyperlinks
- Issue #1800⁴¹⁸² - Problems compiling application on smic
- PR #1799⁴¹⁸³ - Fixing cv exceptions
- PR #1798⁴¹⁸⁴ - Documentation refactoring & updating
- PR #1797⁴¹⁸⁵ - Updating the activeharmony CMake module
- PR #1795⁴¹⁸⁶ - Fixing cv
- PR #1794⁴¹⁸⁷ - Fix connect with hpx::runtime_mode_connect
- PR #1793⁴¹⁸⁸ - fix a wrong use of HPX_MAX_CPU_COUNT instead of HPX_HAVE_MAX_CPU_COUNT
- PR #1792⁴¹⁸⁹ - Allow for default constructed parcel instances to be moved
- PR #1791⁴¹⁹⁰ - Fix connect with hpx::runtime_mode_connect
- Issue #1790⁴¹⁹¹ - assertion action_.get () failed: HPX(assertion_failure) when running Octotiger with pull request 1786
- PR #1789⁴¹⁹² - Fixing discover_counter_types API function
- Issue #1788⁴¹⁹³ - connect with hpx::runtime_mode_connect
- Issue #1787⁴¹⁹⁴ - discover_counter_types not working
- PR #1786⁴¹⁹⁵ - Changing addressing_service to use std::unordered_map instead of std::map
- PR #1785⁴¹⁹⁶ - Fix is_iterator for container algorithms
- PR #1784⁴¹⁹⁷ - Adding new command line options:
- PR #1783⁴¹⁹⁸ - Minor changes for APEX support
- PR #1782⁴¹⁹⁹ - Drop legacy forwarding action traits
- PR #1781⁴²⁰⁰ - Attempt to resolve the race between cv::wait_xxx and cv::notify_all
- PR #1780⁴²⁰¹ - Removing serialize_sequence
- PR #1779⁴²⁰² - Fixed #1501: hwloc configuration options are wrong for MIC

⁴¹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1804>

⁴¹⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/1801>

⁴¹⁸² <https://github.com/STELLAR-GROUP/hpx/issues/1800>

⁴¹⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/1799>

⁴¹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1798>

⁴¹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1797>

⁴¹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1795>

⁴¹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1794>

⁴¹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1793>

4189 <https://github.com/STELLAR-GROUP/hpx/pull/1792>4190 <https://github.com/STELLAR-GROUP/hpx/pull/1791>4191 <https://github.com/STELLAR-GROUP/hpx/issues/1790>4192 <https://github.com/STELLAR-GROUP/hpx/pull/1789>4193 <https://github.com/STELLAR-GROUP/hpx/issues/1788>4194 <https://github.com/STELLAR-GROUP/hpx/issues/1787>4195 <https://github.com/STELLAR-GROUP/hpx/pull/1786>4196 <https://github.com/STELLAR-GROUP/hpx/pull/1785>4197 <https://github.com/STELLAR-GROUP/hpx/pull/1784>4198 <https://github.com/STELLAR-GROUP/hpx/pull/1783>4199 <https://github.com/STELLAR-GROUP/hpx/pull/1782>4200 <https://github.com/STELLAR-GROUP/hpx/pull/1781>4201 <https://github.com/STELLAR-GROUP/hpx/pull/1780>4202 <https://github.com/STELLAR-GROUP/hpx/pull/1779>

- PR #1778⁴²⁰³ - Removing ability to enable/disable parcel handling
- PR #1777⁴²⁰⁴ - Completely removing stackless threads
- PR #1776⁴²⁰⁵ - Cleaning up util/plugin
- PR #1775⁴²⁰⁶ - Agas fixes
- PR #1774⁴²⁰⁷ - Action invocation count
- PR #1773⁴²⁰⁸ - replaced MSVC variable with WIN32
- PR #1772⁴²⁰⁹ - Fixing Problems in MPI parcelport and future serialization.
- PR #1771⁴²¹⁰ - Fixing intel 13 compiler errors related to variadic template template parameters for `lcos::when_` tests
- PR #1770⁴²¹¹ - Forwarding decay to `std::`
- PR #1769⁴²¹² - Add more characters with special regex meaning to the existing patch
- PR #1768⁴²¹³ - Adding test for `receive_buffer`
- PR #1767⁴²¹⁴ - Making sure that uptime counter throws exception on any attempt to be reset
- PR #1766⁴²¹⁵ - Cleaning up code related to throttling scheduler
- PR #1765⁴²¹⁶ - Restricting `thread_data` to creating only with `intrusive_pointers`
- PR #1764⁴²¹⁷ - Fixing 1763
- Issue #1763⁴²¹⁸ - UB in `thread_data::operator delete`
- PR #1762⁴²¹⁹ - Making sure all serialization registries/factories are unique
- PR #1761⁴²²⁰ - Fixed #1751: `hpx::future::wait_for` fails a simple test
- PR #1758⁴²²¹ - Fixing #1757
- Issue #1757⁴²²² - pinning not correct using `-hpx:bind`
- Issue #1756⁴²²³ - compilation error with MinGW
- PR #1755⁴²²⁴ - Making output serialization const-correct
- Issue #1753⁴²²⁵ - HPX performance degrades with time since execution begins

⁴²⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/1778>

⁴²⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1777>

⁴²⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1776>

⁴²⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1775>

⁴²⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1774>

⁴²⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1773>

⁴²⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1772>

⁴²¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1771>

⁴²¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1770>

⁴²¹² <https://github.com/STELLAR-GROUP/hpx/pull/1769>

⁴²¹³ <https://github.com/STELLAR-GROUP/hpx/pull/1768>

⁴²¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1767>

⁴²¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1766>

⁴²¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1765>

⁴²¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1764>

⁴²¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1763>

⁴²¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1762>

⁴²²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1761>

⁴²²¹ <https://github.com/STELLAR-GROUP/hpx/pull/1758>

⁴²²² <https://github.com/STELLAR-GROUP/hpx/issues/1757>

⁴²²³ <https://github.com/STELLAR-GROUP/hpx/issues/1756>

⁴²²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1755>

⁴²²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1753>

- Issue #1752⁴²²⁶ - Error in AGAS
- Issue #1751⁴²²⁷ - hpx::future::wait_for fails a simple test
- PR #1750⁴²²⁸ - Removing hpx_fwd.hpp includes
- PR #1749⁴²²⁹ - Simplify result_of and friends
- PR #1747⁴²³⁰ - Removed superfluous code from message_buffer.hpp
- PR #1746⁴²³¹ - Tuple dependencies
- Issue #1745⁴²³² - Broken when_some which takes iterators
- PR #1744⁴²³³ - Refining archive interface
- PR #1743⁴²³⁴ - Fixing when_all when only a single future is passed
- PR #1742⁴²³⁵ - Config includes
- PR #1741⁴²³⁶ - Os executors
- Issue #1740⁴²³⁷ - hpx::promise has some problems
- PR #1739⁴²³⁸ - Parallel composition with generic containers
- Issue #1738⁴²³⁹ - After building program and successfully linking to a version of hpx DHPX_DIR seems to be ignored
- Issue #1737⁴²⁴⁰ - Uptime problems
- PR #1736⁴²⁴¹ - added convenience c-tor and begin()/end() to serialize_buffer
- PR #1735⁴²⁴² - Config includes
- PR #1734⁴²⁴³ - Fixed #1688: Add timer counters for tfunc_total and exec_total
- Issue #1733⁴²⁴⁴ - Add unit test for hpx/lcos/local/receive_buffer.hpp
- PR #1732⁴²⁴⁵ - Renaming get_os_thread_count
- PR #1731⁴²⁴⁶ - Basename registration
- Issue #1730⁴²⁴⁷ - Use after move of thread_init_data
- PR #1729⁴²⁴⁸ - Rewriting channel based on new gate component

⁴²²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1752>

⁴²²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1751>

⁴²²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1750>

⁴²²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1749>

⁴²³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1747>

⁴²³¹ <https://github.com/STELLAR-GROUP/hpx/pull/1746>

⁴²³² <https://github.com/STELLAR-GROUP/hpx/issues/1745>

⁴²³³ <https://github.com/STELLAR-GROUP/hpx/pull/1744>

⁴²³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1743>

⁴²³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1742>

⁴²³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1741>

⁴²³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1740>

⁴²³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1739>

⁴²³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1738>

⁴²⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1737>

⁴²⁴¹ <https://github.com/STELLAR-GROUP/hpx/pull/1736>

⁴²⁴² <https://github.com/STELLAR-GROUP/hpx/pull/1735>

⁴²⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/1734>

⁴²⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1733>

⁴²⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1732>

⁴²⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1731>

⁴²⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1730>

⁴²⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1729>

- PR #1728⁴²⁴⁹ - Fixing #1722
- PR #1727⁴²⁵⁰ - Fixing compile problems with apply_colocated
- PR #1726⁴²⁵¹ - Apex integration
- PR #1725⁴²⁵² - fixed test timeouts
- PR #1724⁴²⁵³ - Renaming vector
- Issue #1723⁴²⁵⁴ - Drop support for intel compilers and gcc 4.4. based standard libs
- Issue #1722⁴²⁵⁵ - Add support for detecting non-ready futures before serialization
- PR #1721⁴²⁵⁶ - Unifying parallel executors, initializing from launch policy
- PR #1720⁴²⁵⁷ - dropped superfluous typedef
- Issue #1718⁴²⁵⁸ - Windows 10 x64, VS 2015 - Unknown CMake command “add_hpx_pseudo_target”.
- PR #1717⁴²⁵⁹ - Timed executor traits for thread-executors
- PR #1716⁴²⁶⁰ - serialization of arrays didn’t work with non-pod types. fixed
- PR #1715⁴²⁶¹ - List serialization
- PR #1714⁴²⁶² - changing misspellings
- PR #1713⁴²⁶³ - Fixed distribution policy executors
- PR #1712⁴²⁶⁴ - Moving library detection to be executed after feature tests
- PR #1711⁴²⁶⁵ - Simplify parcel
- PR #1710⁴²⁶⁶ - Compile only tests
- PR #1709⁴²⁶⁷ - Implemented timed executors
- PR #1708⁴²⁶⁸ - Implement parallel::executor_traits for thread-executors
- PR #1707⁴²⁶⁹ - Various fixes to threads::executors to make custom schedulers work
- PR #1706⁴²⁷⁰ - Command line option –hpx:cores does not work as expected
- Issue #1705⁴²⁷¹ - command line option –hpx:cores does not work as expected

⁴²⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1728>

⁴²⁵⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1727>

⁴²⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/1726>

⁴²⁵² <https://github.com/STELLAR-GROUP/hpx/pull/1725>

⁴²⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/1724>

⁴²⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1723>

⁴²⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1722>

⁴²⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1721>

⁴²⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1720>

⁴²⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1718>

⁴²⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1717>

⁴²⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1716>

⁴²⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/1715>

⁴²⁶² <https://github.com/STELLAR-GROUP/hpx/pull/1714>

⁴²⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/1713>

⁴²⁶⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1712>

⁴²⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1711>

⁴²⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1710>

⁴²⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1709>

⁴²⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1708>

⁴²⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1707>

⁴²⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1706>

⁴²⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/1705>

- PR #1704⁴²⁷² - vector deserialization is speeded up a little
- PR #1703⁴²⁷³ - Fixing shared_mutes
- Issue #1702⁴²⁷⁴ - Shared_mutex does not compile with no_mutex cond_var
- PR #1701⁴²⁷⁵ - Add distribution_policy_executor
- PR #1700⁴²⁷⁶ - Executor parameters
- PR #1699⁴²⁷⁷ - Readers writer lock
- PR #1698⁴²⁷⁸ - Remove leftovers
- PR #1697⁴²⁷⁹ - Fixing held locks
- PR #1696⁴²⁸⁰ - Modified Scan Partitioner for Algorithms
- PR #1695⁴²⁸¹ - This thread executors
- PR #1694⁴²⁸² - Fixed #1688: Add timer counters for tfunc_total and exec_total
- PR #1693⁴²⁸³ - Fix #1691: is_executor template specification fails for inherited executors
- PR #1692⁴²⁸⁴ - Fixed #1662: Possible exception source in coalescing_message_handler
- Issue #1691⁴²⁸⁵ - is_executor template specification fails for inherited executors
- PR #1690⁴²⁸⁶ - added macro for non-intrusive serialization of classes without a default c-tor
- PR #1689⁴²⁸⁷ - Replace value_or_error with custom storage, unify future_data state
- Issue #1688⁴²⁸⁸ - Add timer counters for tfunc_total and exec_total
- PR #1687⁴²⁸⁹ - Fixed interval timer
- PR #1686⁴²⁹⁰ - Fixing cmake warnings about not existing pseudo target dependencies
- PR #1685⁴²⁹¹ - Converting partitioners to use bulk async execute
- PR #1683⁴²⁹² - Adds a tool for inspect that checks for character limits
- PR #1682⁴²⁹³ - Change project name to (uppercase) HPX
- PR #1681⁴²⁹⁴ - Counter shortnames

⁴²⁷² <https://github.com/STELLAR-GROUP/hpx/pull/1704>

⁴²⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/1703>

⁴²⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1702>

⁴²⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1701>

⁴²⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1700>

⁴²⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1699>

⁴²⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1698>

⁴²⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1697>

⁴²⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1696>

⁴²⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/1695>

⁴²⁸² <https://github.com/STELLAR-GROUP/hpx/pull/1694>

⁴²⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/1693>

⁴²⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1692>

⁴²⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1691>

⁴²⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1690>

⁴²⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1689>

⁴²⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1688>

⁴²⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1687>

⁴²⁹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1686>

⁴²⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1685>

⁴²⁹² <https://github.com/STELLAR-GROUP/hpx/pull/1683>

⁴²⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/1682>

⁴²⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1681>

- PR #1680⁴²⁹⁵ - Extended Non-intrusive Serialization to Ease Usage for Library Developers
- PR #1679⁴²⁹⁶ - Working on 1544: More executor changes
- PR #1678⁴²⁹⁷ - Transpose fixes
- PR #1677⁴²⁹⁸ - Improve Boost compatibility check
- PR #1676⁴²⁹⁹ - 1d stencil fix
- Issue #1675⁴³⁰⁰ - hpx project name is not HPX
- PR #1674⁴³⁰¹ - Fixing the MPI parcelport
- PR #1673⁴³⁰² - added move semantics to map/vector deserialization
- PR #1672⁴³⁰³ - Vs2015 await
- PR #1671⁴³⁰⁴ - Adapt transform for #1668
- PR #1670⁴³⁰⁵ - Started to work on #1668
- PR #1669⁴³⁰⁶ - Add this_thread_executors
- Issue #1667⁴³⁰⁷ - Apple build instructions in docs are out of date
- PR #1666⁴³⁰⁸ - Apex integration
- PR #1665⁴³⁰⁹ - Fixes an error with the whitespace check that showed the incorrect location of the error
- Issue #1664⁴³¹⁰ - Inspect tool found incorrect endline whitespace
- PR #1663⁴³¹¹ - Improve use of locks
- Issue #1662⁴³¹² - Possible exception source in coalescing_message_handler
- PR #1661⁴³¹³ - Added support for 128bit number serialization
- PR #1660⁴³¹⁴ - Serialization 128bits
- PR #1659⁴³¹⁵ - Implemented inner_product and adjacent_diff algos
- PR #1658⁴³¹⁶ - Add serialization for std::set (as there is for std::vector and std::map)
- PR #1657⁴³¹⁷ - Use of shared_ptr in io_service_pool changed to unique_ptr

⁴²⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1680>

⁴²⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1679>

⁴²⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1678>

⁴²⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1677>

⁴²⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1676>

⁴³⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1675>

⁴³⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/1674>

⁴³⁰² <https://github.com/STELLAR-GROUP/hpx/pull/1673>

⁴³⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/1672>

⁴³⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1671>

⁴³⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1670>

⁴³⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1669>

⁴³⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1667>

⁴³⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1666>

⁴³⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1665>

⁴³¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1664>

⁴³¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1663>

⁴³¹² <https://github.com/STELLAR-GROUP/hpx/issues/1662>

⁴³¹³ <https://github.com/STELLAR-GROUP/hpx/pull/1661>

⁴³¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1660>

⁴³¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1659>

⁴³¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1658>

⁴³¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1657>

- Issue #1656⁴³¹⁸ - 1d_stencil codes all have wrong factor
- PR #1654⁴³¹⁹ - When using runtime_mode_connect, find the correct localhost public ip address
- PR #1653⁴³²⁰ - Fixing 1617
- PR #1652⁴³²¹ - Remove traits::action_may_require_id_splitting
- PR #1651⁴³²² - Fixed performance counters related to AGAS cache timings
- PR #1650⁴³²³ - Remove leftovers of traits::type_size
- PR #1649⁴³²⁴ - Shorten target names on Windows to shorten used path names
- PR #1648⁴³²⁵ - Fixing problems introduced by merging #1623 for older compilers
- PR #1647⁴³²⁶ - Simplify running automatic builds on Windows
- Issue #1646⁴³²⁷ - Cache insert and update performance counters are broken
- Issue #1644⁴³²⁸ - Remove leftovers of traits::type_size
- Issue #1643⁴³²⁹ - Remove traits::action_may_require_id_splitting
- PR #1642⁴³³⁰ - Adds spell checker to the inspect tool for qbk and doxygen comments
- PR #1640⁴³³¹ - First step towards fixing 688
- PR #1639⁴³³² - Re-apply remaining changes from limit_dataflow_recursion branch
- PR #1638⁴³³³ - This fixes possible deadlock in the test ignore_while_locked_1485
- PR #1637⁴³³⁴ - Fixing hpx::wait_all() invoked with two vector<future<T>>
- PR #1636⁴³³⁵ - Partially re-apply changes from limit_dataflow_recursion branch
- PR #1635⁴³³⁶ - Adding missing test for #1572
- PR #1634⁴³³⁷ - Revert “Limit recursion-depth in dataflow to a configurable constant”
- PR #1633⁴³³⁸ - Add command line option to ignore batch environment
- PR #1631⁴³³⁹ - hpx::lcos::queue exhibits strange behavior
- PR #1630⁴³⁴⁰ - Fixed endline_whitespace_check.cpp to detect lines with only whitespace

⁴³¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1656>

⁴³¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1654>

⁴³²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1653>

⁴³²¹ <https://github.com/STELLAR-GROUP/hpx/pull/1652>

⁴³²² <https://github.com/STELLAR-GROUP/hpx/pull/1651>

⁴³²³ <https://github.com/STELLAR-GROUP/hpx/pull/1650>

⁴³²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1649>

⁴³²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1648>

⁴³²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1647>

⁴³²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1646>

⁴³²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1644>

⁴³²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1643>

⁴³³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1642>

⁴³³¹ <https://github.com/STELLAR-GROUP/hpx/pull/1640>

⁴³³² <https://github.com/STELLAR-GROUP/hpx/pull/1639>

⁴³³³ <https://github.com/STELLAR-GROUP/hpx/pull/1638>

⁴³³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1637>

⁴³³⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1636>

⁴³³⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1635>

⁴³³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1634>

⁴³³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1633>

⁴³³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1631>

⁴³⁴⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1630>

- Issue #1629⁴³⁴¹ - Inspect trailing whitespace checker problem
- PR #1628⁴³⁴² - Removed meaningless const qualifiers. Minor icpc fix.
- PR #1627⁴³⁴³ - Fixing the queue LCO and add example demonstrating its use
- PR #1626⁴³⁴⁴ - Deprecating get_gid(), add get_id() and get_unmanaged_id()
- PR #1625⁴³⁴⁵ - Allowing to specify whether to send credits along with message
- Issue #1624⁴³⁴⁶ - Lifetime issue
- Issue #1623⁴³⁴⁷ - hpx::wait_all() invoked with two vector<future<T>> fails
- PR #1622⁴³⁴⁸ - Executor partitioners
- PR #1621⁴³⁴⁹ - Clean up coroutines implementation
- Issue #1620⁴³⁵⁰ - Revert #1535
- PR #1619⁴³⁵¹ - Fix result type calculation for hpx::make_continuation
- PR #1618⁴³⁵² - Fixing RDTSC on Xeon/Phi
- Issue #1617⁴³⁵³ - hpx cmake not working when run as a subproject
- Issue #1616⁴³⁵⁴ - cmake problem resulting in RDTSC not working correctly for Xeon Phi creates very strange results for duration counters
- Issue #1615⁴³⁵⁵ - hpx::make_continuation requires input and output to be the same
- PR #1614⁴³⁵⁶ - Fixed remove copy test
- Issue #1613⁴³⁵⁷ - Dataflow causes stack overflow
- PR #1612⁴³⁵⁸ - Modified foreach partitioner to use bulk execute
- PR #1611⁴³⁵⁹ - Limit recursion-depth in dataflow to a configurable constant
- PR #1610⁴³⁶⁰ - Increase timeout for CircleCI
- PR #1609⁴³⁶¹ - Refactoring thread manager, mainly extracting thread pool
- PR #1608⁴³⁶² - Fixed running multiple localities without localities parameter
- PR #1607⁴³⁶³ - More algorithm fixes to adjacentfind

⁴³⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/1629>

⁴³⁴² <https://github.com/STELLAR-GROUP/hpx/pull/1628>

⁴³⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/1627>

⁴³⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1626>

⁴³⁴⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1625>

⁴³⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1624>

⁴³⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1623>

⁴³⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1622>

⁴³⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1621>

⁴³⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1620>

⁴³⁵¹ <https://github.com/STELLAR-GROUP/hpx/pull/1619>

⁴³⁵² <https://github.com/STELLAR-GROUP/hpx/pull/1618>

⁴³⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/1617>

⁴³⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1616>

⁴³⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1615>

⁴³⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1614>

⁴³⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1613>

⁴³⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1612>

⁴³⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1611>

⁴³⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1610>

⁴³⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/1609>

⁴³⁶² <https://github.com/STELLAR-GROUP/hpx/pull/1608>

⁴³⁶³ <https://github.com/STELLAR-GROUP/hpx/pull/1607>

- Issue #1606⁴³⁶⁴ - Running without localities parameter binds to bogus port range
- Issue #1605⁴³⁶⁵ - Too many serializations
- PR #1604⁴³⁶⁶ - Changes the HPX image into a hyperlink
- PR #1601⁴³⁶⁷ - Fixing problems with remove_copy algorithm tests
- PR #1600⁴³⁶⁸ - Actions with ids cleanup
- PR #1599⁴³⁶⁹ - Duplicate binding of global ids should fail
- PR #1598⁴³⁷⁰ - Fixing array access
- PR #1597⁴³⁷¹ - Improved the reliability of connecting/disconnecting localities
- Issue #1596⁴³⁷² - Duplicate id binding should fail
- PR #1595⁴³⁷³ - Fixing more cmake config constants
- PR #1594⁴³⁷⁴ - Fixing preprocessor constant used to enable C++11 chrono
- PR #1593⁴³⁷⁵ - Adding operator() for hpx::launch
- Issue #1592⁴³⁷⁶ - Error (typo) in the docs
- Issue #1590⁴³⁷⁷ - CMake fails when CMAKE_BINARY_DIR contains '+'.
- Issue #1589⁴³⁷⁸ - Disconnecting a locality results in segfault using heartbeat example
- PR #1588⁴³⁷⁹ - Fix doc string for config option HPX_WITH_EXAMPLES
- PR #1586⁴³⁸⁰ - Fixing 1493
- PR #1585⁴³⁸¹ - Additional Check for Inspect Tool to detect Endline Whitespace
- Issue #1584⁴³⁸² - Clean up coroutines implementation
- PR #1583⁴³⁸³ - Adding a check for end line whitespace
- PR #1582⁴³⁸⁴ - Attempt to fix assert firing after scheduling loop was exited
- PR #1581⁴³⁸⁵ - Fixed adjacentfind_binary test
- PR #1580⁴³⁸⁶ - Prevent some of the internal cmake lists from growing indefinitely

⁴³⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1606>

⁴³⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1605>

⁴³⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1604>

⁴³⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1601>

⁴³⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1600>

⁴³⁶⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1599>

⁴³⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1598>

⁴³⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/1597>

⁴³⁷² <https://github.com/STELLAR-GROUP/hpx/issues/1596>

⁴³⁷³ <https://github.com/STELLAR-GROUP/hpx/pull/1595>

⁴³⁷⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1594>

⁴³⁷⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1593>

⁴³⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1592>

⁴³⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1590>

⁴³⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1589>

⁴³⁷⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1588>

⁴³⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1586>

⁴³⁸¹ <https://github.com/STELLAR-GROUP/hpx/pull/1585>

⁴³⁸² <https://github.com/STELLAR-GROUP/hpx/issues/1584>

⁴³⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/1583>

⁴³⁸⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1582>

⁴³⁸⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1581>

⁴³⁸⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1580>

- PR #1579⁴³⁸⁷ - Removing type_size trait, replacing it with special archive type
- Issue #1578⁴³⁸⁸ - Remove demangle_helper
- PR #1577⁴³⁸⁹ - Get ptr problems
- Issue #1576⁴³⁹⁰ - Refactor async, dataflow, and future::then
- PR #1575⁴³⁹¹ - Fixing tests for parallel rotate
- PR #1574⁴³⁹² - Cleaning up schedulers
- PR #1573⁴³⁹³ - Fixing thread pool executor
- PR #1572⁴³⁹⁴ - Fixing number of configured localities
- PR #1571⁴³⁹⁵ - Reimplement decay
- PR #1570⁴³⁹⁶ - Refactoring async, apply, and dataflow APIs
- PR #1569⁴³⁹⁷ - Changed range for mach-o library lookup
- PR #1568⁴³⁹⁸ - Mark decltype support as required
- PR #1567⁴³⁹⁹ - Removed const from algorithms
- Issue #1566⁴⁴⁰⁰ - CMAKE Configuration Test Failures for clang 3.5 on debian
- PR #1565⁴⁴⁰¹ - Dylib support
- PR #1564⁴⁴⁰² - Converted partitioners and some algorithms to use executors
- PR #1563⁴⁴⁰³ - Fix several #includes for Boost.Preprocessor
- PR #1562⁴⁴⁰⁴ - Adding configuration option disabling/enabling all message handlers
- PR #1561⁴⁴⁰⁵ - Removed all occurrences of boost::move replacing it with std::move
- Issue #1560⁴⁴⁰⁶ - Leftover HPX_REGISTER_ACTION_DECLARATION_2
- PR #1558⁴⁴⁰⁷ - Revisit async/apply SFINAE conditions
- PR #1557⁴⁴⁰⁸ - Removing type_size trait, replacing it with special archive type
- PR #1556⁴⁴⁰⁹ - Executor algorithms

⁴³⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1579>

⁴³⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1578>

⁴³⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1577>

⁴³⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1576>

⁴³⁹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1575>

⁴³⁹² <https://github.com/STELLAR-GROUP/hpx/pull/1574>

⁴³⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/1573>

⁴³⁹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1572>

⁴³⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1571>

⁴³⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1570>

⁴³⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1569>

⁴³⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1568>

⁴³⁹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1567>

⁴⁴⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1566>

⁴⁴⁰¹ <https://github.com/STELLAR-GROUP/hpx/pull/1565>

⁴⁴⁰² <https://github.com/STELLAR-GROUP/hpx/pull/1564>

⁴⁴⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/1563>

⁴⁴⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1562>

⁴⁴⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1561>

⁴⁴⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1560>

⁴⁴⁰⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1558>

⁴⁴⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1557>

⁴⁴⁰⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1556>

- PR #1555⁴⁴¹⁰ - Remove the necessity to specify archive flags on the receiving end
- PR #1554⁴⁴¹¹ - Removing obsolete Boost.Serialization macros
- PR #1553⁴⁴¹² - Properly fix HPX_DEFINE_*_ACTION macros
- PR #1552⁴⁴¹³ - Fixed algorithms relying on copy_if implementation
- PR #1551⁴⁴¹⁴ - Pxfs - Modifying FindOrangeFS.cmake based on OrangeFS 2.9.X
- Issue #1550⁴⁴¹⁵ - Passing plain identifier inside HPX_DEFINE_PLAIN_ACTION_1
- PR #1549⁴⁴¹⁶ - Fixing intel14/libstdc++4.4
- PR #1548⁴⁴¹⁷ - Moving raw_ptr to detail namespace
- PR #1547⁴⁴¹⁸ - Adding support for executors to future.then
- PR #1546⁴⁴¹⁹ - Executor traits result types
- PR #1545⁴⁴²⁰ - Integrate executors with dataflow
- PR #1543⁴⁴²¹ - Fix potential zero-copy for primarynamespace::bulk_service_async et.al.
- PR #1542⁴⁴²² - Merging HPX0.9.10 into pxfs branch
- PR #1541⁴⁴²³ - Removed stale cmake tests, unused since the great cmake refactoring
- PR #1540⁴⁴²⁴ - Fix idle-rate on platforms without TSC
- PR #1539⁴⁴²⁵ - Reporting situation if zero-copy-serialization was performed by a parcel generated from a plain apply/async
- PR #1538⁴⁴²⁶ - Changed return type of bulk executors and added test
- Issue #1537⁴⁴²⁷ - Incorrect cpuid config tests
- PR #1536⁴⁴²⁸ - Changed return type of bulk executors and added test
- PR #1535⁴⁴²⁹ - Make sure promise::get_gid() can be called more than once
- PR #1534⁴⁴³⁰ - Fixed async_callback with bound callback
- PR #1533⁴⁴³¹ - Updated the link in the documentation to a publically- accessible URL
- PR #1532⁴⁴³² - Make sure sync primitives are not copyable nor movable

⁴⁴¹⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1555>

⁴⁴¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1554>

⁴⁴¹² <https://github.com/STELLAR-GROUP/hpx/pull/1553>

⁴⁴¹³ <https://github.com/STELLAR-GROUP/hpx/pull/1552>

⁴⁴¹⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1551>

⁴⁴¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1550>

⁴⁴¹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1549>

⁴⁴¹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1548>

⁴⁴¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1547>

⁴⁴¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1546>

⁴⁴²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1545>

⁴⁴²¹ <https://github.com/STELLAR-GROUP/hpx/pull/1543>

⁴⁴²² <https://github.com/STELLAR-GROUP/hpx/pull/1542>

⁴⁴²³ <https://github.com/STELLAR-GROUP/hpx/pull/1541>

⁴⁴²⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1540>

⁴⁴²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1539>

⁴⁴²⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1538>

⁴⁴²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1537>

⁴⁴²⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1536>

⁴⁴²⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1535>

⁴⁴³⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1534>

⁴⁴³¹ <https://github.com/STELLAR-GROUP/hpx/pull/1533>

⁴⁴³² <https://github.com/STELLAR-GROUP/hpx/pull/1532>

- PR #1531⁴⁴³³ - Fix unwrapped issue with future ranges of void type
- PR #1530⁴⁴³⁴ - Serialization complex
- Issue #1528⁴⁴³⁵ - Unwrapped issue with future<void>
- Issue #1527⁴⁴³⁶ - HPX does not build with Boost 1.58.0
- PR #1526⁴⁴³⁷ - Added support for boost.multi_array serialization
- PR #1525⁴⁴³⁸ - Properly handle deferred futures, fixes #1506
- PR #1524⁴⁴³⁹ - Making sure invalid action argument types generate clear error message
- Issue #1522⁴⁴⁴⁰ - Need serialization support for boost multi array
- Issue #1521⁴⁴⁴¹ - Remote async and zero-copy serialization optimizations don't play well together
- PR #1520⁴⁴⁴² - Fixing UB whil registering polymorphic classes for serialization
- PR #1519⁴⁴⁴³ - Making detail::condition_variable safe to use
- PR #1518⁴⁴⁴⁴ - Fix when_some bug missing indices in its result
- Issue #1517⁴⁴⁴⁵ - Typo may affect CMake build system tests
- PR #1516⁴⁴⁴⁶ - Fixing Posix context
- PR #1515⁴⁴⁴⁷ - Fixing Posix context
- PR #1514⁴⁴⁴⁸ - Correct problems with loading dynamic components
- PR #1513⁴⁴⁴⁹ - Fixing intel glibc4 4
- Issue #1508⁴⁴⁵⁰ - memory and papi counters do not work
- Issue #1507⁴⁴⁵¹ - Unrecognized Command Line Option Error causing exit status 0
- Issue #1506⁴⁴⁵² - Properly handle deferred futures
- PR #1505⁴⁴⁵³ - Adding #include - would not compile without this
- Issue #1502⁴⁴⁵⁴ - boost::filesystem::exists throws unexpected exception
- Issue #1501⁴⁴⁵⁵ - hwloc configuration options are wrong for MIC

⁴⁴³³ <https://github.com/STELLAR-GROUP/hpx/pull/1531>

⁴⁴³⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1530>

⁴⁴³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1528>

⁴⁴³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1527>

⁴⁴³⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1526>

⁴⁴³⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1525>

⁴⁴³⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1524>

⁴⁴⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1522>

⁴⁴⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/1521>

⁴⁴⁴² <https://github.com/STELLAR-GROUP/hpx/pull/1520>

⁴⁴⁴³ <https://github.com/STELLAR-GROUP/hpx/pull/1519>

⁴⁴⁴⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1518>

⁴⁴⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1517>

⁴⁴⁴⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1516>

⁴⁴⁴⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1515>

⁴⁴⁴⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1514>

⁴⁴⁴⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1513>

⁴⁴⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1508>

⁴⁴⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/1507>

⁴⁴⁵² <https://github.com/STELLAR-GROUP/hpx/issues/1506>

⁴⁴⁵³ <https://github.com/STELLAR-GROUP/hpx/pull/1505>

⁴⁴⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1502>

⁴⁴⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1501>

- PR #1504⁴⁴⁵⁶ - Making sure boost::filesystem::exists() does not throw
- PR #1500⁴⁴⁵⁷ - Exit application on --hpx:version/-v and --hpx:info
- PR #1498⁴⁴⁵⁸ - Extended task block
- PR #1497⁴⁴⁵⁹ - Unique ptr serialization
- PR #1496⁴⁴⁶⁰ - Unique ptr serialization (closed)
- PR #1495⁴⁴⁶¹ - Switching circleci build type to debug
- Issue #1494⁴⁴⁶² - --hpx:version/-v does not exit after printing version information
- Issue #1493⁴⁴⁶³ - add an hpx_ prefix to libraries and components to avoid name conflicts
- Issue #1492⁴⁴⁶⁴ - Define and ensure limitations for arguments to async/apply
- PR #1489⁴⁴⁶⁵ - Enable idle rate counter on demand
- PR #1488⁴⁴⁶⁶ - Made sure detail::condition_variable can be safely destroyed
- PR #1487⁴⁴⁶⁷ - Introduced default (main) template implementation for ignore_while_checking
- PR #1486⁴⁴⁶⁸ - Add HPX inspect tool
- Issue #1485⁴⁴⁶⁹ - ignore_while_locked doesn't support all Lockable types
- PR #1484⁴⁴⁷⁰ - Docker image generation
- PR #1483⁴⁴⁷¹ - Move external endian library into HPX
- PR #1482⁴⁴⁷² - Actions with integer type ids
- Issue #1481⁴⁴⁷³ - Sync primitives safe destruction
- Issue #1480⁴⁴⁷⁴ - Move external/boost/endian into hpx/util
- Issue #1478⁴⁴⁷⁵ - Boost inspect violations
- PR #1479⁴⁴⁷⁶ - Adds serialization for arrays; some further/minor fixes
- PR #1477⁴⁴⁷⁷ - Fixing problems with the Intel compiler using a GCC 4.4 std library
- PR #1476⁴⁴⁷⁸ - Adding hpx::lcos::latch and hpx::lcos::local::latch

⁴⁴⁵⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1504>

⁴⁴⁵⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1500>

⁴⁴⁵⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1498>

⁴⁴⁵⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1497>

⁴⁴⁶⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1496>

⁴⁴⁶¹ <https://github.com/STELLAR-GROUP/hpx/pull/1495>

⁴⁴⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1494>

⁴⁴⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/1493>

⁴⁴⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1492>

⁴⁴⁶⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1489>

⁴⁴⁶⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1488>

⁴⁴⁶⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1487>

⁴⁴⁶⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1486>

⁴⁴⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1485>

⁴⁴⁷⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1484>

⁴⁴⁷¹ <https://github.com/STELLAR-GROUP/hpx/pull/1483>

⁴⁴⁷² <https://github.com/STELLAR-GROUP/hpx/pull/1482>

⁴⁴⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/1481>

⁴⁴⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1480>

⁴⁴⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1478>

⁴⁴⁷⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1479>

⁴⁴⁷⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1477>

⁴⁴⁷⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1476>

- Issue #1475⁴⁴⁷⁹ - Boost inspect violations
- PR #1473⁴⁴⁸⁰ - Fixing action move tests
- Issue #1471⁴⁴⁸¹ - Sync primitives should not be movable
- PR #1470⁴⁴⁸² - Removing `hpx::util::polymorphic_factory`
- PR #1468⁴⁴⁸³ - Fixed container creation
- Issue #1467⁴⁴⁸⁴ - HPX application fail during finalization
- Issue #1466⁴⁴⁸⁵ - HPX doesn't pick up Torque's nodefile on SuperMIC
- Issue #1464⁴⁴⁸⁶ - HPX option for pre and post bootstrap performance counters
- PR #1463⁴⁴⁸⁷ - Replacing `async_colocated(id, ...)` with `async(colocated(id), ...)`
- PR #1462⁴⁴⁸⁸ - Consolidated task_region with N4411
- PR #1461⁴⁴⁸⁹ - Consolidate inconsistent CMake option names
- Issue #1460⁴⁴⁹⁰ - Which malloc is actually used? or at least which one is HPX built with
- Issue #1459⁴⁴⁹¹ - Make cmake configure step fail explicitly if compiler version is not supported
- Issue #1458⁴⁴⁹² - Update `parallel::task_region` with N4411
- PR #1456⁴⁴⁹³ - Consolidating `new_<>()`
- Issue #1455⁴⁴⁹⁴ - Replace `async_colocated(id, ...)` with `async(colocated(id), ...)`
- PR #1454⁴⁴⁹⁵ - Removed harmful std::moves from return statements
- PR #1453⁴⁴⁹⁶ - Use range-based for-loop instead of Boost.Foreach
- PR #1452⁴⁴⁹⁷ - C++ feature tests
- PR #1451⁴⁴⁹⁸ - When serializing, pass archive flags to traits::get_type_size
- Issue #1450⁴⁴⁹⁹ - traits::get_type_size needs archive flags to enable zero_copy optimizations
- Issue #1449⁴⁵⁰⁰ - “couldn't create performance counter” - AGAS
- Issue #1448⁴⁵⁰¹ - Replace distributing factories with `new_<T[]>(...)`

⁴⁴⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1475>

⁴⁴⁸⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1473>

⁴⁴⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/1471>

⁴⁴⁸² <https://github.com/STELLAR-GROUP/hpx/pull/1470>

⁴⁴⁸³ <https://github.com/STELLAR-GROUP/hpx/pull/1468>

⁴⁴⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1467>

⁴⁴⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1466>

⁴⁴⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1464>

⁴⁴⁸⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1463>

⁴⁴⁸⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1462>

⁴⁴⁸⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1461>

⁴⁴⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1460>

⁴⁴⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1459>

⁴⁴⁹² <https://github.com/STELLAR-GROUP/hpx/issues/1458>

⁴⁴⁹³ <https://github.com/STELLAR-GROUP/hpx/pull/1456>

⁴⁴⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1455>

⁴⁴⁹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1454>

⁴⁴⁹⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1453>

⁴⁴⁹⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1452>

⁴⁴⁹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1451>

⁴⁴⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1450>

⁴⁵⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1449>

⁴⁵⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/1448>

- PR #1447⁴⁵⁰² - Removing obsolete remote_object component
- PR #1446⁴⁵⁰³ - Hpx serialization
- PR #1445⁴⁵⁰⁴ - Replacing travis with circleci
- PR #1443⁴⁵⁰⁵ - Always stripping HPX command line arguments before executing start function
- PR #1442⁴⁵⁰⁶ - Adding –hpx:bind=none to disable thread affinities
- Issue #1439⁴⁵⁰⁷ - Libraries get linked in multiple times, RPATH is not properly set
- PR #1438⁴⁵⁰⁸ - Removed superfluous typedefs
- Issue #1437⁴⁵⁰⁹ - hpx::init() should strip HPX-related flags from argv
- Issue #1436⁴⁵¹⁰ - Add strong scaling option to htts
- PR #1435⁴⁵¹¹ - Adding async_cb, async_continue_cb, and async_colocated_cb
- PR #1434⁴⁵¹² - Added missing install rule, removed some dead CMake code
- PR #1433⁴⁵¹³ - Add GitExternal and SubProject cmake scripts from eyescale/cmake repo
- Issue #1432⁴⁵¹⁴ - Add command line flag to disable thread pinning
- PR #1431⁴⁵¹⁵ - Fix #1423
- Issue #1430⁴⁵¹⁶ - Inconsistent CMake option names
- Issue #1429⁴⁵¹⁷ - Configure setting HPX_HAVE_PARCELPORT_MPI is ignored
- PR #1428⁴⁵¹⁸ - Fixes #1419 (closed)
- PR #1427⁴⁵¹⁹ - Adding stencil_iterator and transform_iterator
- PR #1426⁴⁵²⁰ - Fixes #1419
- PR #1425⁴⁵²¹ - During serialization memory allocation should honour allocator chunk size
- Issue #1424⁴⁵²² - chunk allocation during serialization does not use memory pool/allocator chunk size
- Issue #1423⁴⁵²³ - Remove HPX_STD_UNIQUE_PTR
- Issue #1422⁴⁵²⁴ - hpx:threads=all allocates too many os threads

⁴⁵⁰² <https://github.com/STELLAR-GROUP/hpx/pull/1447>

⁴⁵⁰³ <https://github.com/STELLAR-GROUP/hpx/pull/1446>

⁴⁵⁰⁴ <https://github.com/STELLAR-GROUP/hpx/pull/1445>

⁴⁵⁰⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1443>

⁴⁵⁰⁶ <https://github.com/STELLAR-GROUP/hpx/pull/1442>

⁴⁵⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1439>

⁴⁵⁰⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1438>

⁴⁵⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1437>

⁴⁵¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1436>

⁴⁵¹¹ <https://github.com/STELLAR-GROUP/hpx/pull/1435>

⁴⁵¹² <https://github.com/STELLAR-GROUP/hpx/pull/1434>

⁴⁵¹³ <https://github.com/STELLAR-GROUP/hpx/pull/1433>

⁴⁵¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1432>

⁴⁵¹⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1431>

⁴⁵¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1430>

⁴⁵¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1429>

⁴⁵¹⁸ <https://github.com/STELLAR-GROUP/hpx/pull/1428>

⁴⁵¹⁹ <https://github.com/STELLAR-GROUP/hpx/pull/1427>

⁴⁵²⁰ <https://github.com/STELLAR-GROUP/hpx/pull/1426>

⁴⁵²¹ <https://github.com/STELLAR-GROUP/hpx/pull/1425>

⁴⁵²² <https://github.com/STELLAR-GROUP/hpx/issues/1424>

⁴⁵²³ <https://github.com/STELLAR-GROUP/hpx/issues/1423>

⁴⁵²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1422>

- PR #1420⁴⁵²⁵ - added .travis.yml
- Issue #1419⁴⁵²⁶ - Unify enums: `hpx::runtime::state` and `hpx::state`
- PR #1416⁴⁵²⁷ - Adding travis builder
- Issue #1414⁴⁵²⁸ - Correct directory for dispatch_gcc46.hpp iteration
- Issue #1410⁴⁵²⁹ - Set operation algorithms
- Issue #1389⁴⁵³⁰ - Parallel algorithms relying on scan partitioner break for small number of elements
- Issue #1325⁴⁵³¹ - Exceptions thrown during parcel handling are not handled correctly
- Issue #1315⁴⁵³² - Errors while running performance tests
- Issue #1309⁴⁵³³ - `hpx::vector` partitions are not easily extendable by applications
- PR #1300⁴⁵³⁴ - Added serialization/de-serialization to examples.tuplespace
- Issue #1251⁴⁵³⁵ - `hpx::threads::get_thread_count` doesn't consider pending threads
- Issue #1008⁴⁵³⁶ - Decrease in application performance overtime; occasional spikes of major slowdown
- Issue #1001⁴⁵³⁷ - Zero copy serialization raises assert
- Issue #721⁴⁵³⁸ - Make HPX usable for Xeon Phi
- Issue #524⁴⁵³⁹ - Extend scheduler to support threads which can't be stolen

2.10.16 HPX V0.9.10 (Mar 24, 2015)

General changes

This is the 12th official release of *HPX*. It coincides with the 7th anniversary of the first commit to our source code repository. Since then, we have seen over 12300 commits amounting to more than 220000 lines of C++ code.

The major focus of this release was to improve the reliability of large scale runs. We believe to have achieved this goal as we now can reliably run *HPX* applications on up to ~24k cores. We have also shown that *HPX* can be used with success for symmetric runs (applications using both, host cores and Intel Xeon/Phi coprocessors). This is a huge step forward in terms of the usability of *HPX*. The main focus of this work involved isolating the causes of the segmentation faults at start up and shut down. Many of these issues were discovered to be the result of the suspension of threads which hold locks.

A very important improvement introduced with this release is the refactoring of the code representing our parcel-port implementation. Parcel- ports can now be implemented by 3rd parties as independent plugins which are dynamically loaded at runtime (static linking of parcel-ports is also supported). This refactoring also includes a massive improvement of the performance of our existing parcel-ports. We were able to significantly reduce the networking latencies

⁴⁵²⁵ <https://github.com/STELLAR-GROUP/hpx/pull/1420>

⁴⁵²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1419>

⁴⁵²⁷ <https://github.com/STELLAR-GROUP/hpx/pull/1416>

⁴⁵²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1414>

4529 <https://github.com/STELLAR-GROUP/hpx/issues/1410>4530 <https://github.com/STELLAR-GROUP/hpx/issues/1389>4531 <https://github.com/STELLAR-GROUP/hpx/issues/1325>4532 <https://github.com/STELLAR-GROUP/hpx/issues/1315>4533 <https://github.com/STELLAR-GROUP/hpx/issues/1309>4534 <https://github.com/STELLAR-GROUP/hpx/pull/1300>4535 <https://github.com/STELLAR-GROUP/hpx/issues/1251>4536 <https://github.com/STELLAR-GROUP/hpx/issues/1008>4537 <https://github.com/STELLAR-GROUP/hpx/issues/1001>4538 <https://github.com/STELLAR-GROUP/hpx/issues/721>4539 <https://github.com/STELLAR-GROUP/hpx/issues/524>

and to improve the available networking bandwidth. Please note that in this release we disabled the ibverbs and ipc parcel ports as those have not been ported to the new plugin system yet (see [Issue #839⁴⁵⁴⁰](#)).

Another corner stone of this release is our work towards a complete implementation of `__cpp11_n4104` (Working Draft, Technical Specification for C++ Extensions for Parallelism). This document defines a set of parallel algorithms to be added to the C++ standard library. We now have implemented about 75% of all specified parallel algorithms (see [link hpx.manual.parallel.parallel_algorithms Parallel Algorithms] for more details). We also implemented some extensions to `__cpp11_n4104` allowing to invoke all of the algorithms asynchronously.

This release adds a first implementation of `hpx::vector` which is a distributed data structure closely aligned to the functionality of `std::vector`. The difference is that `hpx::vector` stores the data in partitions where the partitions can be distributed over different localities. We started to work on allowing to use the parallel algorithms with `hpx::vector`. At this point we have implemented only a few of the parallel algorithms to support distributed data structures (like `hpx::vector`) for testing purposes (see [Issue #1338⁴⁵⁴¹](#) for a documentation of our progress).

Breaking changes

With this release we put a lot of effort into changing the code base to be more compatible to C++11. These changes have caused the following issues for backward compatibility:

- Move to Variadics- All of the API now uses variadic templates. However, this change required to modify the argument sequence for some of the exiting API functions (`hpx::async_continue`, `hpx::apply_continue`, `hpx::when_each`, `hpx::wait_each`, synchronous invocation of actions).
- Changes to Macros- We also removed the macros `HPX_STD_FUNCTION` and `HPX_STD_TUPLE`. This shouldn't affect any user code as we replaced `HPX_STD_FUNCTION` with `hpx::util::function_nonser` which was the default expansion used for this macro. All `HPX` API functions which expect a `hpx::util::function_nonser` (or a `hpx::util::unique_function_nonser`) can now be transparently called with a compatible `std::function` instead. Similarly, `HPX_STD_TUPLE` was replaced by its default expansion as well: `hpx::util::tuple`.
- Changes to `hpx::unique_future`- `hpx::unique_future`, which was deprecated in the previous release for `hpx::future` is now completely removed from `HPX`. This completes the transition to a completely standards conforming implementation of `hpx::future`.
- Changes to Supported Compilers. Finally, in order to utilize more C++11 semantics, we have officially dropped support for GCC 4.4 and MSVC 2012. Please see our [Prerequisites](#) page for more details.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- [Issue #1402⁴⁵⁴²](#) - Internal shared_future serialization copies
- [Issue #1399⁴⁵⁴³](#) - Build takes unusually long time...
- [Issue #1398⁴⁵⁴⁴](#) - Tests using the scan partitioner are broken on at least gcc 4.7 and intel compiler
- [Issue #1397⁴⁵⁴⁵](#) - Completely remove `hpx::unique_future`
- [Issue #1396⁴⁵⁴⁶](#) - Parallel scan algorithms with different initial values

⁴⁵⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/839>

⁴⁵⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/1338>

⁴⁵⁴² <https://github.com/STELLAR-GROUP/hpx/issues/1402>

⁴⁵⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/1399>

⁴⁵⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1398>

⁴⁵⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1397>

⁴⁵⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1396>

- Issue #1395⁴⁵⁴⁷ - Race Condition - 1d_stencil_8 - SuperMIC
- Issue #1394⁴⁵⁴⁸ - “suspending thread while at least one lock is being held” - 1d_stencil_8 - SuperMIC
- Issue #1393⁴⁵⁴⁹ - SEGFAULT in 1d_stencil_8 on SuperMIC
- Issue #1392⁴⁵⁵⁰ - Fixing #1168
- Issue #1391⁴⁵⁵¹ - Parallel Algorithms for scan partitioner for small number of elements
- Issue #1387⁴⁵⁵² - Failure with more than 4 localities
- Issue #1386⁴⁵⁵³ - Dispatching unhandled exceptions to outer user code
- Issue #1385⁴⁵⁵⁴ - Adding Copy algorithms, fixing `parallel::copy_if`
- Issue #1384⁴⁵⁵⁵ - Fixing 1325
- Issue #1383⁴⁵⁵⁶ - Fixed #504: Refactor Dataflow LCO to work with futures, this removes the dataflow component as it is obsolete
- Issue #1382⁴⁵⁵⁷ - `is_sorted`, `is_sorted_until` and `is_partitioned` algorithms
- Issue #1381⁴⁵⁵⁸ - fix for CMake versions prior to 3.1
- Issue #1380⁴⁵⁵⁹ - resolved warning in CMake 3.1 and newer
- Issue #1379⁴⁵⁶⁰ - Compilation error with papi
- Issue #1378⁴⁵⁶¹ - Towards safer migration
- Issue #1377⁴⁵⁶² - HPXConfig.cmake should include `TCMALLOC_LIBRARY` and `TCMALLOC_INCLUDE_DIR`
- Issue #1376⁴⁵⁶³ - Warning on uninitialized member
- Issue #1375⁴⁵⁶⁴ - Fixing 1163
- Issue #1374⁴⁵⁶⁵ - Fixing the MSVC 12 release builder
- Issue #1373⁴⁵⁶⁶ - Modifying parallel search algorithm for zero length searches
- Issue #1372⁴⁵⁶⁷ - Modifying parallel search algorithm for zero length searches
- Issue #1371⁴⁵⁶⁸ - Avoid holding a lock during `agас::inref` while doing a credit split
- Issue #1370⁴⁵⁶⁹ - `--hpx:bind` throws unexpected error

⁴⁵⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1395>

⁴⁵⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1394>

⁴⁵⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1393>

⁴⁵⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1392>

⁴⁵⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/1391>

⁴⁵⁵² <https://github.com/STELLAR-GROUP/hpx/issues/1387>

⁴⁵⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/1386>

⁴⁵⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1385>

⁴⁵⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1384>

⁴⁵⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1383>

⁴⁵⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1382>

⁴⁵⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1381>

⁴⁵⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1380>

⁴⁵⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1379>

⁴⁵⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/1378>

⁴⁵⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1377>

⁴⁵⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/1376>

⁴⁵⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1375>

⁴⁵⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1374>

⁴⁵⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1373>

⁴⁵⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1372>

⁴⁵⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1371>

⁴⁵⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1370>

- Issue #1369⁴⁵⁷⁰ - Getting rid of (void) in loops
- Issue #1368⁴⁵⁷¹ - Variadic templates support for tuple
- Issue #1367⁴⁵⁷² - One last batch of variadic templates support
- Issue #1366⁴⁵⁷³ - Fixing symbolic namespace hang
- Issue #1365⁴⁵⁷⁴ - More held locks
- Issue #1364⁴⁵⁷⁵ - Add counters 1363
- Issue #1363⁴⁵⁷⁶ - Add thread overhead counters
- Issue #1362⁴⁵⁷⁷ - Std config removal
- Issue #1361⁴⁵⁷⁸ - Parcelport plugins
- Issue #1360⁴⁵⁷⁹ - Detuplify transfer_action
- Issue #1359⁴⁵⁸⁰ - Removed obsolete checks
- Issue #1358⁴⁵⁸¹ - Fixing 1352
- Issue #1357⁴⁵⁸² - Variadic templates support for runtime_support and components
- Issue #1356⁴⁵⁸³ - fixed coordinate test for intel13
- Issue #1355⁴⁵⁸⁴ - fixed coordinate.hpp
- Issue #1354⁴⁵⁸⁵ - Lexicographical Compare completed
- Issue #1353⁴⁵⁸⁶ - HPX should set Boost_ADDITIONAL_VERSIONS flags
- Issue #1352⁴⁵⁸⁷ - Error: Cannot find action “ in type registry: HPX(bad_action_code)
- Issue #1351⁴⁵⁸⁸ - Variadic templates support for applicers
- Issue #1350⁴⁵⁸⁹ - Actions simplification
- Issue #1349⁴⁵⁹⁰ - Variadic when and wait functions
- Issue #1348⁴⁵⁹¹ - Added hpx_init header to test files
- Issue #1347⁴⁵⁹² - Another batch of variadic templates support

⁴⁵⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1369>

⁴⁵⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/1368>

⁴⁵⁷² <https://github.com/STELLAR-GROUP/hpx/issues/1367>

⁴⁵⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/1366>

⁴⁵⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1365>

⁴⁵⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1364>

⁴⁵⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1363>

⁴⁵⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1362>

⁴⁵⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1361>

⁴⁵⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1360>

⁴⁵⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1359>

⁴⁵⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/1358>

⁴⁵⁸² <https://github.com/STELLAR-GROUP/hpx/issues/1357>

⁴⁵⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/1356>

⁴⁵⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1355>

⁴⁵⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1354>

⁴⁵⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1353>

⁴⁵⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1352>

⁴⁵⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1351>

⁴⁵⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1350>

⁴⁵⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1349>

⁴⁵⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1348>

⁴⁵⁹² <https://github.com/STELLAR-GROUP/hpx/issues/1347>

- Issue #1346⁴⁵⁹³ - Segmented copy
- Issue #1345⁴⁵⁹⁴ - Attempting to fix hangs during shutdown
- Issue #1344⁴⁵⁹⁵ - Std config removal
- Issue #1343⁴⁵⁹⁶ - Removing various distribution policies for hpx::vector
- Issue #1342⁴⁵⁹⁷ - Inclusive scan
- Issue #1341⁴⁵⁹⁸ - Exclusive scan
- Issue #1340⁴⁵⁹⁹ - Adding parallel::count for distributed data structures, adding tests
- Issue #1339⁴⁶⁰⁰ - Update argument order for transform_reduce
- Issue #1337⁴⁶⁰¹ - Fix dataflow to handle properly ranges of futures
- Issue #1336⁴⁶⁰² - dataflow needs to hold onto futures passed to it
- Issue #1335⁴⁶⁰³ - Fails to compile with msvc14
- Issue #1334⁴⁶⁰⁴ - Examples build problem
- Issue #1333⁴⁶⁰⁵ - Distributed transform reduce
- Issue #1332⁴⁶⁰⁶ - Variadic templates support for actions
- Issue #1331⁴⁶⁰⁷ - Some ambiguous calls of map::erase have been prevented by adding additional check in locality constructor.
- Issue #1330⁴⁶⁰⁸ - Defining Plain Actions does not work as described in the documentation
- Issue #1329⁴⁶⁰⁹ - Distributed vector cleanup
- Issue #1328⁴⁶¹⁰ - Sync docs and comments with code in hello_world example
- Issue #1327⁴⁶¹¹ - Typos in docs
- Issue #1326⁴⁶¹² - Documentation and code diverged in Fibonacci tutorial
- Issue #1325⁴⁶¹³ - Exceptions thrown during parcel handling are not handled correctly
- Issue #1324⁴⁶¹⁴ - fixed bandwidth calculation
- Issue #1323⁴⁶¹⁵ - mmap() failed to allocate thread stack due to insufficient resources

⁴⁵⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/1346>

⁴⁵⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1345>

⁴⁵⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1344>

⁴⁵⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1343>

⁴⁵⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1342>

⁴⁵⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1341>

⁴⁵⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1340>

⁴⁶⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1339>

⁴⁶⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/1337>

⁴⁶⁰² <https://github.com/STELLAR-GROUP/hpx/issues/1336>

⁴⁶⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/1335>

⁴⁶⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1334>

⁴⁶⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1333>

⁴⁶⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1332>

⁴⁶⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1331>

⁴⁶⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1330>

⁴⁶⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1329>

⁴⁶¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1328>

⁴⁶¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1327>

⁴⁶¹² <https://github.com/STELLAR-GROUP/hpx/issues/1326>

⁴⁶¹³ <https://github.com/STELLAR-GROUP/hpx/issues/1325>

⁴⁶¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1324>

⁴⁶¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1323>

- Issue #1322⁴⁶¹⁶ - HPX fails to build aa182cf
- Issue #1321⁴⁶¹⁷ - Limiting size of outgoing messages while coalescing parcels
- Issue #1320⁴⁶¹⁸ - passing a future with launch::deferred in remote function call causes hang
- Issue #1319⁴⁶¹⁹ - An exception when tries to specify number high priority threads with abp-priority
- Issue #1318⁴⁶²⁰ - Unable to run program with abp-priority and numa-sensitivity enabled
- Issue #1317⁴⁶²¹ - N4071 Search/Search_n finished, minor changes
- Issue #1316⁴⁶²² - Add config option to make -Ihpx.run_hpx_main!=1 the default
- Issue #1314⁴⁶²³ - Variadic support for async and apply
- Issue #1313⁴⁶²⁴ - Adjust when_any/some to the latest proposed interfaces
- Issue #1312⁴⁶²⁵ - Fixing #857: hpx::naming::locality leaks parcelport specific information into the public interface
- Issue #1311⁴⁶²⁶ - Distributed get'er/set'er_values for distributed vector
- Issue #1310⁴⁶²⁷ - Crashing in hpx::parcelset::policies::mpi::connection_handler::handle_messages() on Super-MIC
- Issue #1308⁴⁶²⁸ - Unable to execute an application with --hpx:threads
- Issue #1307⁴⁶²⁹ - merge_graph linking issue
- Issue #1306⁴⁶³⁰ - First batch of variadic templates support
- Issue #1305⁴⁶³¹ - Create a compiler wrapper
- Issue #1304⁴⁶³² - Provide a compiler wrapper for hpx
- Issue #1303⁴⁶³³ - Drop support for GCC44
- Issue #1302⁴⁶³⁴ - Fixing #1297
- Issue #1301⁴⁶³⁵ - Compilation error when tried to use boost range iterators with wait_all
- Issue #1298⁴⁶³⁶ - Distributed vector
- Issue #1297⁴⁶³⁷ - Unable to invoke component actions recursively

⁴⁶¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1322>

⁴⁶¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1321>

⁴⁶¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1320>

⁴⁶¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1319>

⁴⁶²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1318>

⁴⁶²¹ <https://github.com/STELLAR-GROUP/hpx/issues/1317>

⁴⁶²² <https://github.com/STELLAR-GROUP/hpx/issues/1316>

⁴⁶²³ <https://github.com/STELLAR-GROUP/hpx/issues/1314>

⁴⁶²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1313>

⁴⁶²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1312>

⁴⁶²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1311>

⁴⁶²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1310>

⁴⁶²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1308>

⁴⁶²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1307>

⁴⁶³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1306>

⁴⁶³¹ <https://github.com/STELLAR-GROUP/hpx/issues/1305>

⁴⁶³² <https://github.com/STELLAR-GROUP/hpx/issues/1304>

⁴⁶³³ <https://github.com/STELLAR-GROUP/hpx/issues/1303>

⁴⁶³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1302>

⁴⁶³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1301>

⁴⁶³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1298>

⁴⁶³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1297>

- Issue #1294⁴⁶³⁸ - HDF5 build error
- Issue #1275⁴⁶³⁹ - The parcelport implementation is non-optimal
- Issue #1267⁴⁶⁴⁰ - Added classes and unit tests for local_file, orangefs_file and pxfs_file
- Issue #1264⁴⁶⁴¹ - Error “assertion ‘!m_fun’ failed” randomly occurs when using TCP
- Issue #1254⁴⁶⁴² - thread binding seems to not work properly
- Issue #1220⁴⁶⁴³ - parallel::copy_if is broken
- Issue #1217⁴⁶⁴⁴ - Find a better way of fixing the issue patched by #1216
- Issue #1168⁴⁶⁴⁵ - Starting HPX on Cray machines using aprun isn’t working correctly
- Issue #1085⁴⁶⁴⁶ - Replace startup and shutdown barriers with broadcasts
- Issue #981⁴⁶⁴⁷ - With SLURM, –hpx:threads=8 should not be necessary
- Issue #857⁴⁶⁴⁸ - hpx::naming::locality leaks parcelport specific information into the public interface
- Issue #850⁴⁶⁴⁹ - “flush” not documented
- Issue #763⁴⁶⁵⁰ - Create buildbot instance that uses std::bind as HPX_STD_BIND
- Issue #680⁴⁶⁵¹ - Convert parcel ports into a plugin system
- Issue #582⁴⁶⁵² - Make exception thrown from HPX threads available from hpx::init
- Issue #504⁴⁶⁵³ - Refactor Dataflow LCO to work with futures
- Issue #196⁴⁶⁵⁴ - Don’t store copies of the locality network metadata in the gva table

2.10.17 HPX V0.9.9 (Oct 31, 2014, codename Spooky)

General changes

We have had over 1500 commits since the last release and we have closed over 200 tickets (bugs, feature requests, pull requests, etc.). These are by far the largest numbers of commits and resolved issues for any of the *HPX* releases so far. We are especially happy about the large number of people who contributed for the first time to *HPX*.

- We completed the transition from the older (non-conforming) implementation of hpx::future to the new and fully conforming version by removing the old code and by renaming the type hpx::unique_future to hpx::future. In order to maintain backwards compatibility with existing code which uses the type hpx::unique_future we support the configuration variable HPX_UNIQUE_FUTURE_ALIAS. If this variable is set to ON while running cmake it will additionally define a template alias for this type.

⁴⁶³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1294>

⁴⁶³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1275>

4640 <https://github.com/STELLAR-GROUP/hpx/issues/1267>4641 <https://github.com/STELLAR-GROUP/hpx/issues/1264>4642 <https://github.com/STELLAR-GROUP/hpx/issues/1254>4643 <https://github.com/STELLAR-GROUP/hpx/issues/1220>4644 <https://github.com/STELLAR-GROUP/hpx/issues/1217>4645 <https://github.com/STELLAR-GROUP/hpx/issues/1168>4646 <https://github.com/STELLAR-GROUP/hpx/issues/1085>4647 <https://github.com/STELLAR-GROUP/hpx/issues/981>4648 <https://github.com/STELLAR-GROUP/hpx/issues/857>4649 <https://github.com/STELLAR-GROUP/hpx/issues/850>4650 <https://github.com/STELLAR-GROUP/hpx/issues/763>4651 <https://github.com/STELLAR-GROUP/hpx/issues/680>4652 <https://github.com/STELLAR-GROUP/hpx/issues/582>4653 <https://github.com/STELLAR-GROUP/hpx/issues/504>4654 <https://github.com/STELLAR-GROUP/hpx/issues/196>

- We rewrote and significantly changed our build system. Please have a look at the new (now generated) documentation here: `hpx_build_system`. Please revisit your build scripts to adapt to the changes. The most notable changes are:
 - `HPX_NO_INSTALL` is no longer necessary.
 - For external builds, you need to set `HPX_DIR` instead of `HPX_ROOT` as described here: [Using HPX with CMake-based projects](#).
 - IDEs that support multiple configurations (Visual Studio and XCode) can now be used as intended. that means no build dir.
 - Building HPX statically (without dynamic libraries) is now supported (`-DHGX_STATIC_LINKING=On`).
 - Please note that many variables used to configure the build process have been renamed to unify the naming conventions (see the section [CMake variables used to configure HPX](#) for more information).
 - This also fixes a long list of issues, for more information see [Issue #1204⁴⁶⁵⁵](#).
- We started to implement various proposals to the C++ Standardization committee related to parallelism and concurrency, most notably [N4409⁴⁶⁵⁶](#) (Working Draft, Technical Specification for C++ Extensions for Parallelism), [N4411⁴⁶⁵⁷](#) (Task Region Rev. 3), and [N4313⁴⁶⁵⁸](#) (Working Draft, Technical Specification for C++ Extensions for Concurrency).
- We completely remodeled our automatic build system to run builds and unit tests on various systems and compilers. This allows us to find most bugs right as they were introduced and helps to maintain a high level of quality and compatibility. The newest build logs can be found at [HPX Buildbot Website⁴⁶⁵⁹](#).

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- Issue [#1296⁴⁶⁶⁰](#) - Rename `make_error_future` to `make_exceptional_future`, adjust to N4123
- Issue [#1295⁴⁶⁶¹](#) - building issue
- Issue [#1293⁴⁶⁶²](#) - Transpose example
- Issue [#1292⁴⁶⁶³](#) - Wrong `abs()` function used in example
- Issue [#1291⁴⁶⁶⁴](#) - non-synchronized shift operators have been removed
- Issue [#1290⁴⁶⁶⁵](#) - RDTSCP is defined as true for Xeon Phi build
- Issue [#1289⁴⁶⁶⁶](#) - Fixing 1288
- Issue [#1288⁴⁶⁶⁷](#) - Add new performance counters
- Issue [#1287⁴⁶⁶⁸](#) - Hierarchy scheduler broken performance counters

⁴⁶⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1204>

⁴⁶⁵⁶ <http://wg21.link/n4409>

⁴⁶⁵⁷ <http://wg21.link/n4411>

⁴⁶⁵⁸ <http://wg21.link/n4313>

⁴⁶⁵⁹ <http://rostam.cct.lsu.edu/>

⁴⁶⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1296>

⁴⁶⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/1295>

⁴⁶⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1293>

⁴⁶⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/1292>

⁴⁶⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1291>

⁴⁶⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1290>

⁴⁶⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1289>

⁴⁶⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1288>

⁴⁶⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1287>

- Issue #1286⁴⁶⁶⁹ - Algorithm cleanup
- Issue #1285⁴⁶⁷⁰ - Broken Links in Documentation
- Issue #1284⁴⁶⁷¹ - Uninitialized copy
- Issue #1283⁴⁶⁷² - missing boost::scoped_ptr includes
- Issue #1282⁴⁶⁷³ - Update documentation of build options for schedulers
- Issue #1281⁴⁶⁷⁴ - reset idle rate counter
- Issue #1280⁴⁶⁷⁵ - Bug when executing on Intel MIC
- Issue #1279⁴⁶⁷⁶ - Add improved when_all/wait_all
- Issue #1278⁴⁶⁷⁷ - Implement improved when_all/wait_all
- Issue #1277⁴⁶⁷⁸ - feature request: get access to argc argv and variables_map
- Issue #1276⁴⁶⁷⁹ - Remove merging map
- Issue #1274⁴⁶⁸⁰ - Weird (wrong) string code in papi.cpp
- Issue #1273⁴⁶⁸¹ - Sequential task execution policy
- Issue #1272⁴⁶⁸² - Avoid CMake name clash for Boost.Thread library
- Issue #1271⁴⁶⁸³ - Updates on HPX Test Units
- Issue #1270⁴⁶⁸⁴ - hpx/util/safe_lexical_cast.hpp is added
- Issue #1269⁴⁶⁸⁵ - Added default value for “LIB” cmake variable
- Issue #1268⁴⁶⁸⁶ - Memory Counters not working
- Issue #1266⁴⁶⁸⁷ - FindHPX.cmake is not installed
- Issue #1263⁴⁶⁸⁸ - apply_remote test takes too long
- Issue #1262⁴⁶⁸⁹ - Chrono cleanup
- Issue #1261⁴⁶⁹⁰ - Need make install for papi counters and this builds all the examples
- Issue #1260⁴⁶⁹¹ - Documentation of Stencil example claims

⁴⁶⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1286>

⁴⁶⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1285>

⁴⁶⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/1284>

⁴⁶⁷² <https://github.com/STELLAR-GROUP/hpx/issues/1283>

⁴⁶⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/1282>

⁴⁶⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1281>

⁴⁶⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1280>

⁴⁶⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1279>

⁴⁶⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1278>

⁴⁶⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1277>

4679 <https://github.com/STELLAR-GROUP/hpx/issues/1276>4680 <https://github.com/STELLAR-GROUP/hpx/issues/1274>4681 <https://github.com/STELLAR-GROUP/hpx/issues/1273>4682 <https://github.com/STELLAR-GROUP/hpx/issues/1272>4683 <https://github.com/STELLAR-GROUP/hpx/issues/1271>4684 <https://github.com/STELLAR-GROUP/hpx/issues/1270>4685 <https://github.com/STELLAR-GROUP/hpx/issues/1269>4686 <https://github.com/STELLAR-GROUP/hpx/issues/1268>4687 <https://github.com/STELLAR-GROUP/hpx/issues/1266>4688 <https://github.com/STELLAR-GROUP/hpx/issues/1263>4689 <https://github.com/STELLAR-GROUP/hpx/issues/1262>4690 <https://github.com/STELLAR-GROUP/hpx/issues/1261>4691 <https://github.com/STELLAR-GROUP/hpx/issues/1260>

- Issue #1259⁴⁶⁹² - Avoid double-linking Boost on Windows
- Issue #1257⁴⁶⁹³ - Adding additional parameter to create_thread
- Issue #1256⁴⁶⁹⁴ - added buildbot changes to release notes
- Issue #1255⁴⁶⁹⁵ - Cannot build MiniGhost
- Issue #1253⁴⁶⁹⁶ - hpx::thread defects
- Issue #1252⁴⁶⁹⁷ - HPX_PREFIX is too fragile
- Issue #1250⁴⁶⁹⁸ - switch_to_fiber_emulation does not work properly
- Issue #1249⁴⁶⁹⁹ - Documentation is generated under Release folder
- Issue #1248⁴⁷⁰⁰ - Fix usage of hpx_generic_coroutine_context and get tests passing on powerpc
- Issue #1247⁴⁷⁰¹ - Dynamic linking error
- Issue #1246⁴⁷⁰² - Make cpuid.cpp C++11 compliant
- Issue #1245⁴⁷⁰³ - HPX fails on startup (setting thread affinity mask)
- Issue #1244⁴⁷⁰⁴ - HPX_WITH_RDTSC configure test fails, but should succeed
- Issue #1243⁴⁷⁰⁵ - CTest dashboard info for CSCS CDash drop location
- Issue #1242⁴⁷⁰⁶ - Mac fixes
- Issue #1241⁴⁷⁰⁷ - Failure in Distributed with Boost 1.56
- Issue #1240⁴⁷⁰⁸ - fix a race condition in examples.diskperf
- Issue #1239⁴⁷⁰⁹ - fix wait_each in examples.diskperf
- Issue #1238⁴⁷¹⁰ - Fixed #1237: hpx::util::portable_binary_iarchive failed
- Issue #1237⁴⁷¹¹ - hpx::util::portable_binary_iarchive faileds
- Issue #1235⁴⁷¹² - Fixing clang warnings and errors
- Issue #1234⁴⁷¹³ - TCP runs fail: Transport endpoint is not connected
- Issue #1233⁴⁷¹⁴ - Making sure the correct number of threads is registered with AGAS

⁴⁶⁹² <https://github.com/STELLAR-GROUP/hpx/issues/1259>

⁴⁶⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/1257>

⁴⁶⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1256>

⁴⁶⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1255>

⁴⁶⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1253>

⁴⁶⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1252>

⁴⁶⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1250>

⁴⁶⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1249>

⁴⁷⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1248>

⁴⁷⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/1247>

⁴⁷⁰² <https://github.com/STELLAR-GROUP/hpx/issues/1246>

⁴⁷⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/1245>

⁴⁷⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1244>

⁴⁷⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1243>

⁴⁷⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1242>

⁴⁷⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1241>

⁴⁷⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1240>

⁴⁷⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1239>

⁴⁷¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1238>

⁴⁷¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1237>

⁴⁷¹² <https://github.com/STELLAR-GROUP/hpx/issues/1235>

⁴⁷¹³ <https://github.com/STELLAR-GROUP/hpx/issues/1234>

⁴⁷¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1233>

- Issue #1232⁴⁷¹⁵ - Fixing race in wait_xxx
- Issue #1231⁴⁷¹⁶ - Parallel minmax
- Issue #1230⁴⁷¹⁷ - Distributed run of 1d_stencil_8 uses less threads than spec. & sometimes gives errors
- Issue #1229⁴⁷¹⁸ - Unstable number of threads
- Issue #1228⁴⁷¹⁹ - HPX link error (cmake / MPI)
- Issue #1226⁴⁷²⁰ - Warning about struct/class thread_counters
- Issue #1225⁴⁷²¹ - Adding parallel::replace etc
- Issue #1224⁴⁷²² - Extending dataflow to pass through non-future arguments
- Issue #1223⁴⁷²³ - Remaining find algorithms implemented, N4071
- Issue #1222⁴⁷²⁴ - Merging all the changes
- Issue #1221⁴⁷²⁵ - No error output when using mpirun with hpx
- Issue #1219⁴⁷²⁶ - Adding new AGAS cache performance counters
- Issue #1216⁴⁷²⁷ - Fixing using futures (clients) as arguments to actions
- Issue #1215⁴⁷²⁸ - Error compiling simple component
- Issue #1214⁴⁷²⁹ - Stencil docs
- Issue #1213⁴⁷³⁰ - Using more than a few dozen MPI processes on SuperMike results in a seg fault before getting to hpx_main
- Issue #1212⁴⁷³¹ - Parallel rotate
- Issue #1211⁴⁷³² - Direct actions cause the future's shared_state to be leaked
- Issue #1210⁴⁷³³ - Refactored local::promise to be standard conformant
- Issue #1209⁴⁷³⁴ - Improve command line handling
- Issue #1208⁴⁷³⁵ - Adding parallel::reverse and parallel::reverse_copy
- Issue #1207⁴⁷³⁶ - Add copy_backward and move_backward
- Issue #1206⁴⁷³⁷ - N4071 additional algorithms implemented

⁴⁷¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1232>

⁴⁷¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1231>

⁴⁷¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1230>

⁴⁷¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1229>

⁴⁷¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1228>

⁴⁷²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1226>

⁴⁷²¹ <https://github.com/STELLAR-GROUP/hpx/issues/1225>

⁴⁷²² <https://github.com/STELLAR-GROUP/hpx/issues/1224>

⁴⁷²³ <https://github.com/STELLAR-GROUP/hpx/issues/1223>

⁴⁷²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1222>

⁴⁷²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1221>

⁴⁷²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1219>

⁴⁷²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1216>

⁴⁷²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1215>

⁴⁷²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1214>

⁴⁷³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1213>

⁴⁷³¹ <https://github.com/STELLAR-GROUP/hpx/issues/1212>

⁴⁷³² <https://github.com/STELLAR-GROUP/hpx/issues/1211>

⁴⁷³³ <https://github.com/STELLAR-GROUP/hpx/issues/1210>

⁴⁷³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1209>

⁴⁷³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1208>

⁴⁷³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1207>

⁴⁷³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1206>

- Issue #1204⁴⁷³⁸ - Cmake simplification and various other minor changes
- Issue #1203⁴⁷³⁹ - Implementing new launch policy for (local) async: `hpx::launch::fork`.
- Issue #1202⁴⁷⁴⁰ - Failed assertion in `connection_cache.hpp`
- Issue #1201⁴⁷⁴¹ - `pkg-config` doesn't add mpi link directories
- Issue #1200⁴⁷⁴² - Error when querying time performance counters
- Issue #1199⁴⁷⁴³ - library path is now configurable (again)
- Issue #1198⁴⁷⁴⁴ - Error when querying performance counters
- Issue #1197⁴⁷⁴⁵ - tests fail with intel compiler
- Issue #1196⁴⁷⁴⁶ - Silence several warnings
- Issue #1195⁴⁷⁴⁷ - Rephrase initializers to work with VC++ 2012
- Issue #1194⁴⁷⁴⁸ - Simplify parallel algorithms
- Issue #1193⁴⁷⁴⁹ - Adding `parallel::equal`
- Issue #1192⁴⁷⁵⁰ - HPX(`out_of_memory`) on including `<hpx/hpx.hpp>`
- Issue #1191⁴⁷⁵¹ - Fixing #1189
- Issue #1190⁴⁷⁵² - Chrono cleanup
- Issue #1189⁴⁷⁵³ - Deadlock .. somewhere? (probably serialization)
- Issue #1188⁴⁷⁵⁴ - Removed `future::get_status()`
- Issue #1186⁴⁷⁵⁵ - Fixed FindOpenCL to find current AMD APP SDK
- Issue #1184⁴⁷⁵⁶ - Tweaking future unwrapping
- Issue #1183⁴⁷⁵⁷ - Extended `parallel::reduce`
- Issue #1182⁴⁷⁵⁸ - `future::unwrap` hangs for `launch::deferred`
- Issue #1181⁴⁷⁵⁹ - Adding `all_of`, `any_of`, and `none_of` and corresponding documentation
- Issue #1180⁴⁷⁶⁰ - `hpx::cout` defect

⁴⁷³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1204>

⁴⁷³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1203>

⁴⁷⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1202>

⁴⁷⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/1201>

⁴⁷⁴² <https://github.com/STELLAR-GROUP/hpx/issues/1200>

⁴⁷⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/1199>

⁴⁷⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1198>

⁴⁷⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1197>

⁴⁷⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1196>

⁴⁷⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1195>

⁴⁷⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1194>

⁴⁷⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1193>

⁴⁷⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1192>

⁴⁷⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/1191>

⁴⁷⁵² <https://github.com/STELLAR-GROUP/hpx/issues/1190>

⁴⁷⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/1189>

⁴⁷⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1188>

⁴⁷⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1186>

⁴⁷⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1184>

⁴⁷⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1183>

⁴⁷⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1182>

⁴⁷⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1181>

⁴⁷⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1180>

- Issue #1179⁴⁷⁶¹ - hpx::async does not work for member function pointers when called on types with self-defined unary operator*
- Issue #1178⁴⁷⁶² - Implemented variadic hpx::util::zip_iterator
- Issue #1177⁴⁷⁶³ - MPI parcelport defect
- Issue #1176⁴⁷⁶⁴ - HPX_DEFINE_COMPONENT_CONST_ACTION_TPL does not have a 2-argument version
- Issue #1175⁴⁷⁶⁵ - Create util::zip_iterator working with util::tuple<>
- Issue #1174⁴⁷⁶⁶ - Error Building HPX on linux, root_certificate_authority.cpp
- Issue #1173⁴⁷⁶⁷ - hpx::cout output lost
- Issue #1172⁴⁷⁶⁸ - HPX build error with Clang 3.4.2
- Issue #1171⁴⁷⁶⁹ - CMAKE_INSTALL_PREFIX ignored
- Issue #1170⁴⁷⁷⁰ - Close hpx_benchmarks repository on Github
- Issue #1169⁴⁷⁷¹ - Buildbot emails have syntax error in url
- Issue #1167⁴⁷⁷² - Merge partial implementation of standards proposal N3960
- Issue #1166⁴⁷⁷³ - Fixed several compiler warnings
- Issue #1165⁴⁷⁷⁴ - cmake warns: “tests.regressions.actions” does not exist
- Issue #1164⁴⁷⁷⁵ - Want my own serialization of hpx::future
- Issue #1162⁴⁷⁷⁶ - Segfault in hello_world example
- Issue #1161⁴⁷⁷⁷ - Use HPX_ASSERT to aid the compiler
- Issue #1160⁴⁷⁷⁸ - Do not put -DNDEBUG into hpx_application.pc
- Issue #1159⁴⁷⁷⁹ - Support Clang 3.4.2
- Issue #1158⁴⁷⁸⁰ - Fixed #1157: Rename when_n/wait_n, add when_xxx_n/wait_xxx_n
- Issue #1157⁴⁷⁸¹ - Rename when_n/wait_n, add when_xxx_n/wait_xxx_n
- Issue #1156⁴⁷⁸² - Force inlining fails
- Issue #1155⁴⁷⁸³ - changed header of printout to be compatible with python csv module

⁴⁷⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/1179>

⁴⁷⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1178>

⁴⁷⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/1177>

⁴⁷⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1176>

⁴⁷⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1175>

⁴⁷⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1174>

⁴⁷⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1173>

⁴⁷⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1172>

⁴⁷⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1171>

⁴⁷⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1170>

⁴⁷⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/1169>

⁴⁷⁷² <https://github.com/STELLAR-GROUP/hpx/issues/1167>

⁴⁷⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/1166>

⁴⁷⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1165>

⁴⁷⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1164>

⁴⁷⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1162>

⁴⁷⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1161>

⁴⁷⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1160>

⁴⁷⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1159>

⁴⁷⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1158>

⁴⁷⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/1157>

⁴⁷⁸² <https://github.com/STELLAR-GROUP/hpx/issues/1156>

⁴⁷⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/1155>

- Issue #1154⁴⁷⁸⁴ - Fixing iostreams
- Issue #1153⁴⁷⁸⁵ - Standard manipulators (like std::endl) do not work with hpx::ostream
- Issue #1152⁴⁷⁸⁶ - Functions revamp
- Issue #1151⁴⁷⁸⁷ - Suppressing cmake 3.0 policy warning for CMP0026
- Issue #1150⁴⁷⁸⁸ - Client Serialization error
- Issue #1149⁴⁷⁸⁹ - Segfault on Stampede
- Issue #1148⁴⁷⁹⁰ - Refactoring mini-ghost
- Issue #1147⁴⁷⁹¹ - N3960 copy_if and copy_n implemented and tested
- Issue #1146⁴⁷⁹² - Stencil print
- Issue #1145⁴⁷⁹³ - N3960 hpx::parallel::copy implemented and tested
- Issue #1144⁴⁷⁹⁴ - OpenMP examples 1d_stencil do not build
- Issue #1143⁴⁷⁹⁵ - 1d_stencil OpenMP examples do not build
- Issue #1142⁴⁷⁹⁶ - Cannot build HPX with gcc 4.6 on OS X
- Issue #1140⁴⁷⁹⁷ - Fix OpenMP lookup, enable usage of config tests in external CMake projects.
- Issue #1139⁴⁷⁹⁸ - hpx/hpx/config/compiler_specific.hpp
- Issue #1138⁴⁷⁹⁹ - clean up pkg-config files
- Issue #1137⁴⁸⁰⁰ - Improvements to create binary packages
- Issue #1136⁴⁸⁰¹ - HPX_GCC_VERSION not defined on all compilers
- Issue #1135⁴⁸⁰² - Avoiding collision between winsock2.h and windows.h
- Issue #1134⁴⁸⁰³ - Making sure, that hpx::finalize can be called from any locality
- Issue #1133⁴⁸⁰⁴ - 1d stencil examples
- Issue #1131⁴⁸⁰⁵ - Refactor unique_function implementation
- Issue #1130⁴⁸⁰⁶ - Unique function

⁴⁷⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1154>

⁴⁷⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1153>

⁴⁷⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1152>

⁴⁷⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1151>

⁴⁷⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1150>

⁴⁷⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1149>

⁴⁷⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1148>

⁴⁷⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1147>

⁴⁷⁹² <https://github.com/STELLAR-GROUP/hpx/issues/1146>

⁴⁷⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/1145>

⁴⁷⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1144>

⁴⁷⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1143>

⁴⁷⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1142>

⁴⁷⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1140>

⁴⁷⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1139>

⁴⁷⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1138>

⁴⁸⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1137>

⁴⁸⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/1136>

⁴⁸⁰² <https://github.com/STELLAR-GROUP/hpx/issues/1135>

⁴⁸⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/1134>

⁴⁸⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1133>

⁴⁸⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1131>

⁴⁸⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1130>

- Issue #1129⁴⁸⁰⁷ - Some fixes to the Build system on OS X
- Issue #1128⁴⁸⁰⁸ - Action future args
- Issue #1127⁴⁸⁰⁹ - Executor causes segmentation fault
- Issue #1124⁴⁸¹⁰ - Adding new API functions: register_id_with_basename, unregister_id_with_basename, find_ids_from_basename; adding test
- Issue #1123⁴⁸¹¹ - Reduce nesting of try-catch construct in encode_parcels?
- Issue #1122⁴⁸¹² - Client base fixes
- Issue #1121⁴⁸¹³ - Update hpxrun.py.in
- Issue #1120⁴⁸¹⁴ - HTTS2 tests compile errors on v110 (VS2012)
- Issue #1119⁴⁸¹⁵ - Remove references to boost::atomic in accumulator example
- Issue #1118⁴⁸¹⁶ - Only build test thread_pool_executor_1114_test if HPX_SCHEDULER is set
- Issue #1117⁴⁸¹⁷ - local_queue_executor linker error on vc110
- Issue #1116⁴⁸¹⁸ - Disabled performance counter should give runtime errors, not invalid data
- Issue #1115⁴⁸¹⁹ - Compile error with Intel C++ 13.1
- Issue #1114⁴⁸²⁰ - Default constructed executor is not usable
- Issue #1113⁴⁸²¹ - Fast compilation of logging causes ABI incompatibilities between different NDEBUG values
- Issue #1112⁴⁸²² - Using thread_pool_executors causes segfault
- Issue #1111⁴⁸²³ - hpx::threads::get_thread_data always returns zero
- Issue #1110⁴⁸²⁴ - Remove unnecessary null pointer checks
- Issue #1109⁴⁸²⁵ - More tests adjustments
- Issue #1108⁴⁸²⁶ - Clarify build rules for “libboost_atomic-mt.so”?
- Issue #1107⁴⁸²⁷ - Remove unnecessary null pointer checks
- Issue #1106⁴⁸²⁸ - network_storage benchmark improvements, adding legends to plots and tidying layout
- Issue #1105⁴⁸²⁹ - Add more plot outputs and improve instructions doc

4807 <https://github.com/STELLAR-GROUP/hpx/issues/1129>

4808 <https://github.com/STELLAR-GROUP/hpx/issues/1128>

4809 <https://github.com/STELLAR-GROUP/hpx/issues/1127>

4810 <https://github.com/STELLAR-GROUP/hpx/issues/1124>

4811 <https://github.com/STELLAR-GROUP/hpx/issues/1123>

4812 <https://github.com/STELLAR-GROUP/hpx/issues/1122>

4813 <https://github.com/STELLAR-GROUP/hpx/issues/1121>

4814 <https://github.com/STELLAR-GROUP/hpx/issues/1120>

4815 <https://github.com/STELLAR-GROUP/hpx/issues/1119>

4816 <https://github.com/STELLAR-GROUP/hpx/issues/1118>

4817 <https://github.com/STELLAR-GROUP/hpx/issues/1117>

4818 <https://github.com/STELLAR-GROUP/hpx/issues/1116>

4819 <https://github.com/STELLAR-GROUP/hpx/issues/1115>

4820 <https://github.com/STELLAR-GROUP/hpx/issues/1114>

4821 <https://github.com/STELLAR-GROUP/hpx/issues/1113>

4822 <https://github.com/STELLAR-GROUP/hpx/issues/1112>

4823 <https://github.com/STELLAR-GROUP/hpx/issues/1111>

4824 <https://github.com/STELLAR-GROUP/hpx/issues/1110>

4825 <https://github.com/STELLAR-GROUP/hpx/issues/1109>

4826 <https://github.com/STELLAR-GROUP/hpx/issues/1108>

4827 <https://github.com/STELLAR-GROUP/hpx/issues/1107>

4828 <https://github.com/STELLAR-GROUP/hpx/issues/1106>

4829 <https://github.com/STELLAR-GROUP/hpx/issues/1105>

- Issue #1104⁴⁸³⁰ - Complete quoting for parameters of some CMake commands
- Issue #1103⁴⁸³¹ - Work on test/scripts
- Issue #1102⁴⁸³² - Changed minimum requirement of window install to 2012
- Issue #1101⁴⁸³³ - Changed minimum requirement of window install to 2012
- Issue #1100⁴⁸³⁴ - Changed readme to no longer specify using MSVC 2010 compiler
- Issue #1099⁴⁸³⁵ - Error returning futures from component actions
- Issue #1098⁴⁸³⁶ - Improve storage test
- Issue #1097⁴⁸³⁷ - data_actions quickstart example calls missing function decorate_action of data_get_action
- Issue #1096⁴⁸³⁸ - MPI parcelport broken with new zero copy optimization
- Issue #1095⁴⁸³⁹ - Warning C4005: _WIN32_WINNT: Macro redefinition
- Issue #1094⁴⁸⁴⁰ - Syntax error for -DHPX_UNIQUE_FUTURE_ALIAS in master
- Issue #1093⁴⁸⁴¹ - Syntax error for -DHPX_UNIQUE_FUTURE_ALIAS
- Issue #1092⁴⁸⁴² - Rename unique_future<> back to future<>
- Issue #1091⁴⁸⁴³ - Inconsistent error message
- Issue #1090⁴⁸⁴⁴ - On windows 8.1 the examples crashed if using more than one os thread
- Issue #1089⁴⁸⁴⁵ - Components should be allowed to have their own executor
- Issue #1088⁴⁸⁴⁶ - Add possibility to select a network interface for the ibverbs parcelport
- Issue #1087⁴⁸⁴⁷ - ibverbs and ipc parcelport uses zero copy optimization
- Issue #1083⁴⁸⁴⁸ - Make shell examples copyable in docs
- Issue #1082⁴⁸⁴⁹ - Implement proper termination detection during shutdown
- Issue #1081⁴⁸⁵⁰ - Implement thread_specific_ptr for hpx::threads
- Issue #1072⁴⁸⁵¹ - make install not working properly
- Issue #1070⁴⁸⁵² - Complete quoting for parameters of some CMake commands

⁴⁸³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1104>

⁴⁸³¹ <https://github.com/STELLAR-GROUP/hpx/issues/1103>

⁴⁸³² <https://github.com/STELLAR-GROUP/hpx/issues/1102>

⁴⁸³³ <https://github.com/STELLAR-GROUP/hpx/issues/1101>

⁴⁸³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1100>

⁴⁸³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1099>

⁴⁸³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1098>

⁴⁸³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1097>

⁴⁸³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1096>

⁴⁸³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1095>

⁴⁸⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1094>

⁴⁸⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/1093>

⁴⁸⁴² <https://github.com/STELLAR-GROUP/hpx/issues/1092>

⁴⁸⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/1091>

⁴⁸⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1090>

⁴⁸⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1089>

⁴⁸⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1088>

⁴⁸⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1087>

⁴⁸⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1083>

⁴⁸⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1082>

⁴⁸⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1081>

⁴⁸⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/1072>

⁴⁸⁵² <https://github.com/STELLAR-GROUP/hpx/issues/1070>

- Issue #1059⁴⁸⁵³ - Fix more unused variable warnings
- Issue #1051⁴⁸⁵⁴ - Implement when_each
- Issue #973⁴⁸⁵⁵ - Would like option to report hwloc bindings
- Issue #970⁴⁸⁵⁶ - Bad flags for Fortran compiler
- Issue #941⁴⁸⁵⁷ - Create a proper user level context switching class for BG/Q
- Issue #935⁴⁸⁵⁸ - Build error with gcc 4.6 and Boost 1.54.0 on hpx trunk and 0.9.6
- Issue #934⁴⁸⁵⁹ - Want to build HPX without dynamic libraries
- Issue #927⁴⁸⁶⁰ - Make hpx/lcos/reduce.hpp accept futures of id_type
- Issue #926⁴⁸⁶¹ - All unit tests that are run with more than one thread with CTest/hpx_run_test should configure hpx.os_threads
- Issue #925⁴⁸⁶² - regression_dataflow_791 needs to be brought in line with HPX standards
- Issue #899⁴⁸⁶³ - Fix race conditions in regression tests
- Issue #879⁴⁸⁶⁴ - Hung test leads to cascading test failure; make tests should support the MPI parcelport
- Issue #865⁴⁸⁶⁵ - future<T> and friends shall work for movable only Ts
- Issue #847⁴⁸⁶⁶ - Dynamic libraries are not installed on OS X
- Issue #816⁴⁸⁶⁷ - First Program tutorial pull request
- Issue #799⁴⁸⁶⁸ - Wrap lexical_cast to avoid exceptions
- Issue #720⁴⁸⁶⁹ - broken configuration when using ccmake on Ubuntu
- Issue #622⁴⁸⁷⁰ --hpx:hpx and --hpx:debug-hpx-log is nonsensical
- Issue #525⁴⁸⁷¹ - Extend barrier LCO test to run in distributed
- Issue #515⁴⁸⁷² - Multi-destination version of hpx::apply is broken
- Issue #509⁴⁸⁷³ - Push Boost.Atomic changes upstream
- Issue #503⁴⁸⁷⁴ - Running HPX applications on Windows should not require setting %PATH%
- Issue #461⁴⁸⁷⁵ - Add a compilation sanity test

⁴⁸⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/1059>

⁴⁸⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1051>

⁴⁸⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/973>

⁴⁸⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/970>

⁴⁸⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/941>

⁴⁸⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/935>

⁴⁸⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/934>

⁴⁸⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/927>

⁴⁸⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/926>

⁴⁸⁶² <https://github.com/STELLAR-GROUP/hpx/issues/925>

⁴⁸⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/899>

⁴⁸⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/879>

⁴⁸⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/865>

⁴⁸⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/847>

⁴⁸⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/816>

⁴⁸⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/799>

⁴⁸⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/720>

⁴⁸⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/622>

⁴⁸⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/525>

⁴⁸⁷² <https://github.com/STELLAR-GROUP/hpx/issues/515>

⁴⁸⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/509>

⁴⁸⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/503>

⁴⁸⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/461>

- Issue #456⁴⁸⁷⁶ - hpx_run_tests.py should log output from tests that timeout
- Issue #454⁴⁸⁷⁷ - Investigate threadmanager performance
- Issue #345⁴⁸⁷⁸ - Add more versatile environmental/cmake variable support to hpx_find_* CMake macros
- Issue #209⁴⁸⁷⁹ - Support multiple configurations in generated build files
- Issue #190⁴⁸⁸⁰ - hpx::cout should be a std::ostream
- Issue #189⁴⁸⁸¹ - iostreams component should use startup/shutdown functions
- Issue #183⁴⁸⁸² - Use Boost.ICL for correctness in AGAS
- Issue #44⁴⁸⁸³ - Implement real futures

2.10.18 HPX V0.9.8 (Mar 24, 2014)

We have had over 800 commits since the last release and we have closed over 65 tickets (bugs, feature requests, etc.).

With the changes below, *HPX* is once again leading the charge of a whole new era of computation. By intrinsically breaking down and synchronizing the work to be done, *HPX* insures that application developers will no longer have to fret about where a segment of code executes. That allows coders to focus their time and energy to understanding the data dependencies of their algorithms and thereby the core obstacles to an efficient code. Here are some of the advantages of using *HPX*:

- *HPX* is solidly rooted in a sophisticated theoretical execution model – ParalleX
- *HPX* exposes an API fully conforming to the C++11 and the draft C++14 standards, extended and applied to distributed computing. Everything programmers know about the concurrency primitives of the standard C++ library is still valid in the context of *HPX*.
- It provides a competitive, high performance implementation of modern, future-proof ideas which gives an smooth migration path from today's mainstream techniques
- There is no need for the programmer to worry about lower level parallelization paradigms like threads or message passing; no need to understand pthreads, MPI, OpenMP, or Windows threads, etc.
- There is no need to think about different types of parallelism such as tasks, pipelines, or fork-join, task or data parallelism.
- The same source of your program compiles and runs on Linux, BlueGene/Q, Mac OS X, Windows, and Android.
- The same code runs on shared memory multi-core systems and supercomputers, on handheld devices and Intel® Xeon Phi™ accelerators, or a heterogeneous mix of those.

⁴⁸⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/456>

⁴⁸⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/454>

⁴⁸⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/345>

⁴⁸⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/209>

4880 <https://github.com/STELLAR-GROUP/hpx/issues/190>4881 <https://github.com/STELLAR-GROUP/hpx/issues/189>4882 <https://github.com/STELLAR-GROUP/hpx/issues/183>4883 <https://github.com/STELLAR-GROUP/hpx/issues/44>

General changes

- A major API breaking change for this release was introduced by implementing `hpx::future` and `hpx::shared_future` fully in conformance with the C++11 Standard⁴⁸⁸⁴. While `hpx::shared_future` is new and will not create any compatibility problems, we revised the interface and implementation of the existing `hpx::future`. For more details please see the mailing list archive⁴⁸⁸⁵. To avoid any incompatibilities for existing code we named the type which implements the `std::future` interface as `hpx::unique_future`. For the next release this will be renamed to `hpx::future`, making it full conforming to C++11 Standard⁴⁸⁸⁶.
- A large part of the code base of HPX has been refactored and partially re-implemented. The main changes were related to
 - The threading subsystem: these changes significantly reduce the amount of overheads caused by the schedulers, improve the modularity of the code base, and extend the variety of available scheduling algorithms.
 - The parcel subsystem: these changes improve the performance of the HPX networking layer, modularize the structure of the parcelports, and simplify the creation of new parcelports for other underlying networking libraries.
 - The API subsystem: these changes improved the conformance of the API to C++11 Standard, extend and unify the available API functionality, and decrease the overheads created by various elements of the API.
 - The robustness of the component loading subsystem has been improved significantly, allowing to more portably and more reliably register the components needed by an application at startup. This additionally speeds up general application initialization.
- We added new API functionality like `hpx::migrate` and `hpx::copy_component` which are the basic building blocks necessary for implementing higher level abstractions for system-wide load balancing, runtime-adaptive resource management, and object-oriented checkpointing and state-management.
- We removed the use of C++11 move emulation (using Boost.Move), replacing it with C++11 rvalue references. This is the first step towards using more and more native C++11 facilities which we plan to introduce in the future.
- We improved the reference counting scheme used by HPX which helps managing distributed objects and memory. This improves the overall stability of HPX and further simplifies writing real world applications.
- The minimal Boost version required to use HPX is now V1.49.0.
- This release coincides with the first release of HPXPI (V0.1.0), the first implementation of the XPI specification⁴⁸⁸⁷.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- Issue #1086⁴⁸⁸⁸ - Expose internal boost::shared_array to allow user management of array lifetime
- Issue #1083⁴⁸⁸⁹ - Make shell examples copyable in docs
- Issue #1080⁴⁸⁹⁰ - /threads{locality#*/total}/count/cumulative broken

⁴⁸⁸⁴ <http://www.open-std.org/jtc1/sc22/wg21>

⁴⁸⁸⁵ <http://mail.cct.lsu.edu/pipermail/hpx-users/2014-January/000141.html>

⁴⁸⁸⁶ <http://www.open-std.org/jtc1/sc22/wg21>

⁴⁸⁸⁷ <https://github.com/STELLAR-GROUP/hpxpi/blob/master/spec.pdf?raw=true>

⁴⁸⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1086>

⁴⁸⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1083>

⁴⁸⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1080>

- Issue #1079⁴⁸⁹¹ - Build problems on OS X
- Issue #1078⁴⁸⁹² - Improve robustness of component loading
- Issue #1077⁴⁸⁹³ - Fix a missing enum definition for ‘take’ mode
- Issue #1076⁴⁸⁹⁴ - Merge Jb master
- Issue #1075⁴⁸⁹⁵ - Unknown CMake command “add_hpx_pseudo_target”
- Issue #1074⁴⁸⁹⁶ - Implement apply_continue_callback and apply_colocated_callback
- Issue #1073⁴⁸⁹⁷ - The new apply_colocated and async_colocated functions lead to automatic registered functions
- Issue #1071⁴⁸⁹⁸ - Remove deferred_packaged_task
- Issue #1069⁴⁸⁹⁹ - serialize_buffer with allocator fails at destruction
- Issue #1068⁴⁹⁰⁰ - Coroutine include and forward declarations missing
- Issue #1067⁴⁹⁰¹ - Add allocator support to util::serialize_buffer
- Issue #1066⁴⁹⁰² - Allow for MPI_Init being called before HPX launches
- Issue #1065⁴⁹⁰³ - AGAS cache isn’t used/populated on worker localities
- Issue #1064⁴⁹⁰⁴ - Reorder includes to ensure ws2 includes early
- Issue #1063⁴⁹⁰⁵ - Add hpx::runtime::suspend and hpx::runtime::resume
- Issue #1062⁴⁹⁰⁶ - Fix async_continue to properly handle return types
- Issue #1061⁴⁹⁰⁷ - Implement async_colocated and apply_colocated
- Issue #1060⁴⁹⁰⁸ - Implement minimal component migration
- Issue #1058⁴⁹⁰⁹ - Remove HPX_UTIL_TUPLE from code base
- Issue #1057⁴⁹¹⁰ - Add performance counters for threading subsystem
- Issue #1055⁴⁹¹¹ - Thread allocation uses two memory pools
- Issue #1053⁴⁹¹² - Work stealing flawed
- Issue #1052⁴⁹¹³ - Fix a number of warnings

⁴⁸⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1079>

⁴⁸⁹² <https://github.com/STELLAR-GROUP/hpx/issues/1078>

⁴⁸⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/1077>

⁴⁸⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1076>

⁴⁸⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1075>

⁴⁸⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1074>

⁴⁸⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1073>

⁴⁸⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1071>

⁴⁸⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1069>

⁴⁹⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1068>

⁴⁹⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/1067>

⁴⁹⁰² <https://github.com/STELLAR-GROUP/hpx/issues/1066>

⁴⁹⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/1065>

⁴⁹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1064>

⁴⁹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1063>

⁴⁹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1062>

⁴⁹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1061>

⁴⁹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1060>

⁴⁹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1058>

⁴⁹¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1057>

⁴⁹¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/1055>

⁴⁹¹² <https://github.com/STELLAR-GROUP/hpx/issues/1053>

⁴⁹¹³ <https://github.com/STELLAR-GROUP/hpx/issues/1052>

- Issue #1049⁴⁹¹⁴ - Fixes for TLS on OSX and more reliable test running
- Issue #1048⁴⁹¹⁵ - Fixing after 588 hang
- Issue #1047⁴⁹¹⁶ - Use port ‘0’ for networking when using one locality
- Issue #1046⁴⁹¹⁷ - composable_guard test is broken when having more than one thread
- Issue #1045⁴⁹¹⁸ - Security missing headers
- Issue #1044⁴⁹¹⁹ - Native TLS on FreeBSD via __thread
- Issue #1043⁴⁹²⁰ - async et.al. compute the wrong result type
- Issue #1042⁴⁹²¹ - async et.al. implicitly unwrap reference_wrappers
- Issue #1041⁴⁹²² - Remove redundant costly Kleene stars from regex searches
- Issue #1040⁴⁹²³ - CMake script regex match patterns has unnecessary kleenes
- Issue #1039⁴⁹²⁴ - Remove use of Boost.Move and replace with std::move and real rvalue refs
- Issue #1038⁴⁹²⁵ - Bump minimal required Boost to 1.49.0
- Issue #1037⁴⁹²⁶ - Implicit unwrapping of futures in async broken
- Issue #1036⁴⁹²⁷ - Scheduler hangs when user code attempts to “block” OS-threads
- Issue #1035⁴⁹²⁸ - Idle-rate counter always reports 100% idle rate
- Issue #1034⁴⁹²⁹ - Symbolic name registration causes application hangs
- Issue #1033⁴⁹³⁰ - Application options read in from an options file generate an error message
- Issue #1032⁴⁹³¹ - hpx::id_type local reference counting is wrong
- Issue #1031⁴⁹³² - Negative entry in reference count table
- Issue #1030⁴⁹³³ - Implement condition_variable
- Issue #1029⁴⁹³⁴ - Deadlock in thread scheduling subsystem
- Issue #1028⁴⁹³⁵ - HPX-thread cumulative count performance counters report incorrect value
- Issue #1027⁴⁹³⁶ - Expose hpx::thread_interrupted error code as a separate exception type

⁴⁹¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1049>

⁴⁹¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1048>

⁴⁹¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1047>

⁴⁹¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1046>

⁴⁹¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1045>

⁴⁹¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1044>

⁴⁹²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1043>

⁴⁹²¹ <https://github.com/STELLAR-GROUP/hpx/issues/1042>

⁴⁹²² <https://github.com/STELLAR-GROUP/hpx/issues/1041>

⁴⁹²³ <https://github.com/STELLAR-GROUP/hpx/issues/1040>

⁴⁹²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1039>

⁴⁹²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1038>

⁴⁹²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1037>

⁴⁹²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1036>

⁴⁹²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1035>

⁴⁹²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1034>

⁴⁹³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1033>

⁴⁹³¹ <https://github.com/STELLAR-GROUP/hpx/issues/1032>

⁴⁹³² <https://github.com/STELLAR-GROUP/hpx/issues/1031>

⁴⁹³³ <https://github.com/STELLAR-GROUP/hpx/issues/1030>

⁴⁹³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1029>

⁴⁹³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1028>

⁴⁹³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1027>

- Issue #1026⁴⁹³⁷ - Exceptions thrown in asynchronous calls can be lost if the value of the future is never queried
- Issue #1025⁴⁹³⁸ - `future::wait_for/wait_until` do not remove callback
- Issue #1024⁴⁹³⁹ - Remove dependence to boost assert and create hpx assert
- Issue #1023⁴⁹⁴⁰ - Segfaults with tcmalloc
- Issue #1022⁴⁹⁴¹ - prerequisites link in readme is broken
- Issue #1020⁴⁹⁴² - HPX Deadlock on external synchronization
- Issue #1019⁴⁹⁴³ - Convert using `BOOST_ASSERT` to `HPX_ASSERT`
- Issue #1018⁴⁹⁴⁴ - compiling bug with gcc 4.8.1
- Issue #1017⁴⁹⁴⁵ - Possible crash in io_pool executor
- Issue #1016⁴⁹⁴⁶ - Crash at startup
- Issue #1014⁴⁹⁴⁷ - Implement Increment/Decrement Merging
- Issue #1013⁴⁹⁴⁸ - Add more logging channels to enable greater control over logging granularity
- Issue #1012⁴⁹⁴⁹ - `--hpx:debug-hpx-log` and `--hpx:debug-agas-log` lead to non-thread safe writes
- Issue #1011⁴⁹⁵⁰ - After installation, running applications from the build/staging directory no longer works
- Issue #1010⁴⁹⁵¹ - Mergeable decrement requests are not being merged
- Issue #1009⁴⁹⁵² - `--hpx:list-symbolic-names` crashes
- Issue #1007⁴⁹⁵³ - Components are not properly destroyed
- Issue #1006⁴⁹⁵⁴ - Segfault/hang in `set_data`
- Issue #1003⁴⁹⁵⁵ - Performance counter naming issue
- Issue #982⁴⁹⁵⁶ - Race condition during startup
- Issue #912⁴⁹⁵⁷ - OS X: component type not found in map
- Issue #663⁴⁹⁵⁸ - Create a buildbot slave based on Clang 3.2/OSX
- Issue #636⁴⁹⁵⁹ - Expose `this_locality::apply<act>(p1, p2);` for local execution

⁴⁹³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1026>

⁴⁹³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1025>

⁴⁹³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1024>

⁴⁹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1023>

⁴⁹⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/1022>

⁴⁹⁴² <https://github.com/STELLAR-GROUP/hpx/issues/1020>

⁴⁹⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/1019>

⁴⁹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1018>

⁴⁹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1017>

⁴⁹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1016>

⁴⁹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/1014>

⁴⁹⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/1013>

⁴⁹⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/1012>

⁴⁹⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/1011>

⁴⁹⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/1010>

⁴⁹⁵² <https://github.com/STELLAR-GROUP/hpx/issues/1009>

⁴⁹⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/1007>

⁴⁹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1006>

⁴⁹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1003>

⁴⁹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/982>

⁴⁹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/912>

⁴⁹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/663>

⁴⁹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/636>

- Issue #197⁴⁹⁶⁰ - Add --console=address option for PBS runs
- Issue #175⁴⁹⁶¹ - Asynchronous AGAS API

2.10.19 HPX V0.9.7 (Nov 13, 2013)

We have had over 1000 commits since the last release and we have closed over 180 tickets (bugs, feature requests, etc.).

General changes

- Ported HPX to BlueGene/Q
- Improved HPX support for Xeon/Phi accelerators
- Reimplemented `hpx::bind`, `hpx::tuple`, and `hpx::function` for better performance and better compliance with the C++11 Standard. Added `hpx::mem_fn`.
- Reworked `hpx::when_all` and `hpx::when_any` for better compliance with the ongoing C++ standardization effort, added heterogeneous version for those functions. Added `hpx::when_any_swapped`.
- Added `hpx::copy` as a precursor for a migrate functionality
- Added `hpx::get_ptr` allowing to directly access the memory underlying a given component
- Added the `hpx::lcos::broadcast`, `hpx::lcos::reduce`, and `hpx::lcos::fold` collective operations
- Added `hpx::get_locality_name` allowing to retrieve the name of any of the localities for the application.
- Added support for more flexible thread affinity control from the HPX command line, such as new modes for `--hpx:bind` (balanced, scattered, compact), improved default settings when running multiple localities on the same node.
- Added experimental executors for simpler thread pooling and scheduling. This API may change in the future as it will stay aligned with the ongoing C++ standardization efforts.
- Massively improved the performance of the HPX serialization code. Added partial support for zero copy serialization of array and bitwise-copyable types.
- General performance improvements of the code related to threads and futures.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- Issue #1005⁴⁹⁶² - Allow one to disable array optimizations and zero copy optimizations for each parcelport
- Issue #1004⁴⁹⁶³ - Generate new HPX logo image for the docs
- Issue #1002⁴⁹⁶⁴ - If MPI parcelport is not available, running HPX under mpirun should fail
- Issue #1001⁴⁹⁶⁵ - Zero copy serialization raises assert

⁴⁹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/197>

⁴⁹⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/175>

⁴⁹⁶² <https://github.com/STELLAR-GROUP/hpx/issues/1005>

⁴⁹⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/1004>

⁴⁹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/1002>

⁴⁹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/1001>

- Issue #1000⁴⁹⁶⁶ - Can't connect to a HPX application running with the MPI parcelport from a non MPI parcelport locality
- Issue #999⁴⁹⁶⁷ - Optimize hpx::when_n
- Issue #998⁴⁹⁶⁸ - Fixed const-correctness
- Issue #997⁴⁹⁶⁹ - Making serialize_buffer::data() type save
- Issue #996⁴⁹⁷⁰ - Memory leak in hpx::lcos::promise
- Issue #995⁴⁹⁷¹ - Race while registering pre-shutdown functions
- Issue #994⁴⁹⁷² - thread_rescheduling regression test does not compile
- Issue #992⁴⁹⁷³ - Correct comments and messages
- Issue #991⁴⁹⁷⁴ - setcap cap_sys_rawio=ep for power profiling causes an HPX application to abort
- Issue #989⁴⁹⁷⁵ - Jacobi hangs during execution
- Issue #988⁴⁹⁷⁶ - multiple_init test is failing
- Issue #986⁴⁹⁷⁷ - Can't call a function called "init" from "main" when using <hpx/hpx_main.hpp>
- Issue #984⁴⁹⁷⁸ - Reference counting tests are failing
- Issue #983⁴⁹⁷⁹ - thread_suspension_executor test fails
- Issue #980⁴⁹⁸⁰ - Terminating HPX threads don't leave stack in virgin state
- Issue #979⁴⁹⁸¹ - Static scheduler not in documents
- Issue #978⁴⁹⁸² - Preprocessing limits are broken
- Issue #977⁴⁹⁸³ - Make tests.regressions.lcos.future_hang_on_get shorter
- Issue #976⁴⁹⁸⁴ - Wrong library order in pkgconfig
- Issue #975⁴⁹⁸⁵ - Please reopen #963
- Issue #974⁴⁹⁸⁶ - Option pu-offset ignored in fixing_588 branch
- Issue #972⁴⁹⁸⁷ - Cannot use MKL with HPX
- Issue #969⁴⁹⁸⁸ - Non-existent INI files requested on the command line via --hpx:config do not cause warn-

⁴⁹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/1000>

⁴⁹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/999>

⁴⁹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/998>

⁴⁹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/997>

⁴⁹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/996>

⁴⁹⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/995>

⁴⁹⁷² <https://github.com/STELLAR-GROUP/hpx/issues/994>

⁴⁹⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/992>

⁴⁹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/991>

⁴⁹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/989>

⁴⁹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/988>

⁴⁹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/986>

⁴⁹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/984>

⁴⁹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/983>

⁴⁹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/980>

⁴⁹⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/979>

⁴⁹⁸² <https://github.com/STELLAR-GROUP/hpx/issues/978>

⁴⁹⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/977>

⁴⁹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/976>

⁴⁹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/975>

⁴⁹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/974>

⁴⁹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/972>

⁴⁹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/969>

ings or errors.

- Issue #968⁴⁹⁸⁹ - Cannot build examples in fixing_588 branch
- Issue #967⁴⁹⁹⁰ - Command line description of --hpx:queuing seems wrong
- Issue #966⁴⁹⁹¹ - --hpx:print-bind physical core numbers are wrong
- Issue #965⁴⁹⁹² - Deadlock when building in Release mode
- Issue #963⁴⁹⁹³ - Not all worker threads are working
- Issue #962⁴⁹⁹⁴ - Problem with SLURM integration
- Issue #961⁴⁹⁹⁵ - --hpx:print-bind outputs incorrect information
- Issue #960⁴⁹⁹⁶ - Fix cut and paste error in documentation of get_thread_priority
- Issue #959⁴⁹⁹⁷ - Change link to boost.atomic in documentation to point to boost.org
- Issue #958⁴⁹⁹⁸ - Undefined reference to intrusive_ptr_release
- Issue #957⁴⁹⁹⁹ - Make tuple standard compliant
- Issue #956⁵⁰⁰⁰ - Segfault with a3382fb
- Issue #955⁵⁰⁰¹ - --hpx:nodes and --hpx:nodefiles do not work with foreign nodes
- Issue #954⁵⁰⁰² - Make order of arguments for hpx::async and hpx::broadcast consistent
- Issue #953⁵⁰⁰³ - Cannot use MKL with HPX
- Issue #952⁵⁰⁰⁴ - register_[pre_] shutdown_function never throw
- Issue #951⁵⁰⁰⁵ - Assert when number of threads is greater than hardware concurrency
- Issue #948⁵⁰⁰⁶ - HPX_HAVE_GENERIC_CONTEXT_COROUTINES conflicts with HPX_HAVE_FIBER_BASED_COROUTINES
- Issue #947⁵⁰⁰⁷ - Need MPI_THREAD_MULTIPLE for backward compatibility
- Issue #946⁵⁰⁰⁸ - HPX does not call MPI_Finalize
- Issue #945⁵⁰⁰⁹ - Segfault with hpx::lcos::broadcast
- Issue #944⁵⁰¹⁰ - OS X: assertion pu_offset_ < hardware_concurrency failed

⁴⁹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/968>

⁴⁹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/967>

⁴⁹⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/966>

⁴⁹⁹² <https://github.com/STELLAR-GROUP/hpx/issues/965>

⁴⁹⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/963>

⁴⁹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/962>

⁴⁹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/961>

⁴⁹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/960>

⁴⁹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/959>

⁴⁹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/958>

⁴⁹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/957>

⁵⁰⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/956>

⁵⁰⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/955>

⁵⁰⁰² <https://github.com/STELLAR-GROUP/hpx/issues/954>

⁵⁰⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/953>

⁵⁰⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/952>

⁵⁰⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/951>

⁵⁰⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/948>

⁵⁰⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/947>

⁵⁰⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/946>

⁵⁰⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/945>

⁵⁰¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/944>

- Issue #943⁵⁰¹¹ - #include <hpx/hpx_main.hpp> does not work
- Issue #942⁵⁰¹² - Make the BG/Q work with -O3
- Issue #940⁵⁰¹³ - Use separator when concatenating locality name
- Issue #939⁵⁰¹⁴ - Refactor MPI parcelport to use MPI_Wait instead of multiple MPI_Test calls
- Issue #938⁵⁰¹⁵ - Want to officially access client_base::gid_
- Issue #937⁵⁰¹⁶ - client_base::gid_ should be private``
- Issue #936⁵⁰¹⁷ - Want doxygen-like source code index
- Issue #935⁵⁰¹⁸ - Build error with gcc 4.6 and Boost 1.54.0 on hpx trunk and 0.9.6
- Issue #933⁵⁰¹⁹ - Cannot build HPX with Boost 1.54.0
- Issue #932⁵⁰²⁰ - Components are destructed too early
- Issue #931⁵⁰²¹ - Make HPX work on BG/Q
- Issue #930⁵⁰²² - make git-docs is broken
- Issue #929⁵⁰²³ - Generating index in docs broken
- Issue #928⁵⁰²⁴ - Optimize hpx::util::static_ for C++11 compilers supporting magic statics
- Issue #924⁵⁰²⁵ - Make kill_process_tree (in process.py) more robust on Mac OSX
- Issue #923⁵⁰²⁶ - Correct BLAS and RNPL cmake tests
- Issue #922⁵⁰²⁷ - Cannot link against BLAS
- Issue #921⁵⁰²⁸ - Implement hpx::mem_fn
- Issue #920⁵⁰²⁹ - Output locality with --hpx:print-bind
- Issue #919⁵⁰³⁰ - Correct grammar; simplify boolean expressions
- Issue #918⁵⁰³¹ - Link to hello_world.cpp is broken
- Issue #917⁵⁰³² - adapt cmake file to new boostbook version
- Issue #916⁵⁰³³ - fix problem building documentation with xsltproc >= 1.1.27

⁵⁰¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/943>

⁵⁰¹² <https://github.com/STELLAR-GROUP/hpx/issues/942>

⁵⁰¹³ <https://github.com/STELLAR-GROUP/hpx/issues/940>

⁵⁰¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/939>

⁵⁰¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/938>

⁵⁰¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/937>

⁵⁰¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/936>

⁵⁰¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/935>

⁵⁰¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/933>

⁵⁰²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/932>

⁵⁰²¹ <https://github.com/STELLAR-GROUP/hpx/issues/931>

⁵⁰²² <https://github.com/STELLAR-GROUP/hpx/issues/930>

⁵⁰²³ <https://github.com/STELLAR-GROUP/hpx/issues/929>

⁵⁰²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/928>

⁵⁰²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/924>

⁵⁰²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/923>

⁵⁰²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/922>

⁵⁰²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/921>

⁵⁰²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/920>

⁵⁰³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/919>

⁵⁰³¹ <https://github.com/STELLAR-GROUP/hpx/issues/918>

⁵⁰³² <https://github.com/STELLAR-GROUP/hpx/issues/917>

⁵⁰³³ <https://github.com/STELLAR-GROUP/hpx/issues/916>

- Issue #915⁵⁰³⁴ - Add another TBBMalloc library search path
- Issue #914⁵⁰³⁵ - Build problem with Intel compiler on Stampede (TACC)
- Issue #913⁵⁰³⁶ - fix error messages in fibonacci examples
- Issue #911⁵⁰³⁷ - Update OS X build instructions
- Issue #910⁵⁰³⁸ - Want like to specify MPI_ROOT instead of compiler wrapper script
- Issue #909⁵⁰³⁹ - Warning about void* arithmetic
- Issue #908⁵⁰⁴⁰ - Buildbot for MIC is broken
- Issue #906⁵⁰⁴¹ - Can't use --hpx:bind=balanced with multiple MPI processes
- Issue #905⁵⁰⁴² - --hpx:bind documentation should describe full grammar
- Issue #904⁵⁰⁴³ - Add hpx::lcos::fold and hpx::lcos::inverse_fold collective operation
- Issue #903⁵⁰⁴⁴ - Add hpx::when_any_swapped()
- Issue #902⁵⁰⁴⁵ - Add hpx::lcos::reduce collective operation
- Issue #901⁵⁰⁴⁶ - Web documentation is not searchable
- Issue #900⁵⁰⁴⁷ - Web documentation for trunk has no index
- Issue #898⁵⁰⁴⁸ - Some tests fail with GCC 4.8.1 and MPI parcel port
- Issue #897⁵⁰⁴⁹ - HWLOC causes failures on Mac
- Issue #896⁵⁰⁵⁰ - pu-offset leads to startup error
- Issue #895⁵⁰⁵¹ - hpx::get_locality_name not defined
- Issue #894⁵⁰⁵² - Race condition at shutdown
- Issue #893⁵⁰⁵³ - --hpx:print-bind switches std::cout to hexadecimal mode
- Issue #892⁵⁰⁵⁴ - hwloc_topology_load can be expensive – don't call multiple times
- Issue #891⁵⁰⁵⁵ - The documentation for get_locality_name is wrong
- Issue #890⁵⁰⁵⁶ - --hpx:print-bind should not exit

⁵⁰³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/915>

⁵⁰³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/914>

⁵⁰³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/913>

⁵⁰³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/911>

⁵⁰³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/910>

⁵⁰³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/909>

⁵⁰⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/908>

⁵⁰⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/906>

⁵⁰⁴² <https://github.com/STELLAR-GROUP/hpx/issues/905>

⁵⁰⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/904>

⁵⁰⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/903>

⁵⁰⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/902>

⁵⁰⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/901>

⁵⁰⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/900>

⁵⁰⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/898>

⁵⁰⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/897>

⁵⁰⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/896>

⁵⁰⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/895>

⁵⁰⁵² <https://github.com/STELLAR-GROUP/hpx/issues/894>

⁵⁰⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/893>

⁵⁰⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/892>

⁵⁰⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/891>

⁵⁰⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/890>

- Issue #889⁵⁰⁵⁷ --hpx:debug-hpx-log=FILE does not work
- Issue #888⁵⁰⁵⁸ - MPI parcelport does not exit cleanly for -hpx:print-bind
- Issue #887⁵⁰⁵⁹ - Choose thread affinities more cleverly
- Issue #886⁵⁰⁶⁰ - Logging documentation is confusing
- Issue #885⁵⁰⁶¹ - Two threads are slower than one
- Issue #884⁵⁰⁶² - is_callable failing with member pointers in C++11
- Issue #883⁵⁰⁶³ - Need help with is_callable_test
- Issue #882⁵⁰⁶⁴ - tests.regressions.lcos.future_hang_on_get does not terminate
- Issue #881⁵⁰⁶⁵ - tests/regressions/block_matrix/matrix.hh won't compile with GCC 4.8.1
- Issue #880⁵⁰⁶⁶ - HPX does not work on OS X
- Issue #878⁵⁰⁶⁷ - future::unwrap triggers assertion
- Issue #877⁵⁰⁶⁸ - “make tests” has build errors on Ubuntu 12.10
- Issue #876⁵⁰⁶⁹ - tcmalloc is used by default, even if it is not present
- Issue #875⁵⁰⁷⁰ - global_fixture is defined in a header file
- Issue #874⁵⁰⁷¹ - Some tests take very long
- Issue #873⁵⁰⁷² - Add block-matrix code as regression test
- Issue #872⁵⁰⁷³ - HPX documentation does not say how to run tests with detailed output
- Issue #871⁵⁰⁷⁴ - All tests fail with “make test”
- Issue #870⁵⁰⁷⁵ - Please explicitly disable serialization in classes that don't support it
- Issue #868⁵⁰⁷⁶ - boost_any test failing
- Issue #867⁵⁰⁷⁷ - Reduce the number of copies of hpx::function arguments
- Issue #863⁵⁰⁷⁸ - Futures should not require a default constructor
- Issue #862⁵⁰⁷⁹ - value_or_error shall not default construct its result

⁵⁰⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/889>

⁵⁰⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/888>

⁵⁰⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/887>

⁵⁰⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/886>

⁵⁰⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/885>

⁵⁰⁶² <https://github.com/STELLAR-GROUP/hpx/issues/884>

⁵⁰⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/883>

⁵⁰⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/882>

⁵⁰⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/881>

⁵⁰⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/880>

⁵⁰⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/878>

⁵⁰⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/877>

⁵⁰⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/876>

⁵⁰⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/875>

⁵⁰⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/874>

⁵⁰⁷² <https://github.com/STELLAR-GROUP/hpx/issues/873>

⁵⁰⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/872>

⁵⁰⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/871>

⁵⁰⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/870>

⁵⁰⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/868>

⁵⁰⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/867>

⁵⁰⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/863>

⁵⁰⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/862>

- Issue #861⁵⁰⁸⁰ - HPX_UNUSED macro
- Issue #860⁵⁰⁸¹ - Add functionality to copy construct a component
- Issue #859⁵⁰⁸² - hpx::endl should flush
- Issue #858⁵⁰⁸³ - Create hpx::get_ptr<> allowing to access component implementation
- Issue #855⁵⁰⁸⁴ - Implement hpx::INVOKER
- Issue #854⁵⁰⁸⁵ - hpx/hpx.hpp does not include hpx/include/iostreams.hpp
- Issue #853⁵⁰⁸⁶ - Feature request: null future
- Issue #852⁵⁰⁸⁷ - Feature request: Locality names
- Issue #851⁵⁰⁸⁸ - hpx::cout output does not appear on screen
- Issue #849⁵⁰⁸⁹ - All tests fail on OS X after installing
- Issue #848⁵⁰⁹⁰ - Update OS X build instructions
- Issue #846⁵⁰⁹¹ - Update hpx_external_example
- Issue #845⁵⁰⁹² - Issues with having both debug and release modules in the same directory
- Issue #844⁵⁰⁹³ - Create configuration header
- Issue #843⁵⁰⁹⁴ - Tests should use CTest
- Issue #842⁵⁰⁹⁵ - Remove buffer_pool from MPI parcelport
- Issue #841⁵⁰⁹⁶ - Add possibility to broadcast an index with hpx::lcos::broadcast
- Issue #838⁵⁰⁹⁷ - Simplify util::tuple
- Issue #837⁵⁰⁹⁸ - Adopt boost::tuple tests for util::tuple
- Issue #836⁵⁰⁹⁹ - Adopt boost::function tests for util::function
- Issue #835⁵¹⁰⁰ - Tuple interface missing pieces
- Issue #833⁵¹⁰¹ - Partially preprocessing files not working
- Issue #832⁵¹⁰² - Native papi counters do not work with wild cards

5080 <https://github.com/STELLAR-GROUP/hpx/issues/861>

5081 <https://github.com/STELLAR-GROUP/hpx/issues/860>

5082 <https://github.com/STELLAR-GROUP/hpx/issues/859>

5083 <https://github.com/STELLAR-GROUP/hpx/issues/858>

5084 <https://github.com/STELLAR-GROUP/hpx/issues/855>

5085 <https://github.com/STELLAR-GROUP/hpx/issues/854>

5086 <https://github.com/STELLAR-GROUP/hpx/issues/853>

5087 <https://github.com/STELLAR-GROUP/hpx/issues/852>

5088 <https://github.com/STELLAR-GROUP/hpx/issues/851>

5089 <https://github.com/STELLAR-GROUP/hpx/issues/849>

5090 <https://github.com/STELLAR-GROUP/hpx/issues/848>

5091 <https://github.com/STELLAR-GROUP/hpx/issues/846>

5092 <https://github.com/STELLAR-GROUP/hpx/issues/845>

5093 <https://github.com/STELLAR-GROUP/hpx/issues/844>

5094 <https://github.com/STELLAR-GROUP/hpx/issues/843>

5095 <https://github.com/STELLAR-GROUP/hpx/issues/842>

5096 <https://github.com/STELLAR-GROUP/hpx/issues/841>

5097 <https://github.com/STELLAR-GROUP/hpx/issues/838>

5098 <https://github.com/STELLAR-GROUP/hpx/issues/837>

5099 <https://github.com/STELLAR-GROUP/hpx/issues/836>

5100 <https://github.com/STELLAR-GROUP/hpx/issues/835>

5101 <https://github.com/STELLAR-GROUP/hpx/issues/833>

5102 <https://github.com/STELLAR-GROUP/hpx/issues/832>

- Issue #831⁵¹⁰³ - Arithmetics counter fails if only one parameter is given
- Issue #830⁵¹⁰⁴ - Convert hpx::util::function to use new scheme for serializing its base pointer
- Issue #829⁵¹⁰⁵ - Consistently use `decay<T>` instead of `remove_const< remove_reference<T>>`
- Issue #828⁵¹⁰⁶ - Update future implementation to N3721 and N3722
- Issue #827⁵¹⁰⁷ - Enable MPI parcelport for bootstrapping whenever application was started using `mpirun`
- Issue #826⁵¹⁰⁸ - Support command line option `--hpx:print-bind` even if `--hpx::bind` was not used
- Issue #825⁵¹⁰⁹ - Memory counters give segfault when attempting to use thread wild cards or numbers only total works
- Issue #824⁵¹¹⁰ - Enable lambda functions to be used with `hpx::async/hpx::apply`
- Issue #823⁵¹¹¹ - Using a hashing filter
- Issue #822⁵¹¹² - Silence unused variable warning
- Issue #821⁵¹¹³ - Detect if a function object is callable with given arguments
- Issue #820⁵¹¹⁴ - Allow wildcards to be used for performance counter names
- Issue #819⁵¹¹⁵ - Make the AGAS symbolic name registry distributed
- Issue #818⁵¹¹⁶ - Add `future::then()` overload taking an executor
- Issue #817⁵¹¹⁷ - Fixed typo
- Issue #815⁵¹¹⁸ - Create an lco that is performing an efficient broadcast of actions
- Issue #814⁵¹¹⁹ - Papi counters cannot specify `thread#*` to get the counts for all threads
- Issue #813⁵¹²⁰ - Scoped unlock
- Issue #811⁵¹²¹ - `simple_central_tuplespace_client` run error
- Issue #810⁵¹²² - ostream error when << any objects
- Issue #809⁵¹²³ - Optimize parcel serialization
- Issue #808⁵¹²⁴ - HPX applications throw exception when executed from the build directory
- Issue #807⁵¹²⁵ - Create performance counters exposing overall AGAS statistics

⁵¹⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/831>

⁵¹⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/830>

⁵¹⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/829>

⁵¹⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/828>

⁵¹⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/827>

⁵¹⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/826>

⁵¹⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/825>

⁵¹¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/824>

⁵¹¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/823>

⁵¹¹² <https://github.com/STELLAR-GROUP/hpx/issues/822>

⁵¹¹³ <https://github.com/STELLAR-GROUP/hpx/issues/821>

⁵¹¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/820>

⁵¹¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/819>

⁵¹¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/818>

⁵¹¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/817>

⁵¹¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/815>

⁵¹¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/814>

⁵¹²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/813>

⁵¹²¹ <https://github.com/STELLAR-GROUP/hpx/issues/811>

⁵¹²² <https://github.com/STELLAR-GROUP/hpx/issues/810>

⁵¹²³ <https://github.com/STELLAR-GROUP/hpx/issues/809>

⁵¹²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/808>

⁵¹²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/807>

- Issue #795⁵¹²⁶ - Create timed make_ready_future
- Issue #794⁵¹²⁷ - Create heterogeneous when_all/when_any/etc.
- Issue #721⁵¹²⁸ - Make HPX usable for Xeon Phi
- Issue #694⁵¹²⁹ - CMake should complain if you attempt to build an example without its dependencies
- Issue #692⁵¹³⁰ - SLURM support broken
- Issue #683⁵¹³¹ - python/hpx/process.py imports epoll on all platforms
- Issue #619⁵¹³² - Automate the doc building process
- Issue #600⁵¹³³ - GTC performance broken
- Issue #577⁵¹³⁴ - Allow for zero copy serialization/networking
- Issue #551⁵¹³⁵ - Change executable names to have debug postfix in Debug builds
- Issue #544⁵¹³⁶ - Write a custom .lib file on Windows pulling in hpx_init and hpx.dll, phase out hpx_init
- Issue #534⁵¹³⁷ - hpx::init should take functions by std::function and should accept all forms of hpx_main
- Issue #508⁵¹³⁸ - FindPackage fails to set FOO_LIBRARY_DIR
- Issue #506⁵¹³⁹ - Add cmake support to generate ini files for external applications
- Issue #470⁵¹⁴⁰ - Changing build-type after configure does not update boost library names
- Issue #453⁵¹⁴¹ - Document hpx_run_tests.py
- Issue #445⁵¹⁴² - Significant performance mismatch between MPI and HPX in SMP for allgather example
- Issue #443⁵¹⁴³ - Make docs viewable from build directory
- Issue #421⁵¹⁴⁴ - Support multiple HPX instances per node in a batch environment like PBS or SLURM
- Issue #316⁵¹⁴⁵ - Add message size limitation
- Issue #249⁵¹⁴⁶ - Clean up locking code in big boot barrier
- Issue #136⁵¹⁴⁷ - Persistent CMake variables need to be marked as cache variables

⁵¹²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/795>

⁵¹²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/794>

⁵¹²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/721>

⁵¹²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/694>

⁵¹³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/692>

⁵¹³¹ <https://github.com/STELLAR-GROUP/hpx/issues/683>

⁵¹³² <https://github.com/STELLAR-GROUP/hpx/issues/619>

⁵¹³³ <https://github.com/STELLAR-GROUP/hpx/issues/600>

⁵¹³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/577>

⁵¹³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/551>

⁵¹³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/544>

⁵¹³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/534>

⁵¹³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/508>

⁵¹³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/506>

⁵¹⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/470>

⁵¹⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/453>

⁵¹⁴² <https://github.com/STELLAR-GROUP/hpx/issues/445>

⁵¹⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/443>

⁵¹⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/421>

⁵¹⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/316>

⁵¹⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/249>

⁵¹⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/136>

2.10.20 HPX V0.9.6 (Jul 30, 2013)

We have had over 1200 commits since the last release and we have closed roughly 140 tickets (bugs, feature requests, etc.).

General changes

The major new features in this release are:

- We further consolidated the API exposed by *HPX*. We aligned our APIs as much as possible with the existing C++11 Standard⁵¹⁴⁸ and related proposals to the C++ standardization committee (such as N3632⁵¹⁴⁹ and N3857⁵¹⁵⁰).
- We implemented a first version of a distributed AGAS service which essentially eliminates all explicit AGAS network traffic.
- We created a native ibverbs parcelport allowing to take advantage of the superior latency and bandwidth characteristics of Infiniband networks.
- We successfully ported *HPX* to the Xeon Phi platform.
- Support for the SLURM scheduling system was implemented.
- Major efforts have been dedicated to improving the performance counter framework, numerous new counters were implemented and new APIs were added.
- We added a modular parcel compression system allowing to improve bandwidth utilization (by reducing the overall size of the transferred data).
- We added a modular parcel coalescing system allowing to combine several parcels into larger messages. This reduces latencies introduced by the communication layer.
- Added an experimental executors API allowing to use different scheduling policies for different parts of the code. This API has been modelled after the Standards proposal N3562⁵¹⁵¹. This API is bound to change in the future, though.
- Added minimal security support for localities which is enforced on the parcelport level. This support is preliminary and experimental and might change in the future.
- We created a parcelport using low level MPI functions. This is in support of legacy applications which are to be gradually ported and to support platforms where MPI is the only available portable networking layer.
- We added a preliminary and experimental implementation of a tuple-space object which exposes an interface similar to such systems described in the literature (see for instance The Linda Coordination Language⁵¹⁵²).

⁵¹⁴⁸ <http://www.open-std.org/jtc1/sc22/wg21>

⁵¹⁴⁹ <http://wg21.link/n3632>

⁵¹⁵⁰ <http://wg21.link/n3857>

⁵¹⁵¹ <http://wg21.link/n3562>

⁵¹⁵² [https://en.wikipedia.org/wiki/Linda_\(coordination_language\)](https://en.wikipedia.org/wiki/Linda_(coordination_language))

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release. This is again a very long list of newly implemented features and fixed issues.

- Issue #806⁵¹⁵³ - make (all) in examples folder does nothing
- Issue #805⁵¹⁵⁴ - Adding the introduction and fixing DOCBOOK dependencies for Windows use
- Issue #804⁵¹⁵⁵ - Add stackless (non-suspendable) thread type
- Issue #803⁵¹⁵⁶ - Create proper serialization support functions for util::tuple
- Issue #800⁵¹⁵⁷ - Add possibility to disable array optimizations during serialization
- Issue #798⁵¹⁵⁸ - HPX_LIMIT does not work for local dataflow
- Issue #797⁵¹⁵⁹ - Create a parcelport which uses MPI
- Issue #796⁵¹⁶⁰ - Problem with Large Numbers of Threads
- Issue #793⁵¹⁶¹ - Changing dataflow test case to hang consistently
- Issue #792⁵¹⁶² - CMake Error
- Issue #791⁵¹⁶³ - Problems with local::dataflow
- Issue #790⁵¹⁶⁴ - wait_for() doesn't compile
- Issue #789⁵¹⁶⁵ - HPX with Intel compiler segfaults
- Issue #788⁵¹⁶⁶ - Intel compiler support
- Issue #787⁵¹⁶⁷ - Fixed SFINAEd specializations
- Issue #786⁵¹⁶⁸ - Memory issues during benchmarking.
- Issue #785⁵¹⁶⁹ - Create an API allowing to register external threads with HPX
- Issue #784⁵¹⁷⁰ - util::plugin is throwing an error when a symbol is not found
- Issue #783⁵¹⁷¹ - How does hpx:bind work?
- Issue #782⁵¹⁷² - Added quotes around STRING REPLACE potentially empty arguments
- Issue #781⁵¹⁷³ - Make sure no exceptions propagate into the thread manager

⁵¹⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/806>

⁵¹⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/805>

⁵¹⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/804>

⁵¹⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/803>

⁵¹⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/800>

⁵¹⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/798>

⁵¹⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/797>

⁵¹⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/796>

⁵¹⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/793>

⁵¹⁶² <https://github.com/STELLAR-GROUP/hpx/issues/792>

⁵¹⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/791>

⁵¹⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/790>

⁵¹⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/789>

⁵¹⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/788>

⁵¹⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/787>

⁵¹⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/786>

⁵¹⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/785>

⁵¹⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/784>

⁵¹⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/783>

⁵¹⁷² <https://github.com/STELLAR-GROUP/hpx/issues/782>

⁵¹⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/781>

- Issue #780⁵¹⁷⁴ - Allow arithmetics performance counters to expand its parameters
- Issue #779⁵¹⁷⁵ - Test case for 778
- Issue #778⁵¹⁷⁶ - Swapping futures segfaults
- Issue #777⁵¹⁷⁷ - hpx::lcos::details::when_xxx don't restore completion handlers
- Issue #776⁵¹⁷⁸ - Compiler chokes on dataflow overload with launch policy
- Issue #775⁵¹⁷⁹ - Runtime error with local dataflow (copying futures?)
- Issue #774⁵¹⁸⁰ - Using local dataflow without explicit namespace
- Issue #773⁵¹⁸¹ - Local dataflow with unwrap: functor operators need to be const
- Issue #772⁵¹⁸² - Allow (remote) actions to return a future
- Issue #771⁵¹⁸³ - Setting HPX_LIMIT gives huge boost MPL errors
- Issue #770⁵¹⁸⁴ - Add launch policy to (local) dataflow
- Issue #769⁵¹⁸⁵ - Make compile time configuration information available
- Issue #768⁵¹⁸⁶ - Const correctness problem in local dataflow
- Issue #767⁵¹⁸⁷ - Add launch policies to async
- Issue #766⁵¹⁸⁸ - Mark data structures for optimized (array based) serialization
- Issue #765⁵¹⁸⁹ - Align hpx::any with N3508: Any Library Proposal (Revision 2)
- Issue #764⁵¹⁹⁰ - Align hpx::future with newest N3558: A Standardized Representation of Asynchronous Operations
- Issue #762⁵¹⁹¹ - added a human readable output for the ping pong example
- Issue #761⁵¹⁹² - Ambiguous typename when constructing derived component
- Issue #760⁵¹⁹³ - Simple components can not be derived
- Issue #759⁵¹⁹⁴ - make install doesn't give a complete install
- Issue #758⁵¹⁹⁵ - Stack overflow when using locking_hook<>
- Issue #757⁵¹⁹⁶ - copy paste error; unsupported function overloading

⁵¹⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/780>

⁵¹⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/779>

⁵¹⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/778>

⁵¹⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/777>

⁵¹⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/776>

⁵¹⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/775>

⁵¹⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/774>

⁵¹⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/773>

⁵¹⁸² <https://github.com/STELLAR-GROUP/hpx/issues/772>

⁵¹⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/771>

⁵¹⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/770>

⁵¹⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/769>

⁵¹⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/768>

⁵¹⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/767>

⁵¹⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/766>

⁵¹⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/765>

⁵¹⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/764>

⁵¹⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/762>

⁵¹⁹² <https://github.com/STELLAR-GROUP/hpx/issues/761>

⁵¹⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/760>

⁵¹⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/759>

⁵¹⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/758>

⁵¹⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/757>

- Issue #756⁵¹⁹⁷ - GTCX runtime issue in Gordon
- Issue #755⁵¹⁹⁸ - Papi counters don't work with reset and evaluate API's
- Issue #753⁵¹⁹⁹ - cmake bugfix and improved component action docs
- Issue #752⁵²⁰⁰ - hpx simple component docs
- Issue #750⁵²⁰¹ - Add hpx::util::any
- Issue #749⁵²⁰² - Thread phase counter is not reset
- Issue #748⁵²⁰³ - Memory performance counter are not registered
- Issue #747⁵²⁰⁴ - Create performance counters exposing arithmetic operations
- Issue #745⁵²⁰⁵ - apply_callback needs to invoke callback when applied locally
- Issue #744⁵²⁰⁶ - CMake fixes
- Issue #743⁵²⁰⁷ - Problem Building github version of HPX
- Issue #742⁵²⁰⁸ - Remove HPX_STD_BIND
- Issue #741⁵²⁰⁹ - assertion 'px != 0' failed: HPX(assertion_failure) for low numbers of OS threads
- Issue #739⁵²¹⁰ - Performance counters do not count to the end of the program or evaluation
- Issue #738⁵²¹¹ - Dedicated AGAS server runs don't work; console ignores -a option.
- Issue #737⁵²¹² - Missing bind overloads
- Issue #736⁵²¹³ - Performance counter wildcards do not always work
- Issue #735⁵²¹⁴ - Create native ibverbs parcelport based on rdma operations
- Issue #734⁵²¹⁵ - Threads stolen performance counter total is incorrect
- Issue #733⁵²¹⁶ - Test benchmarks need to be checked and fixed
- Issue #732⁵²¹⁷ - Build fails with Mac, using mac ports clang-3.3 on latest git branch
- Issue #731⁵²¹⁸ - Add global start/stop API for performance counters
- Issue #730⁵²¹⁹ - Performance counter values are apparently incorrect

⁵¹⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/756>

⁵¹⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/755>

⁵¹⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/753>

⁵²⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/752>

⁵²⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/750>

⁵²⁰² <https://github.com/STELLAR-GROUP/hpx/issues/749>

⁵²⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/748>

⁵²⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/747>

⁵²⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/745>

⁵²⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/744>

⁵²⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/743>

⁵²⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/742>

⁵²⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/741>

⁵²¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/739>

⁵²¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/738>

⁵²¹² <https://github.com/STELLAR-GROUP/hpx/issues/737>

⁵²¹³ <https://github.com/STELLAR-GROUP/hpx/issues/736>

⁵²¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/735>

⁵²¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/734>

⁵²¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/733>

⁵²¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/732>

⁵²¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/731>

⁵²¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/730>

- Issue #729⁵²²⁰ - Unhandled switch
- Issue #728⁵²²¹ - Serialization of hpx::util::function between two localities causes seg faults
- Issue #727⁵²²² - Memory counters on Mac OS X
- Issue #725⁵²²³ - Restore original thread priority on resume
- Issue #724⁵²²⁴ - Performance benchmarks do not depend on main HPX libraries
- Issue #723⁵²²⁵ - [teletype]-hpx:nodes=``cat \$PBS_NODEFILE`` works; -hpx:nodefile=\$PBS_NODEFILE does not.[c++]
- Issue #722⁵²²⁶ - Fix binding const member functions as actions
- Issue #719⁵²²⁷ - Create performance counter exposing compression ratio
- Issue #718⁵²²⁸ - Add possibility to compress parcel data
- Issue #717⁵²²⁹ - strip_credit_from_gid has misleading semantics
- Issue #716⁵²³⁰ - Non-option arguments to programs run using pbsdsh must be before --hpx:nodes, contrary to directions
- Issue #715⁵²³¹ - Re-thrown exceptions should retain the original call site
- Issue #714⁵²³² - failed assertion in debug mode
- Issue #713⁵²³³ - Add performance counters monitoring connection caches
- Issue #712⁵²³⁴ - Adjust parcel related performance counters to be connection type specific
- Issue #711⁵²³⁵ - configuration failure
- Issue #710⁵²³⁶ - Error “timed out while trying to find room in the connection cache” when trying to start multiple localities on a single computer
- Issue #709⁵²³⁷ - Add new thread state ‘staged’ referring to task descriptions
- Issue #708⁵²³⁸ - Detect/mitigate bad non-system installs of GCC on Redhat systems
- Issue #707⁵²³⁹ - Many examples do not link with Git HEAD version
- Issue #706⁵²⁴⁰ - hpx::init removes portions of non-option command line arguments before last = sign
- Issue #705⁵²⁴¹ - Create rolling average and median aggregating performance counters

⁵²²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/729>

⁵²²¹ <https://github.com/STELLAR-GROUP/hpx/issues/728>

⁵²²² <https://github.com/STELLAR-GROUP/hpx/issues/727>

⁵²²³ <https://github.com/STELLAR-GROUP/hpx/issues/725>

⁵²²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/724>

⁵²²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/723>

⁵²²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/722>

⁵²²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/719>

⁵²²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/718>

⁵²²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/717>

⁵²³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/716>

⁵²³¹ <https://github.com/STELLAR-GROUP/hpx/issues/715>

⁵²³² <https://github.com/STELLAR-GROUP/hpx/issues/714>

⁵²³³ <https://github.com/STELLAR-GROUP/hpx/issues/713>

⁵²³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/712>

⁵²³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/711>

⁵²³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/710>

⁵²³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/709>

⁵²³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/708>

⁵²³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/707>

⁵²⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/706>

⁵²⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/705>

- Issue #704⁵²⁴² - Create performance counter to expose thread queue waiting time
- Issue #703⁵²⁴³ - Add support to HPX build system to find libcrctool.a and related headers
- Issue #699⁵²⁴⁴ - Generalize instrumentation support
- Issue #698⁵²⁴⁵ - compilation failure with hwloc absent
- Issue #697⁵²⁴⁶ - Performance counter counts should be zero indexed
- Issue #696⁵²⁴⁷ - Distributed problem
- Issue #695⁵²⁴⁸ - Bad perf counter time printed
- Issue #693⁵²⁴⁹ - --help doesn't print component specific command line options
- Issue #692⁵²⁵⁰ - SLURM support broken
- Issue #691⁵²⁵¹ - exception while executing any application linked with hwloc
- Issue #690⁵²⁵² - thread_id_test and thread_launcher_test failing
- Issue #689⁵²⁵³ - Make the buildbots use hwloc
- Issue #687⁵²⁵⁴ - compilation error fix (hwloc_topology)
- Issue #686⁵²⁵⁵ - Linker Error for Applications
- Issue #684⁵²⁵⁶ - Pinning of service thread fails when number of worker threads equals the number of cores
- Issue #682⁵²⁵⁷ - Add performance counters exposing number of stolen threads
- Issue #681⁵²⁵⁸ - Add apply_continue for asynchronous chaining of actions
- Issue #679⁵²⁵⁹ - Remove obsolete async_callback API functions
- Issue #678⁵²⁶⁰ - Add new API for setting/triggering LCOs
- Issue #677⁵²⁶¹ - Add async_continue for true continuation style actions
- Issue #676⁵²⁶² - Buildbot for gcc 4.4 broken
- Issue #675⁵²⁶³ - Partial preprocessing broken
- Issue #674⁵²⁶⁴ - HPX segfaults when built with gcc 4.7

⁵²⁴² <https://github.com/STELLAR-GROUP/hpx/issues/704>

⁵²⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/703>

⁵²⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/699>

⁵²⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/698>

⁵²⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/697>

⁵²⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/696>

⁵²⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/695>

⁵²⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/693>

⁵²⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/692>

5251 <https://github.com/STELLAR-GROUP/hpx/issues/691>5252 <https://github.com/STELLAR-GROUP/hpx/issues/690>5253 <https://github.com/STELLAR-GROUP/hpx/issues/689>5254 <https://github.com/STELLAR-GROUP/hpx/issues/687>5255 <https://github.com/STELLAR-GROUP/hpx/issues/686>5256 <https://github.com/STELLAR-GROUP/hpx/issues/684>5257 <https://github.com/STELLAR-GROUP/hpx/issues/682>5258 <https://github.com/STELLAR-GROUP/hpx/issues/681>5259 <https://github.com/STELLAR-GROUP/hpx/issues/679>5260 <https://github.com/STELLAR-GROUP/hpx/issues/678>5261 <https://github.com/STELLAR-GROUP/hpx/issues/677>5262 <https://github.com/STELLAR-GROUP/hpx/issues/676>5263 <https://github.com/STELLAR-GROUP/hpx/issues/675>5264 <https://github.com/STELLAR-GROUP/hpx/issues/674>

- Issue #673⁵²⁶⁵ - use_guard_pages has inconsistent preprocessor guards
- Issue #672⁵²⁶⁶ - External build breaks if library path has spaces
- Issue #671⁵²⁶⁷ - release tarballs are tarbombs
- Issue #670⁵²⁶⁸ - CMake won't find Boost headers in layout=versioned install
- Issue #669⁵²⁶⁹ - Links in docs to source files broken if not installed
- Issue #667⁵²⁷⁰ - Not reading ini file properly
- Issue #664⁵²⁷¹ - Adapt new meanings of ‘const’ and ‘mutable’
- Issue #661⁵²⁷² - Implement BTL Parcel port
- Issue #655⁵²⁷³ - Make HPX work with the “decltype” result_of
- Issue #647⁵²⁷⁴ - documentation for specifying the number of high priority threads --hpx:high-priority-threads
- Issue #643⁵²⁷⁵ - Error parsing host file
- Issue #642⁵²⁷⁶ - HWLoc issue with TAU
- Issue #639⁵²⁷⁷ - Logging potentially suspends a running thread
- Issue #634⁵²⁷⁸ - Improve error reporting from parcel layer
- Issue #627⁵²⁷⁹ - Add tests for async and apply overloads that accept regular C++ functions
- Issue #626⁵²⁸⁰ - hpx/future.hpp header
- Issue #601⁵²⁸¹ - Intel support
- Issue #557⁵²⁸² - Remove action codes
- Issue #531⁵²⁸³ - AGAS request and response classes should use switch statements
- Issue #529⁵²⁸⁴ - Investigate the state of hwloc support
- Issue #526⁵²⁸⁵ - Make HPX aware of hyper-threading
- Issue #518⁵²⁸⁶ - Create facilities allowing to use plain arrays as action arguments
- Issue #473⁵²⁸⁷ - hwloc thread binding is broken on CPUs with hyperthreading

⁵²⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/673>

⁵²⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/672>

⁵²⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/671>

⁵²⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/670>

⁵²⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/669>

⁵²⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/667>

⁵²⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/664>

⁵²⁷² <https://github.com/STELLAR-GROUP/hpx/issues/661>

⁵²⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/655>

⁵²⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/647>

⁵²⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/643>

⁵²⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/642>

⁵²⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/639>

⁵²⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/634>

⁵²⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/627>

⁵²⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/626>

⁵²⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/601>

⁵²⁸² <https://github.com/STELLAR-GROUP/hpx/issues/557>

⁵²⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/531>

⁵²⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/529>

⁵²⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/526>

⁵²⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/518>

⁵²⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/473>

- Issue #383⁵²⁸⁸ - Change result type detection for hpx::util::bind to use result_of protocol
- Issue #341⁵²⁸⁹ - Consolidate route code
- Issue #219⁵²⁹⁰ - Only copy arguments into actions once
- Issue #177⁵²⁹¹ - Implement distributed AGAS
- Issue #43⁵²⁹² - Support for Darwin (Xcode + Clang)

2.10.21 HPX V0.9.5 (Jan 16, 2013)

We have had over 1000 commits since the last release and we have closed roughly 150 tickets (bugs, feature requests, etc.).

General changes

This release is continuing along the lines of code and API consolidation, and overall usability improvements. We dedicated much attention to performance and we were able to significantly improve the threading and networking subsystems.

We successfully ported *HPX* to the Android platform. *HPX* applications now not only can run on mobile devices, but we support heterogeneous applications running across architecture boundaries. At the Supercomputing Conference 2012 we demonstrated connecting Android tablets to simulations running on a Linux cluster. The Android tablet was used to query performance counters from the Linux simulation and to steer its parameters.

We successfully ported *HPX* to Mac OSX (using the Clang compiler). Thanks to Pyry Jahkola for contributing the corresponding patches. Please see the section `macos_installation` for more details.

We made a special effort to make *HPX* usable in highly concurrent use cases. Many of the *HPX* API functions which possibly take longer than 100 microseconds to execute now can be invoked asynchronously. We added uniform support for composing futures which simplifies to write asynchronous code. *HPX* actions (function objects encapsulating possibly concurrent remote function invocations) are now well integrated with all other API facilities such like `hpx::bind`.

All of the API has been aligned as much as possible with established paradigms. *HPX* now mirrors many of the facilities as defined in the C++11 Standard, such as `hpx::thread`, `hpx::function`, `hpx::future`, etc.

A lot of work has been put into improving the documentation. Many of the API functions are documented now, concepts are explained in detail, and examples are better described than before. The new documentation index enables finding information with lesser effort.

This is the first release of *HPX* we perform after the move to [Github](#)⁵²⁹³. This step has enabled a wider participation from the community and further encourages us in our decision to release *HPX* as a true open source library (*HPX* is licensed under the very liberal [Boost Software License](#)⁵²⁹⁴).

⁵²⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/383>

⁵²⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/341>

⁵²⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/219>

⁵²⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/177>

⁵²⁹² <https://github.com/STELLAR-GROUP/hpx/issues/43>

⁵²⁹³ <https://github.com/STELLAR-GROUP/hpx/>

⁵²⁹⁴ https://www.boost.org/LICENSE_1_0.txt

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release. This is by far the longest list of newly implemented features and fixed issues for any of HPX' releases so far.

- Issue #666⁵²⁹⁵ - Segfault on calling hpx::finalize twice
- Issue #665⁵²⁹⁶ - Adding declaration num_of_cores
- Issue #662⁵²⁹⁷ - pkgconfig is building wrong
- Issue #660⁵²⁹⁸ - Need uninterrupt function
- Issue #659⁵²⁹⁹ - Move our logging library into a different namespace
- Issue #658⁵³⁰⁰ - Dynamic performance counter types are broken
- Issue #657⁵³⁰¹ - HPX v0.9.5 (RC1) hello_world example segfaulting
- Issue #656⁵³⁰² - Define the affinity of parcel-pool, io-pool, and timer-pool threads
- Issue #654⁵³⁰³ - Integrate the Boost auto_index tool with documentation
- Issue #653⁵³⁰⁴ - Make HPX build on OS X + Clang + libc++
- Issue #651⁵³⁰⁵ - Add fine-grained control for thread pinning
- Issue #650⁵³⁰⁶ - Command line no error message when using -hpx:(anything)
- Issue #645⁵³⁰⁷ - Command line aliases don't work in [teletype]` ` @file` ` [c++]
- Issue #644⁵³⁰⁸ - Terminated threads are not always properly cleaned up
- Issue #640⁵³⁰⁹ - future_data<T>::set_on_completed_ used without locks
- Issue #638⁵³¹⁰ - hpx build with intel compilers fails on linux
- Issue #637⁵³¹¹ - --copy-dt-needed-entries breaks with gold
- Issue #635⁵³¹² - Boost V1.53 will add Boost.Lockfree and Boost.Atomic
- Issue #633⁵³¹³ - Re-add examples to final 0.9.5 release
- Issue #632⁵³¹⁴ - Example thread_aware_timer is broken
- Issue #631⁵³¹⁵ - FFT application throws error in parcellayer

⁵²⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/666>

⁵²⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/665>

⁵²⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/662>

⁵²⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/660>

⁵²⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/659>

⁵³⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/658>

⁵³⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/657>

⁵³⁰² <https://github.com/STELLAR-GROUP/hpx/issues/656>

⁵³⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/654>

⁵³⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/653>

⁵³⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/651>

⁵³⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/650>

⁵³⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/645>

⁵³⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/644>

⁵³⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/640>

⁵³¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/638>

⁵³¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/637>

⁵³¹² <https://github.com/STELLAR-GROUP/hpx/issues/635>

⁵³¹³ <https://github.com/STELLAR-GROUP/hpx/issues/633>

⁵³¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/632>

⁵³¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/631>

- Issue #630⁵³¹⁶ - Event synchronization example is broken
- Issue #629⁵³¹⁷ - Waiting on futures hangs
- Issue #628⁵³¹⁸ - Add an HPX_ALWAYS_ASSERT macro
- Issue #625⁵³¹⁹ - Port coroutines context switch benchmark
- Issue #621⁵³²⁰ - New INI section for stack sizes
- Issue #618⁵³²¹ - pkg_config support does not work with a HPX debug build
- Issue #617⁵³²² - hpx/external/logging/boost/logging/detail/cache_before_init.hpp:139:67: error: ‘get_thread_id’ was not declared in this scope
- Issue #616⁵³²³ - Change wait_xxx not to use locking
- Issue #615⁵³²⁴ - Revert visibility ‘fix’ (fb0b6b8245dad1127b0c25ebafd9386b3945cca9)
- Issue #614⁵³²⁵ - Fix Dataflow linker error
- Issue #613⁵³²⁶ - find_here should throw an exception on failure
- Issue #612⁵³²⁷ - Thread phase doesn’t show up in debug mode
- Issue #611⁵³²⁸ - Make stack guard pages configurable at runtime (initialization time)
- Issue #610⁵³²⁹ - Co-Locate Components
- Issue #609⁵³³⁰ - future_overhead
- Issue #608⁵³³¹ - --hpx:list-counter-infos problem
- Issue #607⁵³³² - Update Boost.Context based backend for coroutines
- Issue #606⁵³³³ - 1d_wave_equation is not working
- Issue #605⁵³³⁴ - Any C++ function that has serializable arguments and a serializable return type should be remotable
- Issue #604⁵³³⁵ - Connecting localities isn’t working anymore
- Issue #603⁵³³⁶ - Do not verify any ini entries read from a file
- Issue #602⁵³³⁷ - Rename argument_size to type_size/ added implementation to get parcel size

⁵³¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/630>

⁵³¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/629>

⁵³¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/628>

⁵³¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/625>

⁵³²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/621>

⁵³²¹ <https://github.com/STELLAR-GROUP/hpx/issues/618>

⁵³²² <https://github.com/STELLAR-GROUP/hpx/issues/617>

⁵³²³ <https://github.com/STELLAR-GROUP/hpx/issues/616>

⁵³²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/615>

⁵³²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/614>

⁵³²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/613>

5327 <https://github.com/STELLAR-GROUP/hpx/issues/612>5328 <https://github.com/STELLAR-GROUP/hpx/issues/611>5329 <https://github.com/STELLAR-GROUP/hpx/issues/610>5330 <https://github.com/STELLAR-GROUP/hpx/issues/609>5331 <https://github.com/STELLAR-GROUP/hpx/issues/608>5332 <https://github.com/STELLAR-GROUP/hpx/issues/607>5333 <https://github.com/STELLAR-GROUP/hpx/issues/606>5334 <https://github.com/STELLAR-GROUP/hpx/issues/605>5335 <https://github.com/STELLAR-GROUP/hpx/issues/604>5336 <https://github.com/STELLAR-GROUP/hpx/issues/603>5337 <https://github.com/STELLAR-GROUP/hpx/issues/602>

- Issue #599⁵³³⁸ - Enable locality specific command line options
- Issue #598⁵³³⁹ - Need an API that accesses the performance counter reporting the system uptime
- Issue #597⁵³⁴⁰ - compiling on ranger
- Issue #595⁵³⁴¹ - I need a place to store data in a thread self pointer
- Issue #594⁵³⁴² - 32/64 interoperability
- Issue #593⁵³⁴³ - Warn if logging is disabled at compile time but requested at runtime
- Issue #592⁵³⁴⁴ - Add optional argument value to --hpx:list-counters and --hpx:list-counter-infos
- Issue #591⁵³⁴⁵ - Allow for wildcards in performance counter names specified with --hpx:print-counter
- Issue #590⁵³⁴⁶ - Local promise semantic differences
- Issue #589⁵³⁴⁷ - Create API to query performance counter names
- Issue #587⁵³⁴⁸ - Add get_num_localities and get_num_threads to AGAS API
- Issue #586⁵³⁴⁹ - Adjust local AGAS cache size based on number of localities
- Issue #585⁵³⁵⁰ - Error while using counters in HPX
- Issue #584⁵³⁵¹ - counting argument size of actions, initial pass.
- Issue #581⁵³⁵² - Remove RemoteResult template parameter for future<>
- Issue #580⁵³⁵³ - Add possibility to hook into actions
- Issue #578⁵³⁵⁴ - Use angle brackets in HPX error dumps
- Issue #576⁵³⁵⁵ - Exception incorrectly thrown when --help is used
- Issue #575⁵³⁵⁶ - HPX(bad_component_type) with gcc 4.7.2 and boost 1.51
- Issue #574⁵³⁵⁷ - --hpx:connect command line parameter not working correctly
- Issue #571⁵³⁵⁸ - hpx::wait () (callback version) should pass the future to the callback function
- Issue #570⁵³⁵⁹ - hpx::wait should operate on boost::arrays and std::lists
- Issue #569⁵³⁶⁰ - Add a logging sink for Android

⁵³³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/599>

⁵³³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/598>

⁵³⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/597>

⁵³⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/595>

⁵³⁴² <https://github.com/STELLAR-GROUP/hpx/issues/594>

⁵³⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/593>

⁵³⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/592>

⁵³⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/591>

⁵³⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/590>

⁵³⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/589>

⁵³⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/587>

⁵³⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/586>

⁵³⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/585>

⁵³⁵¹ <https://github.com/STELLAR-GROUP/hpx/issues/584>

⁵³⁵² <https://github.com/STELLAR-GROUP/hpx/issues/581>

⁵³⁵³ <https://github.com/STELLAR-GROUP/hpx/issues/580>

⁵³⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/578>

⁵³⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/576>

⁵³⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/575>

⁵³⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/574>

⁵³⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/571>

⁵³⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/570>

⁵³⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/569>

- Issue #568⁵³⁶¹ - 2-argument version of HPX_DEFINE_COMPONENT_ACTION
- Issue #567⁵³⁶² - Connecting to a running HPX application works only once
- Issue #565⁵³⁶³ - HPX doesn't shutdown properly
- Issue #564⁵³⁶⁴ - Partial preprocessing of new component creation interface
- Issue #563⁵³⁶⁵ - Add hpx::start/hpx::stop to avoid blocking main thread
- Issue #562⁵³⁶⁶ - All command line arguments swallowed by hpx
- Issue #561⁵³⁶⁷ - Boost.Tuple is not move aware
- Issue #558⁵³⁶⁸ - boost::shared_ptr<> style semantics/syntax for client classes
- Issue #556⁵³⁶⁹ - Creation of partially preprocessed headers should be enabled for Boost newer than V1.50
- Issue #555⁵³⁷⁰ - BOOST_FORCEINLINE does not name a type
- Issue #554⁵³⁷¹ - Possible race condition in thread get_id()
- Issue #552⁵³⁷² - Move enable client_base
- Issue #550⁵³⁷³ - Add stack size category 'huge'
- Issue #549⁵³⁷⁴ - ShenEOS run seg-faults on single or distributed runs
- Issue #545⁵³⁷⁵ - AUTOLOB broken for add_hpx_component
- Issue #542⁵³⁷⁶ - FindHPX_HDF5 still searches multiple times
- Issue #541⁵³⁷⁷ - Quotes around application name in hpx::init
- Issue #539⁵³⁷⁸ - Race condition occurring with new lightweight threads
- Issue #535⁵³⁷⁹ - hpx_run_tests.py exits with no error code when tests are missing
- Issue #530⁵³⁸⁰ - Thread description(<unknown>) in logs
- Issue #523⁵³⁸¹ - Make thread objects more lightweight
- Issue #521⁵³⁸² - hpx::error_code is not usable for lightweight error handling
- Issue #520⁵³⁸³ - Add full user environment to HPX logs

⁵³⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/568>

⁵³⁶² <https://github.com/STELLAR-GROUP/hpx/issues/567>

⁵³⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/565>

⁵³⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/564>

⁵³⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/563>

⁵³⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/562>

⁵³⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/561>

⁵³⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/558>

⁵³⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/556>

⁵³⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/555>

⁵³⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/554>

⁵³⁷² <https://github.com/STELLAR-GROUP/hpx/issues/552>

⁵³⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/550>

⁵³⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/549>

⁵³⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/545>

⁵³⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/542>

⁵³⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/541>

⁵³⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/539>

⁵³⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/535>

⁵³⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/530>

⁵³⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/523>

⁵³⁸² <https://github.com/STELLAR-GROUP/hpx/issues/521>

⁵³⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/520>

- Issue #519⁵³⁸⁴ - Build succeeds, running fails
- Issue #517⁵³⁸⁵ - Add a guard page to linux coroutine stacks
- Issue #516⁵³⁸⁶ - hpx::thread::detach suspends while holding locks, leads to hang in debug
- Issue #514⁵³⁸⁷ - Preprocessed headers for <hpx/apply.hpp> don't compile
- Issue #513⁵³⁸⁸ - Buildbot configuration problem
- Issue #512⁵³⁸⁹ - Implement action based stack size customization
- Issue #511⁵³⁹⁰ - Move action priority into a separate type trait
- Issue #510⁵³⁹¹ - trunk broken
- Issue #507⁵³⁹² - no matching function for call to boost::scoped_ptr<hpx::threads::topology>::scoped_ptr(hpx::threads::topology*)
- Issue #505⁵³⁹³ - undefined_symbol regression test currently failing
- Issue #502⁵³⁹⁴ - Adding OpenCL and OCLM support to HPX for Windows and Linux
- Issue #501⁵³⁹⁵ - find_package(HPX) sets cmake output variables
- Issue #500⁵³⁹⁶ - wait_any/wait_all are badly named
- Issue #499⁵³⁹⁷ - Add support for disabling pbs support in pbs runs
- Issue #498⁵³⁹⁸ - Error during no-cache runs
- Issue #496⁵³⁹⁹ - Add partial preprocessing support to cmake
- Issue #495⁵⁴⁰⁰ - Support HPX modules exporting startup/shutdown functions only
- Issue #494⁵⁴⁰¹ - Allow modules to specify when to run startup/shutdown functions
- Issue #493⁵⁴⁰² - Avoid constructing a string in make_success_code
- Issue #492⁵⁴⁰³ - Performance counter creation is no longer synchronized at startup
- Issue #491⁵⁴⁰⁴ - Performance counter creation is no longer synchronized at startup
- Issue #490⁵⁴⁰⁵ - Sheneos on_completed_bulk seg fault in distributed
- Issue #489⁵⁴⁰⁶ - compiling issue with g++44

⁵³⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/519>

⁵³⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/517>

⁵³⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/516>

⁵³⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/514>

⁵³⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/513>

⁵³⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/512>

⁵³⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/511>

⁵³⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/510>

⁵³⁹² <https://github.com/STELLAR-GROUP/hpx/issues/507>

⁵³⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/505>

⁵³⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/502>

⁵³⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/501>

⁵³⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/500>

⁵³⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/499>

5398 <https://github.com/STELLAR-GROUP/hpx/issues/498>5399 <https://github.com/STELLAR-GROUP/hpx/issues/496>5400 <https://github.com/STELLAR-GROUP/hpx/issues/495>5401 <https://github.com/STELLAR-GROUP/hpx/issues/494>5402 <https://github.com/STELLAR-GROUP/hpx/issues/493>5403 <https://github.com/STELLAR-GROUP/hpx/issues/492>5404 <https://github.com/STELLAR-GROUP/hpx/issues/491>5405 <https://github.com/STELLAR-GROUP/hpx/issues/490>5406 <https://github.com/STELLAR-GROUP/hpx/issues/489>

- Issue #488⁵⁴⁰⁷ - Adding OpenCL and OCLM support to HPX for the MSVC platform
- Issue #487⁵⁴⁰⁸ - FindHPX.cmake problems
- Issue #485⁵⁴⁰⁹ - Change distributing_factory and binpacking_factory to use bulk creation
- Issue #484⁵⁴¹⁰ - Change HPX_DONT_USE_PREPROCESSED_FILES to HPX_USE_PREPROCESSED_FILES
- Issue #483⁵⁴¹¹ - Memory counter for Windows
- Issue #479⁵⁴¹² - strange errors appear when requesting performance counters on multiple nodes
- Issue #477⁵⁴¹³ - Create (global) timer for multi-threaded measurements
- Issue #472⁵⁴¹⁴ - Add partial preprocessing using Wave
- Issue #471⁵⁴¹⁵ - Segfault stack traces don't show up in release
- Issue #468⁵⁴¹⁶ - External projects need to link with internal components
- Issue #462⁵⁴¹⁷ - Startup/shutdown functions are called more than once
- Issue #458⁵⁴¹⁸ - Consolidate hpx::util::high_resolution_timer and hpx::util::high_resolution_clock
- Issue #457⁵⁴¹⁹ - index out of bounds in allgather_and_gate on 4 cores or more
- Issue #448⁵⁴²⁰ - Make HPX compile with clang
- Issue #447⁵⁴²¹ - 'make tests' should execute tests on local installation
- Issue #446⁵⁴²² - Remove SVN-related code from the codebase
- Issue #444⁵⁴²³ - race condition in smp
- Issue #441⁵⁴²⁴ - Patched Boost.Serialization headers should only be installed if needed
- Issue #439⁵⁴²⁵ - Components using HPX_REGISTER_STARTUP_MODULE fail to compile with MSVC
- Issue #436⁵⁴²⁶ - Verify that no locks are being held while threads are suspended
- Issue #435⁵⁴²⁷ - Installing HPX should not clobber existing Boost installation
- Issue #434⁵⁴²⁸ - Logging external component failed (Boost 1.50)
- Issue #433⁵⁴²⁹ - Runtime crash when building all examples

5407 <https://github.com/STELLAR-GROUP/hpx/issues/488>

5408 <https://github.com/STELLAR-GROUP/hpx/issues/487>

5409 <https://github.com/STELLAR-GROUP/hpx/issues/485>

5410 <https://github.com/STELLAR-GROUP/hpx/issues/484>

5411 <https://github.com/STELLAR-GROUP/hpx/issues/483>

5412 <https://github.com/STELLAR-GROUP/hpx/issues/479>

5413 <https://github.com/STELLAR-GROUP/hpx/issues/477>

5414 <https://github.com/STELLAR-GROUP/hpx/issues/472>

5415 <https://github.com/STELLAR-GROUP/hpx/issues/471>

5416 <https://github.com/STELLAR-GROUP/hpx/issues/468>

5417 <https://github.com/STELLAR-GROUP/hpx/issues/462>

5418 <https://github.com/STELLAR-GROUP/hpx/issues/458>

5419 <https://github.com/STELLAR-GROUP/hpx/issues/457>

5420 <https://github.com/STELLAR-GROUP/hpx/issues/448>

5421 <https://github.com/STELLAR-GROUP/hpx/issues/447>

5422 <https://github.com/STELLAR-GROUP/hpx/issues/446>

5423 <https://github.com/STELLAR-GROUP/hpx/issues/444>

5424 <https://github.com/STELLAR-GROUP/hpx/issues/441>

5425 <https://github.com/STELLAR-GROUP/hpx/issues/439>

5426 <https://github.com/STELLAR-GROUP/hpx/issues/436>

5427 <https://github.com/STELLAR-GROUP/hpx/issues/435>

5428 <https://github.com/STELLAR-GROUP/hpx/issues/434>

5429 <https://github.com/STELLAR-GROUP/hpx/issues/433>

- Issue #432⁵⁴³⁰ - Dataflow hangs on 512 cores/64 nodes
- Issue #430⁵⁴³¹ - Problem with distributing factory
- Issue #424⁵⁴³² - File paths referring to XSL-files need to be properly escaped
- Issue #417⁵⁴³³ - Make dataflow LCOs work out of the box by using partial preprocessing
- Issue #413⁵⁴³⁴ - hpx_svnversion.py fails on Windows
- Issue #412⁵⁴³⁵ - Make hpx::error_code equivalent to hpx::exception
- Issue #398⁵⁴³⁶ - HPX clobbers out-of-tree application specific CMake variables (specifically CMAKE_BUILD_TYPE)
- Issue #394⁵⁴³⁷ - Remove code generating random port numbers for network
- Issue #378⁵⁴³⁸ - ShenEOS scaling issues
- Issue #354⁵⁴³⁹ - Create a coroutines wrapper for Boost.Context
- Issue #349⁵⁴⁴⁰ - Commandline option --localities=N/-lN should be necessary only on AGAS locality
- Issue #334⁵⁴⁴¹ - Add auto_index support to cmake based documentation toolchain
- Issue #318⁵⁴⁴² - Network benchmarks
- Issue #317⁵⁴⁴³ - Implement network performance counters
- Issue #310⁵⁴⁴⁴ - Duplicate logging entries
- Issue #230⁵⁴⁴⁵ - Add compile time option to disable thread debugging info
- Issue #171⁵⁴⁴⁶ - Add an INI option to turn off deadlock detection independently of logging
- Issue #170⁵⁴⁴⁷ - OSHL internal counters are incorrect
- Issue #103⁵⁴⁴⁸ - Better diagnostics for multiple component/action registrations under the same name
- Issue #48⁵⁴⁴⁹ - Support for Darwin (Xcode + Clang)
- Issue #21⁵⁴⁵⁰ - Build fails with GCC 4.6

⁵⁴³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/432>

⁵⁴³¹ <https://github.com/STELLAR-GROUP/hpx/issues/430>

⁵⁴³² <https://github.com/STELLAR-GROUP/hpx/issues/424>

⁵⁴³³ <https://github.com/STELLAR-GROUP/hpx/issues/417>

⁵⁴³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/413>

⁵⁴³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/412>

⁵⁴³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/398>

⁵⁴³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/394>

⁵⁴³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/378>

⁵⁴³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/354>

⁵⁴⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/349>

⁵⁴⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/334>

⁵⁴⁴² <https://github.com/STELLAR-GROUP/hpx/issues/318>

⁵⁴⁴³ <https://github.com/STELLAR-GROUP/hpx/issues/317>

⁵⁴⁴⁴ <https://github.com/STELLAR-GROUP/hpx/issues/310>

⁵⁴⁴⁵ <https://github.com/STELLAR-GROUP/hpx/issues/230>

⁵⁴⁴⁶ <https://github.com/STELLAR-GROUP/hpx/issues/171>

⁵⁴⁴⁷ <https://github.com/STELLAR-GROUP/hpx/issues/170>

⁵⁴⁴⁸ <https://github.com/STELLAR-GROUP/hpx/issues/103>

⁵⁴⁴⁹ <https://github.com/STELLAR-GROUP/hpx/issues/48>

⁵⁴⁵⁰ <https://github.com/STELLAR-GROUP/hpx/issues/21>

2.10.22 HPX V0.9.0 (Jul 5, 2012)

We have had roughly 800 commits since the last release and we have closed approximately 80 tickets (bugs, feature requests, etc.).

General changes

- Significant improvements made to the usability of *HPX* in large-scale, distributed environments.
- Renamed `hpx::lcos::packaged_task` to `hpx::lcos::packaged_action` to reflect the semantic differences to a `packaged_task` as defined by the C++11 Standard⁵⁴⁵¹.
- *HPX* now exposes `hpx::thread` which is compliant to the C++11 `std::thread` type except that it (purely locally) represents an *HPX* thread. This new type does not expose any of the remote capabilities of the underlying *HPX*-thread implementation.
- The type `hpx::lcos::future` is now compliant to the C++11 `std::future<>` type. This type can be used to synchronize both, local and remote operations. In both cases the control flow will ‘return’ to the future in order to trigger any continuation.
- The types `hpx::lcos::local::promise` and `hpx::lcos::local::packaged_task` are now compliant to the C++11 `std::promise<>` and `std::packaged_task<>` types. These can be used to create a future representing local work only. Use the types `hpx::lcos::promise` and `hpx::lcos::packaged_action` to wrap any (possibly remote) action into a future.
- `hpx::thread` and `hpx::lcos::future` are now cancelable.
- Added support for sequential and logic composition of `hpx::lcos::futures`. The member function `hpx::lcos::future::when` permits futures to be sequentially composed. The helper functions `hpx::wait_all`, `hpx::wait_any`, and `hpx::wait_n` can be used to wait for more than one future at a time.
- *HPX* now exposes `hpx::apply` and `hpx::async` as the preferred way of creating (or invoking) any deferred work. These functions are usable with various types of functions, function objects, and actions and provide a uniform way to spawn deferred tasks.
- *HPX* now utilizes `hpx::util::bind` to (partially) bind local functions and function objects, and also actions. Remote bound actions can have placeholders as well.
- *HPX* continuations are now fully polymorphic. The class `hpx::actions::forwarding_continuation` is an example of how the user can write their own types of continuations. It can be used to execute any function as an continuation of a particular action.
- Reworked the action invocation API to be fully conformant to normal functions. Actions can now be invoked using `hpx::apply`, `hpx::async`, or using the operator `()` implemented on actions. Actions themselves can now be cheaply instantiated as they do not have any members anymore.
- Reworked the lazy action invocation API. Actions can now be directly bound using `hpx::util::bind` by passing an action instance as the first argument.
- A minimal *HPX* program now looks like this:

```
#include <hpx/hpx_init.hpp>

int hpx_main()
{
    return hpx::finalize();
}
```

(continues on next page)

⁵⁴⁵¹ <http://www.open-std.org/jtc1/sc22/wg21>

(continued from previous page)

```

}

int main()
{
    return hpx::init();
}

```

This removes the immediate dependency on the [Boost.Program Options⁵⁴⁵²](#) library.

Note: This minimal version of an *HPX* program does not support any of the default command line arguments (such as –help, or command line options related to PBS). It is suggested to always pass `argc` and `argv` to *HPX* as shown in the example below.

- In order to support those, but still not to depend on [Boost.Program Options⁵⁴⁵³](#), the minimal program can be written as:

```

#include <hpx/hpx_init.hpp>

// The arguments for hpx_main can be left off, which very similar to the
// behavior of ``main()`` as defined by C++.
int hpx_main(int argc, char* argv[])
{
    return hpx::finalize();
}

int main(int argc, char* argv[])
{
    return hpx::init(argc, argv);
}

```

- Added performance counters exposing the number of component instances which are alive on a given locality.
- Added performance counters exposing then number of messages sent and received, the number of parcels sent and received, the number of bytes sent and received, the overall time required to send and receive data, and the overall time required to serialize and deserialize the data.
- Added a new component: `hpx::components::binpacking_factory` which is equivalent to the existing `hpx::components::distributing_factory` component, except that it equalizes the overall population of the components to create. It exposes two factory methods, one based on the number of existing instances of the component type to create, and one based on an arbitrary performance counter which will be queried for all relevant localities.
- Added API functions allowing to access elements of the diagnostic information embedded in the given exception: `hpx::get_locality_id`, `hpx::get_host_name`, `hpx::get_process_id`, `hpx::get_function_name`, `hpx::get_file_name`, `hpx::get_line_number`, `hpx::get_os_thread`, `hpx::get_thread_id`, and `hpx::get_thread_description`.

⁵⁴⁵² https://www.boost.org/doc/html/program_options.html

⁵⁴⁵³ https://www.boost.org/doc/html/program_options.html

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release:

- Issue #71⁵⁴⁵⁴ - GIDs that are not serialized via `handle_gid<>` should raise an exception
- Issue #105⁵⁴⁵⁵ - Allow for `hpx::util::functions` to be registered in the AGAS symbolic namespace
- Issue #107⁵⁴⁵⁶ - Nasty threadmanger race condition (reproducible in `sheneos_test`)
- Issue #108⁵⁴⁵⁷ - Add millisecond resolution to *HPX* logs on Linux
- Issue #110⁵⁴⁵⁸ - Shutdown hang in distributed with release build
- Issue #116⁵⁴⁵⁹ - Don't use TSS for the applier and runtime pointers
- Issue #162⁵⁴⁶⁰ - Move local synchronous execution shortcut from `hpx::function` to the applier
- Issue #172⁵⁴⁶¹ - Cache sources in CMake and check if they change manually
- Issue #178⁵⁴⁶² - Add an INI option to turn off ranged-based AGAS caching
- Issue #187⁵⁴⁶³ - Support for disabling performance counter deployment
- Issue #202⁵⁴⁶⁴ - Support for sending performance counter data to a specific file
- Issue #218⁵⁴⁶⁵ - `boost.coroutines` allows different stack sizes, but stack pool is unaware of this
- Issue #231⁵⁴⁶⁶ - Implement movable `boost::bind`
- Issue #232⁵⁴⁶⁷ - Implement movable `boost::function`
- Issue #236⁵⁴⁶⁸ - Allow binding `hpx::util::function` to actions
- Issue #239⁵⁴⁶⁹ - Replace `hpx::function` with `hpx::util::function`
- Issue #240⁵⁴⁷⁰ - Can't specify `RemoteResult` with `Icos::async`
- Issue #242⁵⁴⁷¹ - `REGISTER_TEMPLATE` support for plain actions
- Issue #243⁵⁴⁷² - `handle_gid<>` support for `hpx::util::function`
- Issue #245⁵⁴⁷³ - `*_c_cache` code throws an exception if the queried GID is not in the local cache
- Issue #246⁵⁴⁷⁴ - Undefined references in dataflow/adaptive1d example

⁵⁴⁵⁴ <https://github.com/STELLAR-GROUP/hpx/issues/71>

⁵⁴⁵⁵ <https://github.com/STELLAR-GROUP/hpx/issues/105>

⁵⁴⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/107>

⁵⁴⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/108>

⁵⁴⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/110>

⁵⁴⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/116>

⁵⁴⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/162>

⁵⁴⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/172>

⁵⁴⁶² <https://github.com/STELLAR-GROUP/hpx/issues/178>

5463 <https://github.com/STELLAR-GROUP/hpx/issues/187>5464 <https://github.com/STELLAR-GROUP/hpx/issues/202>5465 <https://github.com/STELLAR-GROUP/hpx/issues/218>5466 <https://github.com/STELLAR-GROUP/hpx/issues/231>5467 <https://github.com/STELLAR-GROUP/hpx/issues/232>5468 <https://github.com/STELLAR-GROUP/hpx/issues/236>5469 <https://github.com/STELLAR-GROUP/hpx/issues/239>5470 <https://github.com/STELLAR-GROUP/hpx/issues/240>5471 <https://github.com/STELLAR-GROUP/hpx/issues/242>5472 <https://github.com/STELLAR-GROUP/hpx/issues/243>5473 <https://github.com/STELLAR-GROUP/hpx/issues/245>5474 <https://github.com/STELLAR-GROUP/hpx/issues/246>

- Issue #252⁵⁴⁷⁵ - Problems configuring sheneos with CMake
- Issue #254⁵⁴⁷⁶ - Lifetime of components doesn't end when client goes out of scope
- Issue #259⁵⁴⁷⁷ - CMake does not detect that MSVC10 has lambdas
- Issue #260⁵⁴⁷⁸ - io_service_pool segfault
- Issue #261⁵⁴⁷⁹ - Late parcel executed outside of pthead
- Issue #263⁵⁴⁸⁰ - Cannot select allocator with CMake
- Issue #264⁵⁴⁸¹ - Fix allocator select
- Issue #267⁵⁴⁸² - Runtime error for hello_world
- Issue #269⁵⁴⁸³ - pthread_affinity_np test fails to compile
- Issue #270⁵⁴⁸⁴ - Compiler noise due to -Wcast-qual
- Issue #275⁵⁴⁸⁵ - Problem with configuration tests/include paths on Gentoo
- Issue #325⁵⁴⁸⁶ - Sheneos is 200-400 times slower than the fortran equivalent
- Issue #331⁵⁴⁸⁷ - `hpx::init` and `hpx_main()` should not depend on `program_options`
- Issue #333⁵⁴⁸⁸ - Add doxygen support to CMake for doc toolchain
- Issue #340⁵⁴⁸⁹ - Performance counters for parcels
- Issue #346⁵⁴⁹⁰ - Component loading error when running hello_world in distributed on MSVC2010
- Issue #362⁵⁴⁹¹ - Missing initializer error
- Issue #363⁵⁴⁹² - Parcel port serialization error
- Issue #366⁵⁴⁹³ - Parcel buffering leads to types incompatible exception
- Issue #368⁵⁴⁹⁴ - Scalable alternative to rand() needed for *HPX*
- Issue #369⁵⁴⁹⁵ - IB over IP is substantially slower than just using standard TCP/IP
- Issue #374⁵⁴⁹⁶ - `hpx::lcos::wait` should work with dataflows and arbitrary classes meeting the future interface
- Issue #375⁵⁴⁹⁷ - Conflicting/ambiguous overloads of `hpx::lcos::wait`

⁵⁴⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/252>

⁵⁴⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/254>

⁵⁴⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/259>

⁵⁴⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/260>

⁵⁴⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/261>

⁵⁴⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/263>

⁵⁴⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/264>

⁵⁴⁸² <https://github.com/STELLAR-GROUP/hpx/issues/267>

⁵⁴⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/269>

⁵⁴⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/270>

⁵⁴⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/275>

⁵⁴⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/325>

⁵⁴⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/331>

⁵⁴⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/333>

⁵⁴⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/340>

⁵⁴⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/346>

⁵⁴⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/362>

⁵⁴⁹² <https://github.com/STELLAR-GROUP/hpx/issues/363>

⁵⁴⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/366>

⁵⁴⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/368>

⁵⁴⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/369>

⁵⁴⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/374>

⁵⁴⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/375>

- Issue #376⁵⁴⁹⁸ - Find_HPX.cmake should set CMake variable HPX_FOUND for out of tree builds
- Issue #377⁵⁴⁹⁹ - ShenEOS interpolate bulk and interpolate_one_bulk are broken
- Issue #379⁵⁵⁰⁰ - Add support for distributed runs under SLURM
- Issue #382⁵⁵⁰¹ - _Unwind_Word not declared in boost.backtrace
- Issue #387⁵⁵⁰² - Doxygen should look only at list of specified files
- Issue #388⁵⁵⁰³ - Running make install on an out-of-tree application is broken
- Issue #391⁵⁵⁰⁴ - Out-of-tree application segfaults when running in qsub
- Issue #392⁵⁵⁰⁵ - Remove HPX_NO_INSTALL option from cmake build system
- Issue #396⁵⁵⁰⁶ - Pragma related warnings when compiling with older gcc versions
- Issue #399⁵⁵⁰⁷ - Out of tree component build problems
- Issue #400⁵⁵⁰⁸ - Out of source builds on Windows: linker should not receive compiler flags
- Issue #401⁵⁵⁰⁹ - Out of source builds on Windows: components need to be linked with hpx_serialization
- Issue #404⁵⁵¹⁰ - gfortran fails to link automatically when fortran files are present
- Issue #405⁵⁵¹¹ - Inability to specify linking order for external libraries
- Issue #406⁵⁵¹² - Adapt action limits such that dataflow applications work without additional defines
- Issue #415⁵⁵¹³ - locality_results is not a member of hpx::components::server
- Issue #425⁵⁵¹⁴ - Breaking changes to traits::*result wrt std::vector<id_type>
- Issue #426⁵⁵¹⁵ - AUTOLOB needs to be updated to support fortran

2.10.23 HPX V0.8.1 (Apr 21, 2012)

This is a point release including important bug fixes for *HPX V0.8.0 (Mar 23, 2012)*.

⁵⁴⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/376>

⁵⁴⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/377>

⁵⁵⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/379>

⁵⁵⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/382>

⁵⁵⁰² <https://github.com/STELLAR-GROUP/hpx/issues/387>

⁵⁵⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/388>

⁵⁵⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/391>

⁵⁵⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/392>

⁵⁵⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/396>

⁵⁵⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/399>

5508 <https://github.com/STELLAR-GROUP/hpx/issues/400>5509 <https://github.com/STELLAR-GROUP/hpx/issues/401>5510 <https://github.com/STELLAR-GROUP/hpx/issues/404>5511 <https://github.com/STELLAR-GROUP/hpx/issues/405>5512 <https://github.com/STELLAR-GROUP/hpx/issues/406>5513 <https://github.com/STELLAR-GROUP/hpx/issues/415>5514 <https://github.com/STELLAR-GROUP/hpx/issues/425>5515 <https://github.com/STELLAR-GROUP/hpx/issues/426>

General changes

- HPX does not need to be installed anymore to be functional.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this point release:

- Issue #295⁵⁵¹⁶ - Don't require install path to be known at compile time.
- Issue #371⁵⁵¹⁷ - Add hpx iostreams to standard build.
- Issue #384⁵⁵¹⁸ - Fix compilation with GCC 4.7.
- Issue #390⁵⁵¹⁹ - Remove keep_factory_alive startup call from ShenEOS; add shutdown call to H5close.
- Issue #393⁵⁵²⁰ - Thread affinity control is broken.

Bug fixes (commits)

Here is a list of the important commits included in this point release:

- r7642 - External: Fix backtrace memory violation.
- **r7775 - Components: Fix symbol visibility bug with component startup** providers. This prevents one components providers from overriding another components.
- r7778 - Components: Fix startup/shutdown provider shadowing issues.

2.10.24 HPX V0.8.0 (Mar 23, 2012)

We have had roughly 1000 commits since the last release and we have closed approximately 70 tickets (bugs, feature requests, etc.).

General changes

- Improved PBS support, allowing for arbitrary naming schemes of node-hostnames.
- Finished verification of the reference counting framework.
- Implemented decrement merging logic to optimize the distributed reference counting system.
- Restructured the LCO framework. Renamed `hpx::lcos::eager_future<>` and `hpx::lcos::lazy_future<>` into `hpx::lcos::packaged_task` and `hpx::lcos::deferred_packaged_task`. Split `hpx::lcos::promise` into `hpx::lcos::packaged_task` and `hpx::lcos::future`. Added 'local' futures (in namespace `hpx::lcos::local`).
- Improved the general performance of local and remote action invocations. This (under certain circumstances) drastically reduces the number of copies created for each of the parameters and return values.

⁵⁵¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/295>

⁵⁵¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/371>

⁵⁵¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/384>

⁵⁵¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/390>

⁵⁵²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/393>

- Reworked the performance counter framework. Performance counters are now created only when needed, which reduces the overall resource requirements. The new framework allows for much more flexible creation and management of performance counters. The new sine example application demonstrates some of the capabilities of the new infrastructure.
- Added a buildbot-based continuous build system which gives instant, automated feedback on each commit to SVN.
- Added more automated tests to verify proper functioning of *HPX*.
- Started to create documentation for *HPX* and its API.
- Added documentation toolchain to the build system.
- Added dataflow LCO.
- Changed default *HPX* command line options to have `hpx:` prefix. For instance, the former option `--threads` is now `--hpx:threads`. This has been done to make ambiguities with possible application specific command line options as unlikely as possible. See the section [HPX Command Line Options](#) for a full list of available options.
- Added the possibility to define command line aliases. The former short (one-letter) command line options have been predefined as aliases for backwards compatibility. See the section [HPX Command Line Options](#) for a detailed description of command line option aliasing.
- Network connections are now cached based on the connected host. The number of simultaneous connections to a particular host is now limited. Parcels are buffered and bundled if all connections are in use.
- Added more refined thread affinity control. This is based on the external library Portable Hardware Locality (HWLOC).
- Improved support for Windows builds with CMake.
- Added support for components to register their own command line options.
- Added the possibility to register custom startup/shutdown functions for any component. These functions are guaranteed to be executed by an *HPX* thread.
- Added two new experimental thread schedulers: `hierarchy_scheduler` and `periodic_priority_scheduler`. These can be activated by using the command line options `--hpx:queuing=hierarchy` or `--hpx:queuing=periodic`.

Example applications

- Graph500 performance benchmark⁵⁵²¹ (thanks to Matthew Anderson for contributing this application).
- GTC (Gyrokinetic Toroidal Code)⁵⁵²²: a skeleton for particle in cell type codes.
- Random Memory Access: an example demonstrating random memory accesses in a large array
- ShenEOS example⁵⁵²³, demonstrating partitioning of large read-only data structures and exposing an interpolation API.
- Sine performance counter demo.
- Accumulator examples demonstrating how to write and use *HPX* components.
- Quickstart examples (like `hello_world`, `fibonacci`, `quicksort`, `factorial`, etc.) demonstrating simple *HPX* concepts which introduce some of the concepts in *HPX*.

⁵⁵²¹ <http://www.graph500.org/>

⁵⁵²² <http://www.nersc.gov/research-and-development/benchmarking-and-workload-characterization/nersc-6-benchmarks/gtc/>

⁵⁵²³ <http://stellarcollapse.org/equationofstate>

- Load balancing and work stealing demos.

API changes

- Moved all local LCOs into a separate namespace `hpx::lcos::local` (for instance, `hpx::lcos::local_mutex` is now `hpx::lcos::local::mutex`).
- Replaced `hpx::actions::function` with `hpx::util::function`. Cleaned up related code.
- Removed `hpx::traits::handle_gid` and moved handling of global reference counts into the corresponding serialization code.
- Changed terminology: `prefix` is now called `locality_id`, renamed the corresponding API functions (such as `hpx::get_prefix`, which is now called `hpx::get_locality_id`).
- Adding `hpx::find_remote_localities`, and `hpx::get_num_localities`.
- Changed performance counter naming scheme to make it more bash friendly. The new performance counter naming scheme is now

```
/object{parentname#parentindex/instance#index}/counter#parameters
```

- Added `hpx::get_worker_thread_num` replacing `hpx::threadmanager_base::get_thread_num`.
- Renamed `hpx::get_num_os_threads` to `hpx::get_os_threads_count`.
- Added `hpx::threads::get_thread_count`.
- Restructured the Futures sub-system, renaming types in accordance with the terminology used by the C++11 ISO standard.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release:

- Issue #31^{[5524](https://github.com/STELLAR-GROUP/hpx/issues/31)} - Specialize handle_gid<> for examples and tests
- Issue #72^{[5525](https://github.com/STELLAR-GROUP/hpx/issues/72)} - Fix AGAS reference counting
- Issue #104^{[5526](https://github.com/STELLAR-GROUP/hpx/issues/104)} - heartbeat throws an exception when derefing the performance counter it's watching
- Issue #111^{[5527](https://github.com/STELLAR-GROUP/hpx/issues/111)} - throttle causes an exception on the target application
- Issue #142^{[5528](https://github.com/STELLAR-GROUP/hpx/issues/142)} - One failed component loading causes an unrelated component to fail
- Issue #165^{[5529](https://github.com/STELLAR-GROUP/hpx/issues/165)} - Remote exception propagation bug in AGAS reference counting test
- Issue #186^{[5530](https://github.com/STELLAR-GROUP/hpx/issues/186)} - Test credit exhaustion/splitting (e.g. prepare_gid and symbol NS)
- Issue #188^{[5531](https://github.com/STELLAR-GROUP/hpx/issues/188)} - Implement remaining AGAS reference counting test cases
- Issue #258^{[5532](https://github.com/STELLAR-GROUP/hpx/issues/258)} - No type checking of GIDs in stubs classes

⁵⁵²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/31>

⁵⁵²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/72>

⁵⁵²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/104>

⁵⁵²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/111>

⁵⁵²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/142>

⁵⁵²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/165>

⁵⁵³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/186>

⁵⁵³¹ <https://github.com/STELLAR-GROUP/hpx/issues/188>

⁵⁵³² <https://github.com/STELLAR-GROUP/hpx/issues/258>

- Issue #271⁵⁵³³ - Seg fault/shared pointer assertion in distributed code
- Issue #281⁵⁵³⁴ - CMake options need descriptive text
- Issue #283⁵⁵³⁵ - AGAS caching broken (gva_cache needs to be rewritten with ICL)
- Issue #285⁵⁵³⁶ - HPX_INSTALL root directory not the same as CMAKE_INSTALL_PREFIX
- Issue #286⁵⁵³⁷ - New segfault in dataflow applications
- Issue #289⁵⁵³⁸ - Exceptions should only be logged if not handled
- Issue #290⁵⁵³⁹ - c++11 tests failure
- Issue #293⁵⁵⁴⁰ - Build target for component libraries
- Issue #296⁵⁵⁴¹ - Compilation error with Boost V1.49rc1
- Issue #298⁵⁵⁴² - Illegal instructions on termination
- Issue #299⁵⁵⁴³ - gravity aborts with multiple threads
- Issue #301⁵⁵⁴⁴ - Build error with Boost trunk
- Issue #303⁵⁵⁴⁵ - Logging assertion failure in distributed runs
- Issue #304⁵⁵⁴⁶ - Exception ‘what’ strings are lost when exceptions from decode_parcel are reported
- Issue #306⁵⁵⁴⁷ - Performance counter user interface issues
- Issue #307⁵⁵⁴⁸ - Logging exception in distributed runs
- Issue #308⁵⁵⁴⁹ - Logging deadlocks in distributed
- Issue #309⁵⁵⁵⁰ - Reference counting test failures and exceptions
- Issue #311⁵⁵⁵¹ - Merge AGAS remote_interface with the runtime_support object
- Issue #314⁵⁵⁵² - Object tracking for id_types
- Issue #315⁵⁵⁵³ - Remove handle_gid and handle credit splitting in id_type serialization
- Issue #320⁵⁵⁵⁴ - applier::get_locality_id() should return an error value (or throw an exception)
- Issue #321⁵⁵⁵⁵ - Optimization for id_types which are never split should be restored

⁵⁵³³ <https://github.com/STELLAR-GROUP/hpx/issues/271>

⁵⁵³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/281>

⁵⁵³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/283>

⁵⁵³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/285>

⁵⁵³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/286>

⁵⁵³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/289>

⁵⁵³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/290>

⁵⁵⁴⁰ <https://github.com/STELLAR-GROUP/hpx/issues/293>

⁵⁵⁴¹ <https://github.com/STELLAR-GROUP/hpx/issues/296>

5542 <https://github.com/STELLAR-GROUP/hpx/issues/298>5543 <https://github.com/STELLAR-GROUP/hpx/issues/299>5544 <https://github.com/STELLAR-GROUP/hpx/issues/301>5545 <https://github.com/STELLAR-GROUP/hpx/issues/303>5546 <https://github.com/STELLAR-GROUP/hpx/issues/304>5547 <https://github.com/STELLAR-GROUP/hpx/issues/306>5548 <https://github.com/STELLAR-GROUP/hpx/issues/307>5549 <https://github.com/STELLAR-GROUP/hpx/issues/308>5550 <https://github.com/STELLAR-GROUP/hpx/issues/309>5551 <https://github.com/STELLAR-GROUP/hpx/issues/311>5552 <https://github.com/STELLAR-GROUP/hpx/issues/314>5553 <https://github.com/STELLAR-GROUP/hpx/issues/315>5554 <https://github.com/STELLAR-GROUP/hpx/issues/320>5555 <https://github.com/STELLAR-GROUP/hpx/issues/321>

- Issue #322⁵⁵⁵⁶ - Command line processing ignored with Boost 1.47.0
- Issue #323⁵⁵⁵⁷ - Credit exhaustion causes object to stay alive
- Issue #324⁵⁵⁵⁸ - Duplicate exception messages
- Issue #326⁵⁵⁵⁹ - Integrate Quickbook with CMake
- Issue #329⁵⁵⁶⁰ - --help and --version should still work
- Issue #330⁵⁵⁶¹ - Create pkg-config files
- Issue #337⁵⁵⁶² - Improve usability of performance counter timestamps
- Issue #338⁵⁵⁶³ - Non-std exceptions deriving from std::exceptions in tfunc may be sliced
- Issue #339⁵⁵⁶⁴ - Decrease the number of send_pending_parcels threads
- Issue #343⁵⁵⁶⁵ - Dynamically setting the stack size doesn't work
- Issue #351⁵⁵⁶⁶ - 'make install' does not update documents
- Issue #353⁵⁵⁶⁷ - Disable FIXMEs in the docs by default; add a doc developer CMake option to enable FIXMEs
- Issue #355⁵⁵⁶⁸ - 'make' doesn't do anything after correct configuration
- Issue #356⁵⁵⁶⁹ - Don't use hpx::util::static_ in topology code
- Issue #359⁵⁵⁷⁰ - Infinite recursion in hpx::tuple serialization
- Issue #361⁵⁵⁷¹ - Add compile time option to disable logging completely
- Issue #364⁵⁵⁷² - Installation seriously broken in r7443

2.10.25 HPX V0.7.0 (Dec 12, 2011)

We have had roughly 1000 commits since the last release and we have closed approximately 120 tickets (bugs, feature requests, etc.).

⁵⁵⁵⁶ <https://github.com/STELLAR-GROUP/hpx/issues/322>

⁵⁵⁵⁷ <https://github.com/STELLAR-GROUP/hpx/issues/323>

⁵⁵⁵⁸ <https://github.com/STELLAR-GROUP/hpx/issues/324>

⁵⁵⁵⁹ <https://github.com/STELLAR-GROUP/hpx/issues/326>

⁵⁵⁶⁰ <https://github.com/STELLAR-GROUP/hpx/issues/329>

⁵⁵⁶¹ <https://github.com/STELLAR-GROUP/hpx/issues/330>

⁵⁵⁶² <https://github.com/STELLAR-GROUP/hpx/issues/337>

⁵⁵⁶³ <https://github.com/STELLAR-GROUP/hpx/issues/338>

⁵⁵⁶⁴ <https://github.com/STELLAR-GROUP/hpx/issues/339>

⁵⁵⁶⁵ <https://github.com/STELLAR-GROUP/hpx/issues/343>

⁵⁵⁶⁶ <https://github.com/STELLAR-GROUP/hpx/issues/351>

⁵⁵⁶⁷ <https://github.com/STELLAR-GROUP/hpx/issues/353>

⁵⁵⁶⁸ <https://github.com/STELLAR-GROUP/hpx/issues/355>

⁵⁵⁶⁹ <https://github.com/STELLAR-GROUP/hpx/issues/356>

⁵⁵⁷⁰ <https://github.com/STELLAR-GROUP/hpx/issues/359>

⁵⁵⁷¹ <https://github.com/STELLAR-GROUP/hpx/issues/361>

⁵⁵⁷² <https://github.com/STELLAR-GROUP/hpx/issues/364>

General changes

- Completely removed code related to deprecated AGAS V1, started to work on AGAS V2.1.
- Started to clean up and streamline the exposed APIs (see ‘API changes’ below for more details).
- Revamped and unified performance counter framework, added a lot of new performance counter instances for monitoring of a diverse set of internal *HPX* parameters (queue lengths, access statistics, etc.).
- Improved general error handling and logging support.
- Fixed several race conditions, improved overall stability, decreased memory footprint, improved overall performance (major optimizations include native TLS support and ranged-based AGAS caching).
- Added support for running *HPX* applications with PBS.
- Many updates to the build system, added support for gcc 4.5.x and 4.6.x, added C++11 support.
- Many updates to default command line options.
- Added many tests, set up buildbot for continuous integration testing.
- Better shutdown handling of distributed applications.

Example applications

- quickstart/factorial and quickstart/fibonacci, future-recursive parallel algorithms.
- quickstart/hello_world, distributed hello world example.
- quickstart/rma, simple remote memory access example
- quickstart/quicksort, parallel quicksort implementation.
- gtc, gyrokinetic torodial code.
- bfs, breadth-first-search, example code for a graph application.
- sheneos, partitioning of large data sets.
- accumulator, simple component example.
- balancing/os_thread_num, balancing/px_thread_phase, examples demonstrating load balancing and work stealing.

API changes

- Added `hpx::find_all_localities`.
- Added `hpx::terminate` for non-graceful termination of applications.
- Added `hpx::lcos::async` functions for simpler asynchronous programming.
- Added new AGAS interface for handling of symbolic namespace (`hpx::agas::*`).
- Renamed `hpx::components::wait` to `hpx::lcos::wait`.
- Renamed `hpx::lcos::future_value` to `hpx::lcos::promise`.
- Renamed `hpx::lcos::recursive_mutex` to `hpx::lcos::local_recursive_mutex`, `hpx::lcos::mutex` to `hpx::lcos::local_mutex`
- Removed support for Boost versions older than V1.38, recommended Boost version is now V1.47 and newer.
- Removed `hpx::process` (this will be replaced by a real process implementation in the future).

- Removed non-functional LCO code (`hpx::lcos::dataflow`, `hpx::lcos::thunk`, `hpx::lcos::dataflow_variable`).
- Removed deprecated `hpx::naming::full_address`.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release:

- Issue #28⁵⁵⁷³ - Integrate Windows/Linux CMake code for *HPX* core
- Issue #32⁵⁵⁷⁴ - `hpx::cout()` should be `hpx::cout`
- Issue #33⁵⁵⁷⁵ - AGAS V2 legacy client does not properly handle `error_code`
- Issue #60⁵⁵⁷⁶ - AGAS: allow for registerid to optionally take ownership of the gid
- Issue #62⁵⁵⁷⁷ - adaptive1d compilation failure in Fusion
- Issue #64⁵⁵⁷⁸ - Parcel subsystem doesn't resolve domain names
- Issue #83⁵⁵⁷⁹ - No error handling if no console is available
- Issue #84⁵⁵⁸⁰ - No error handling if a hosted locality is treated as the bootstrap server
- Issue #90⁵⁵⁸¹ - Add general commandline option `-N`
- Issue #91⁵⁵⁸² - Add possibility to read command line arguments from file
- Issue #92⁵⁵⁸³ - Always log exceptions/errors to the log file
- Issue #93⁵⁵⁸⁴ - Log the command line/program name
- Issue #95⁵⁵⁸⁵ - Support for distributed launches
- Issue #97⁵⁵⁸⁶ - Attempt to create a bad component type in AMR examples
- Issue #100⁵⁵⁸⁷ - factorial and factorial_get examples trigger AGAS component type assertions
- Issue #101⁵⁵⁸⁸ - Segfault when `hpx::process::here()` is called in fibonacci2
- Issue #102⁵⁵⁸⁹ - unknown_component_address in int_object_semaphore_client
- Issue #114⁵⁵⁹⁰ - marduk raises assertion with default parameters
- Issue #115⁵⁵⁹¹ - Logging messages for SMP runs (on the console) shouldn't be buffered

⁵⁵⁷³ <https://github.com/STELLAR-GROUP/hpx/issues/28>

⁵⁵⁷⁴ <https://github.com/STELLAR-GROUP/hpx/issues/32>

⁵⁵⁷⁵ <https://github.com/STELLAR-GROUP/hpx/issues/33>

⁵⁵⁷⁶ <https://github.com/STELLAR-GROUP/hpx/issues/60>

⁵⁵⁷⁷ <https://github.com/STELLAR-GROUP/hpx/issues/62>

⁵⁵⁷⁸ <https://github.com/STELLAR-GROUP/hpx/issues/64>

⁵⁵⁷⁹ <https://github.com/STELLAR-GROUP/hpx/issues/83>

⁵⁵⁸⁰ <https://github.com/STELLAR-GROUP/hpx/issues/84>

⁵⁵⁸¹ <https://github.com/STELLAR-GROUP/hpx/issues/90>

⁵⁵⁸² <https://github.com/STELLAR-GROUP/hpx/issues/91>

⁵⁵⁸³ <https://github.com/STELLAR-GROUP/hpx/issues/92>

⁵⁵⁸⁴ <https://github.com/STELLAR-GROUP/hpx/issues/93>

⁵⁵⁸⁵ <https://github.com/STELLAR-GROUP/hpx/issues/95>

⁵⁵⁸⁶ <https://github.com/STELLAR-GROUP/hpx/issues/97>

⁵⁵⁸⁷ <https://github.com/STELLAR-GROUP/hpx/issues/100>

⁵⁵⁸⁸ <https://github.com/STELLAR-GROUP/hpx/issues/101>

⁵⁵⁸⁹ <https://github.com/STELLAR-GROUP/hpx/issues/102>

⁵⁵⁹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/114>

⁵⁵⁹¹ <https://github.com/STELLAR-GROUP/hpx/issues/115>

- Issue #119⁵⁵⁹² - marduk linking strategy breaks other applications
- Issue #121⁵⁵⁹³ - pbsdsh problem
- Issue #123⁵⁵⁹⁴ - marduk, dataflow and adaptive1d fail to build
- Issue #124⁵⁵⁹⁵ - Lower default preprocessing arity
- Issue #125⁵⁵⁹⁶ - Move hpx::detail::diagnostic_information out of the detail namespace
- Issue #126⁵⁵⁹⁷ - Test definitions for AGAS reference counting
- Issue #128⁵⁵⁹⁸ - Add averaging performance counter
- Issue #129⁵⁵⁹⁹ - Error with endian.hpp while building adaptive1d
- Issue #130⁵⁶⁰⁰ - Bad initialization of performance counters
- Issue #131⁵⁶⁰¹ - Add global startup/shutdown functions to component modules
- Issue #132⁵⁶⁰² - Avoid using auto_ptr
- Issue #133⁵⁶⁰³ - On Windows hpx.dll doesn't get installed
- Issue #134⁵⁶⁰⁴ - HPX_LIBRARY does not reflect real library name (on Windows)
- Issue #135⁵⁶⁰⁵ - Add detection of unique_ptr to build system
- Issue #137⁵⁶⁰⁶ - Add command line option allowing to repeatedly evaluate performance counters
- Issue #139⁵⁶⁰⁷ - Logging is broken
- Issue #140⁵⁶⁰⁸ - CMake problem on windows
- Issue #141⁵⁶⁰⁹ - Move all non-component libraries into \$PREFIX/lib/hpx
- Issue #143⁵⁶¹⁰ - adaptive1d throws an exception with the default command line options
- Issue #146⁵⁶¹¹ - Early exception handling is broken
- Issue #147⁵⁶¹² - Sheneos doesn't link on Linux
- Issue #149⁵⁶¹³ - sheneos_test hangs
- Issue #154⁵⁶¹⁴ - Compilation fails for r5661

⁵⁵⁹² <https://github.com/STELLAR-GROUP/hpx/issues/119>

⁵⁵⁹³ <https://github.com/STELLAR-GROUP/hpx/issues/121>

⁵⁵⁹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/123>

⁵⁵⁹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/124>

⁵⁵⁹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/125>

⁵⁵⁹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/126>

⁵⁵⁹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/128>

⁵⁵⁹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/129>

⁵⁶⁰⁰ <https://github.com/STELLAR-GROUP/hpx/issues/130>

⁵⁶⁰¹ <https://github.com/STELLAR-GROUP/hpx/issues/131>

⁵⁶⁰² <https://github.com/STELLAR-GROUP/hpx/issues/132>

⁵⁶⁰³ <https://github.com/STELLAR-GROUP/hpx/issues/133>

⁵⁶⁰⁴ <https://github.com/STELLAR-GROUP/hpx/issues/134>

⁵⁶⁰⁵ <https://github.com/STELLAR-GROUP/hpx/issues/135>

⁵⁶⁰⁶ <https://github.com/STELLAR-GROUP/hpx/issues/137>

⁵⁶⁰⁷ <https://github.com/STELLAR-GROUP/hpx/issues/139>

⁵⁶⁰⁸ <https://github.com/STELLAR-GROUP/hpx/issues/140>

⁵⁶⁰⁹ <https://github.com/STELLAR-GROUP/hpx/issues/141>

⁵⁶¹⁰ <https://github.com/STELLAR-GROUP/hpx/issues/143>

⁵⁶¹¹ <https://github.com/STELLAR-GROUP/hpx/issues/146>

⁵⁶¹² <https://github.com/STELLAR-GROUP/hpx/issues/147>

⁵⁶¹³ <https://github.com/STELLAR-GROUP/hpx/issues/149>

⁵⁶¹⁴ <https://github.com/STELLAR-GROUP/hpx/issues/154>

- Issue #155⁵⁶¹⁵ - Sine performance counters example chokes on chrono headers
- Issue #156⁵⁶¹⁶ - Add build type to –version
- Issue #157⁵⁶¹⁷ - Extend AGAS caching to store gid ranges
- Issue #158⁵⁶¹⁸ - r5691 doesn't compile
- Issue #160⁵⁶¹⁹ - Re-add AGAS function for resolving a locality to its prefix
- Issue #168⁵⁶²⁰ - Managed components should be able to access their own GID
- Issue #169⁵⁶²¹ - Rewrite AGAS future pool
- Issue #179⁵⁶²² - Complete switch to request class for AGAS server interface
- Issue #182⁵⁶²³ - Sine performance counter is loaded by other examples
- Issue #185⁵⁶²⁴ - Write tests for symbol namespace reference counting
- Issue #191⁵⁶²⁵ - Assignment of read-only variable in point_geometry
- Issue #200⁵⁶²⁶ - Seg faults when querying performance counters
- Issue #204⁵⁶²⁷ - –ifnames and suffix stripping needs to be more generic
- Issue #205⁵⁶²⁸ - –list-* and –print-counter-* options do not work together and produce no warning
- Issue #207⁵⁶²⁹ - Implement decrement entry merging
- Issue #208⁵⁶³⁰ - Replace the spinlocks in AGAS with hpx::lcos::local_mutexes
- Issue #210⁵⁶³¹ - Add an –ifprefix option
- Issue #214⁵⁶³² - Performance test for PX-thread creation
- Issue #216⁵⁶³³ - VS2010 compilation
- Issue #222⁵⁶³⁴ - r6045 context_linux_x86.hpp
- Issue #223⁵⁶³⁵ - fibonacci hangs when changing the state of an active thread
- Issue #225⁵⁶³⁶ - Active threads end up in the FEB wait queue
- Issue #226⁵⁶³⁷ - VS Build Error for Accumulator Client

⁵⁶¹⁵ <https://github.com/STELLAR-GROUP/hpx/issues/155>

⁵⁶¹⁶ <https://github.com/STELLAR-GROUP/hpx/issues/156>

⁵⁶¹⁷ <https://github.com/STELLAR-GROUP/hpx/issues/157>

⁵⁶¹⁸ <https://github.com/STELLAR-GROUP/hpx/issues/158>

⁵⁶¹⁹ <https://github.com/STELLAR-GROUP/hpx/issues/160>

⁵⁶²⁰ <https://github.com/STELLAR-GROUP/hpx/issues/168>

⁵⁶²¹ <https://github.com/STELLAR-GROUP/hpx/issues/169>

⁵⁶²² <https://github.com/STELLAR-GROUP/hpx/issues/179>

⁵⁶²³ <https://github.com/STELLAR-GROUP/hpx/issues/182>

⁵⁶²⁴ <https://github.com/STELLAR-GROUP/hpx/issues/185>

⁵⁶²⁵ <https://github.com/STELLAR-GROUP/hpx/issues/191>

⁵⁶²⁶ <https://github.com/STELLAR-GROUP/hpx/issues/200>

⁵⁶²⁷ <https://github.com/STELLAR-GROUP/hpx/issues/204>

⁵⁶²⁸ <https://github.com/STELLAR-GROUP/hpx/issues/205>

⁵⁶²⁹ <https://github.com/STELLAR-GROUP/hpx/issues/207>

⁵⁶³⁰ <https://github.com/STELLAR-GROUP/hpx/issues/208>

⁵⁶³¹ <https://github.com/STELLAR-GROUP/hpx/issues/210>

⁵⁶³² <https://github.com/STELLAR-GROUP/hpx/issues/214>

⁵⁶³³ <https://github.com/STELLAR-GROUP/hpx/issues/216>

⁵⁶³⁴ <https://github.com/STELLAR-GROUP/hpx/issues/222>

⁵⁶³⁵ <https://github.com/STELLAR-GROUP/hpx/issues/223>

⁵⁶³⁶ <https://github.com/STELLAR-GROUP/hpx/issues/225>

⁵⁶³⁷ <https://github.com/STELLAR-GROUP/hpx/issues/226>

- Issue #228⁵⁶³⁸ - Move all traits into namespace hpx::traits
- Issue #229⁵⁶³⁹ - Invalid initialization of reference in thread_init_data
- Issue #235⁵⁶⁴⁰ - Invalid GID in iostreams
- Issue #238⁵⁶⁴¹ - Demangle type names for the default implementation of get_action_name
- Issue #241⁵⁶⁴² - C++11 support breaks GCC 4.5
- Issue #247⁵⁶⁴³ - Reference to temporary with GCC 4.4
- Issue #248⁵⁶⁴⁴ - Seg fault at shutdown with GCC 4.4
- Issue #253⁵⁶⁴⁵ - Default component action registration kills compiler
- Issue #272⁵⁶⁴⁶ - G++ unrecognized command line option
- Issue #273⁵⁶⁴⁷ - quicksort example doesn't compile
- Issue #277⁵⁶⁴⁸ - Invalid CMake logic for Windows

2.11 Citing HPX

Please cite *HPX* whenever you use it for publications. Use our paper in The Journal of Open Source Software as the main citation for *HPX*:⁵⁶⁴⁹. Use the Zenodo entry for referring to the latest version of *HPX*:⁵⁶⁵⁰. Entries for citing specific versions of *HPX* can also be found at⁵⁶⁵¹.

2.12 HPX users

A list of institutions and projects using *HPX* can be found on the *HPX* Users⁵⁶⁵² page.

2.13 About HPX

2.13.1 History

The development of High Performance ParalleX (*HPX*) began in 2007. At that time, Hartmut Kaiser became interested in the work done by the ParalleX group at the Center for Computation and Technology (CCT)⁵⁶⁵³, a multi-disciplinary research institute at Louisiana State University (LSU)⁵⁶⁵⁴. The ParalleX group was working to develop a new and

⁵⁶³⁸ <https://github.com/STELLAR-GROUP/hpx/issues/228>

⁵⁶³⁹ <https://github.com/STELLAR-GROUP/hpx/issues/229>

5640 <https://github.com/STELLAR-GROUP/hpx/issues/235>5641 <https://github.com/STELLAR-GROUP/hpx/issues/238>5642 <https://github.com/STELLAR-GROUP/hpx/issues/241>5643 <https://github.com/STELLAR-GROUP/hpx/issues/247>5644 <https://github.com/STELLAR-GROUP/hpx/issues/248>5645 <https://github.com/STELLAR-GROUP/hpx/issues/253>5646 <https://github.com/STELLAR-GROUP/hpx/issues/272>5647 <https://github.com/STELLAR-GROUP/hpx/issues/273>5648 <https://github.com/STELLAR-GROUP/hpx/issues/277>5649 <https://joss.theoj.org/papers/022e5917b95517dff20cd3742ab95eca>5650 <https://doi.org/10.5281/zenodo.598202>5651 <https://doi.org/10.5281/zenodo.598202>5652 <https://hpx.stellar-group.org/hpx-users/>5653 <https://www.cct.lsu.edu>5654 <https://www.lsu.edu>

experimental execution model for future high performance computing architectures. This model was christened ParalleX. The first implementations of ParalleX were crude, and many of those designs had to be discarded entirely. However, over time the team learned quite a bit about how to design a parallel, distributed runtime system which implements the concepts of ParalleX.

From the very beginning, this endeavour has been a group effort. In addition to a handful of interested researchers, there have always been graduate and undergraduate students participating in the discussions, design, and implementation of *HPX*. In 2011 we decided to formalize our collective research efforts by creating the STE||AR⁵⁶⁵⁵ group (Systems Technology, Emergent Parallelism, and Algorithm Research). Over time, the team grew to include researchers around the country and the world. In 2014, the STE||AR⁵⁶⁵⁶ Group was reorganized to become the international community it is today. This consortium of researchers aims to develop stable, sustainable, and scalable tools which will enable application developers to exploit the parallelism latent in the machines of today and tomorrow. Our goal of the *HPX* project is to create a high quality, freely available, open source implementation of ParalleX concepts for conventional and future systems by building a modular and standards conforming runtime system for SMP and distributed application environments. The API exposed by *HPX* is conformant to the interfaces defined by the C++ ISO Standard and adheres to the programming guidelines used by the Boost⁵⁶⁵⁷ collection of C++ libraries. We steer the development of *HPX* with real world applications and aim to provide a smooth migration path for domain scientists.

To learn more about STE||AR⁵⁶⁵⁸ and ParalleX, see *People* and *Why HPX?*.

2.13.2 People

The STE||AR⁵⁶⁵⁹ Group (pronounced as stellar) stands for “Systems Technology, Emergent Parallelism, and Algorithm Research”. We are an international group of faculty, researchers, and students working at various institutions around the world. The goal of the STE||AR⁵⁶⁶⁰ Group is to promote the development of scalable parallel applications by providing a community for ideas, a framework for collaboration, and a platform for communicating these concepts to the broader community.

Our work is focused on building technologies for scalable parallel applications. *HPX*, our general purpose C++ runtime system for parallel and distributed applications, is no exception. We use *HPX* for a broad range of scientific applications, helping scientists and developers to write code which scales better and shows better performance compared to more conventional programming models such as MPI.

HPX is based on *ParalleX* which is a new (and still experimental) parallel execution model aiming to overcome the limitations imposed by the current hardware and the techniques we use to write applications today. Our group focuses on two types of applications - those requiring excellent strong scaling, allowing for a dramatic reduction of execution time for fixed workloads and those needing highest level of sustained performance through massive parallelism. These applications are presently unable (through conventional practices) to effectively exploit a relatively small number of cores in a multi-core system. By extension, these application will not be able to exploit high-end exascale computing systems which are likely to employ hundreds of millions of such cores by the end of this decade.

Critical bottlenecks to the effective use of new generation high performance computing (HPC) systems include:

- *Starvation*: due to lack of usable application parallelism and means of managing it,
- *Overhead*: reduction to permit strong scalability, improve efficiency, and enable dynamic resource management,
- *Latency*: from remote access across system or to local memories,
- *Contention*: due to multicore chip I/O pins, memory banks, and system interconnects.

⁵⁶⁵⁵ <https://stellar-group.org>

⁵⁶⁵⁶ <https://stellar-group.org>

⁵⁶⁵⁷ <https://www.boost.org/>

⁵⁶⁵⁸ <https://stellar-group.org>

⁵⁶⁵⁹ <https://stellar-group.org>

⁵⁶⁶⁰ <https://stellar-group.org>

The ParalleX model has been devised to address these challenges by enabling a new computing dynamic through the application of message-driven computation in a global address space context with lightweight synchronization. The work on *HPX* is centered around implementing the concepts as defined by the ParalleX model. *HPX* is currently targeted at conventional machines, such as classical Linux based Beowulf clusters and SMP nodes.

We fully understand that the success of *HPX* (and ParalleX) is very much the result of the work of many people. To see a list of who is contributing see our tables below.

HPX contributors

Table 2.68: Contributors

Name	Institution	Email
Hartmut Kaiser	Center for Computation and Technology (CCT) ⁵⁶⁶¹ , Louisiana State University (LSU) ⁵⁶⁶²	hkaiser@cct.lsu.edu
Thomas Heller	Department of Computer Science 3 - Computer Architecture ⁵⁶⁶³ , Friedrich-Alexander University Erlangen-Nuremberg (FAU) ⁵⁶⁶⁴	thom.heller@gmail.com
Agustin Berge	Center for Computation and Technology (CCT) ⁵⁶⁶⁵ , Louisiana State University (LSU) ⁵⁶⁶⁶	kaballo86@hotmail.com
Mikael Simberg	Swiss National Supercomputing Centre ⁵⁶⁶⁷	simbergm@cscs.ch
John Biddiscombe	Swiss National Supercomputing Centre ⁵⁶⁶⁸	biddisco@cscs.ch
Anton Bikineev	Center for Computation and Technology (CCT) ⁵⁶⁶⁹ , Louisiana State University (LSU) ⁵⁶⁷⁰	ant.bikineev@gmail.com
Martin Stumpf	Department of Computer Science 3 - Computer Architecture ⁵⁶⁷¹ , Friedrich-Alexander University Erlangen-Nuremberg (FAU) ⁵⁶⁷²	martin.h.stumpf@gmail.com
Bryce Adelstein Lelbach	NVIDIA ⁵⁶⁷³	brycelelbach@gmail.com
Shuangyang Yang	Center for Computation and Technology (CCT) ⁵⁶⁷⁴ , Louisiana State University (LSU) ⁵⁶⁷⁵	syang16@cct.lsu.edu
Jeroen Habraken	Technische Universiteit Eindhoven ⁵⁶⁷⁶	jhabraken@cct.lsu.edu
Steven Brandt	Center for Computation and Technology (CCT) ⁵⁶⁷⁷ , Louisiana State University (LSU) ⁵⁶⁷⁸	sbrandt@cct.lsu.edu
Antoine Tran Tan	Center for Computation and Technology (CCT) ⁵⁶⁷⁹ , Louisiana State University (LSU) ⁵⁶⁸⁰	antoine.trantan@lri.fr
Adrian Serio	Center for Computation and Technology (CCT) ⁵⁶⁸¹ , Louisiana State University (LSU) ⁵⁶⁸²	aserio@cct.lsu.edu
Maciej Brodowicz	Center for Research in Extreme Scale Technologies (CREST) ⁵⁶⁸³ , Indiana University (IU) ⁵⁶⁸⁴	mbrodowi@indiana.edu
Giannis Gonidelis	Center for Computation and Technology (CCT) ⁵⁶⁸⁵ , Louisiana State University (LSU) ⁵⁶⁸⁶	gonidelis@hotmail.com

Contributors to this document

Table 2.69: Documentation authors

Name	Institution	Email
Hartmut Kaiser	Center for Computation and Technology (CCT) ⁵⁶⁸⁷ , Louisiana State University (LSU) ⁵⁶⁸⁸	hkaiser@cct.lsu.edu
Thomas Heller	Department of Computer Science 3 - Computer Architecture ⁵⁶⁸⁹ , Friedrich-Alexander University Erlangen-Nuremberg (FAU) ⁵⁶⁹⁰	thom.heller@gmail.com
Bryce Adelstein Lelbach	NVIDIA ⁵⁶⁹¹	brycelelbach@gmail.com
Vinay C Amatya	Center for Computation and Technology (CCT) ⁵⁶⁹² , Louisiana State University (LSU) ⁵⁶⁹³	vamatya@cct.lsu.edu
Steven Brandt	Center for Computation and Technology (CCT) ⁵⁶⁹⁴ , Louisiana State University (LSU) ⁵⁶⁹⁵	sbrandt@cct.lsu.edu
Maciej Brodowicz	Center for Research in Extreme Scale Technologies (CREST) ⁵⁶⁹⁶ , Indiana University (IU) ⁵⁶⁹⁷	mbrodowi@indiana.edu
Adrian Serio	Center for Computation and Technology (CCT) ⁵⁶⁹⁸ , Louisiana State University (LSU) ⁵⁶⁹⁹	aserio@cct.lsu.edu
Rebecca Stobaugh		rstobaugh1@gmail.com
Dimitra Karatzas	Faculty of Electrical Engineering, Mathematics & Computer Science ⁵⁷⁰⁰ , Delft University of Technology ⁵⁷⁰¹	dimitra.karatzas11@gmail.com

5661 <https://www.cct.lsu.edu>5662 <https://www.lsu.edu>5663 <https://www3.cs.fau.de>5664 <https://www.fau.de>5665 <https://www.cct.lsu.edu>5666 <https://www.lsu.edu>5667 <https://www.cs.ces.ch>5668 <https://www.cs.ces.ch>5669 <https://www.cct.lsu.edu>5670 <https://www.lsu.edu>5671 <https://www3.cs.fau.de>5672 <https://www.fau.de>5673 <https://nvidia.com/>5674 <https://www.cct.lsu.edu>5675 <https://www.lsu.edu>5676 <https://www.tui.nl>5677 <https://www.cct.lsu.edu>5678 <https://www.lsu.edu>5679 <https://www.cct.lsu.edu>5680 <https://www.lsu.edu>5681 <https://www.cct.lsu.edu>5682 <https://www.lsu.edu>5683 <https://pti.iu.edu>5684 <https://www.iu.edu>5685 <https://www.cct.lsu.edu>5686 <https://www.lsu.edu>

Acknowledgements

Thanks also to the following people who contributed directly or indirectly to the project through discussions, pull requests, documentation patches, etc.

- Dimitra Karatza, for her work on refactoring the documentation and providing a new user-friendly environment during and after Google Season of Docs 2021.
- Srinivas Yadav, for his work on SIMD support in algorithms before and during Google Summer of Code 2021.
- Akhil Nair, for his work on adapting algorithms to C++20 before and during Google Summer of Code 2021.
- Alexander Toktarev, for updating the parallel algorithm customization points to use `tag_fallback_invoke` for the default implementations.
- Brice Goglin, for reporting and helping fix issues related to the integration of `hwloc` in *HPX*.
- Giannis Gonidelis, for his work on the ranges adaptation during the Google Summer of Code 2020.
- Auriane Reverdell (Swiss National Supercomputing Centre⁵⁷⁰²), for her tireless work on refactoring our CMake setup and modularizing *HPX*.
- Christopher Hinz, for his work on refactoring our CMake setup.
- Weile Wei, for fixing *HPX* builds with CUDA on Summit.
- Severin Strobl, for fixing our CMake setup related to linking and adding new entry points to the *HPX* runtime.
- Rebecca Stobaugh, for her major documentation review and contributions during and after the 2019 Google Season of Documentation.
- Jan Melech, for adding automatic serialization of simple structs.
- Austin McCartney, for adding concept emulation of the Ranges TS bidirectional and random access iterator concepts.
- Marco Diers, reporting and fixing issues related PMIx.
- Maximilian Bremer, for reporting multiple issues and extending the component migration tests.
- Piotr Mikolajczyk, for his improvements and fixes to the set and count algorithms.
- Grant Rostig, for reporting several deficiencies on our web pages.
- Jakub Golinowski, for implementing an *HPX* backend for OpenCV and in the process improving documentation and reporting issues.
- Mikael Simberg (Swiss National Supercomputing Centre⁵⁷⁰³), for his tireless help cleaning up and maintaining *HPX*.

⁵⁶⁸⁷ <https://www.cct.lsu.edu>

⁵⁶⁸⁸ <https://www.lsu.edu>

⁵⁶⁸⁹ <https://www3.cs.fau.de>

⁵⁶⁹⁰ <https://www.fau.de>

⁵⁶⁹¹ <https://nvidia.com/>

⁵⁶⁹² <https://www.cct.lsu.edu>

⁵⁶⁹³ <https://www.lsu.edu>

⁵⁶⁹⁴ <https://www.cct.lsu.edu>

⁵⁶⁹⁵ <https://www.lsu.edu>

⁵⁶⁹⁶ <https://pti.iu.edu>

⁵⁶⁹⁷ <https://www.iu.edu>

⁵⁶⁹⁸ <https://www.cct.lsu.edu>

⁵⁶⁹⁹ <https://www.lsu.edu>

⁵⁷⁰⁰ <https://www.tudelft.nl/en/eemcs>

⁵⁷⁰¹ <https://www.tudelft.nl/en/>

⁵⁷⁰² <https://www.cscs.ch>

⁵⁷⁰³ <https://www.cscs.ch>

- Tianyi Zhang, for his work on HPXMP.
- Shahrzad Shirzad, for her contributions related to Phylanx.
- Christopher Ogle, for his contributions to the parallel algorithms.
- Surya Priy, for his work with statistic performance counters.
- Anushi Maheshwari, for her work on random number generation.
- Bruno Pitrus, for his work with parallel algorithms.
- Nikunj Gupta, for rewriting the implementation of `hpx_main.hpp` and for his fixes for tests.
- Christopher Taylor, for his interest in *HPX* and the fixes he provided.
- Shoshana Jakobovits, for her work on the resource partitioner.
- Denis Blank, who re-wrote our unwrapped function to accept plain values arbitrary containers, and properly deal with nested futures.
- Ajai V. George, who implemented several of the parallel algorithms.
- Taeguk Kwon, who worked on implementing parallel algorithms as well as adapting the parallel algorithms to the Ranges TS.
- Zach Byerly ([Louisiana State University \(LSU\)](#)⁵⁷⁰⁴), who in his work developing applications on top of *HPX* opened tickets and contributed to the *HPX* examples.
- Daniel Estermann, for his work porting *HPX* to the Raspberry Pi.
- Alireza Kheirkhahan ([Louisiana State University \(LSU\)](#)⁵⁷⁰⁵), who built and administered our local cluster as well as his work in distributed IO.
- Abhimanyu Rawat, who worked on stack overflow detection.
- David Pfander, who improved signal handling in *HPX*, provided his optimization expertise, and worked on incorporating the Vc vectorization into *HPX*.
- Denis Demidov, who contributed his insights with VexCL.
- Khalid Hasanov, who contributed changes which allowed to run *HPX* on 64Bit power-pc architectures.
- Zahra Khatami ([Louisiana State University \(LSU\)](#)⁵⁷⁰⁶), who contributed the prefetching iterators and the persistent auto chunking executor parameters implementation.
- Marcin Copik, who worked on implementing GPU support for C++AMP and HCC. He also worked on implementing a HCC backend for *HPX*.*Compute*.
- Minh-Khanh Do, who contributed the implementation of several segmented algorithms.
- Bibek Wagle ([Louisiana State University \(LSU\)](#)⁵⁷⁰⁷), who worked on fixing and analyzing the performance of the *parcel* coalescing plugin in *HPX*.
- Lukas Troska, who reported several problems and contributed various test cases allowing to reproduce the corresponding issues.
- Andreas Schaefer, who worked on integrating his library ([LibGeoDecomp](#)⁵⁷⁰⁸) with *HPX*. He reported various problems and submitted several patches to fix issues allowing for a better integration with [LibGeoDecomp](#)⁵⁷⁰⁹.
- Satyaki Upadhyay, who contributed several examples to *HPX*.

⁵⁷⁰⁴ <https://www.lsu.edu>⁵⁷⁰⁵ <https://www.lsu.edu>⁵⁷⁰⁶ <https://www.lsu.edu>⁵⁷⁰⁷ <https://www.lsu.edu>⁵⁷⁰⁸ <https://www.libgeodecomp.org/>⁵⁷⁰⁹ <https://www.libgeodecomp.org/>

- Brandon Cordes, who contributed several improvements to the inspect tool.
- Harris Brakmic, who contributed an extensive build system description for building *HPX* with Visual Studio.
- Parsa Amini ([Louisiana State University \(LSU\)](#)⁵⁷¹⁰), who refactored and simplified the implementation of *AGAS* in *HPX* and who works on its implementation and optimization.
- Luis Martinez de Bartolome who implemented a build system extension for *HPX* integrating it with the [Conan](#)⁵⁷¹¹ C/C++ package manager.
- Vinay C Amatya ([Louisiana State University \(LSU\)](#)⁵⁷¹²), who contributed to the documentation and provided some of the *HPX* examples.
- Kevin Huck and Nick Chaimov ([University of Oregon](#)⁵⁷¹³), who contributed the integration of APEX (Autonomic Performance Environment for eXascale) with *HPX*.
- Francisco Jose Tapia, who helped with implementing the parallel sort algorithm for *HPX*.
- Patrick Diehl, who worked on implementing CUDA support for our companion library targeting GPGPUs ([HPXCL](#)⁵⁷¹⁴).
- Eric Lemanissier contributed fixes to allow compilation using the MingW toolchain.
- Nidhi Makhijani who helped cleaning up some enum consistencies in *HPX* and contributed to the resource manager used in the thread scheduling subsystem. She also worked on *HPX* in the context of the Google Summer of Code 2015.
- Larry Xiao, Devang Bacharwar, Marcin Copik, and Konstantin Kronfeldner who worked on *HPX* in the context of the Google Summer of Code program 2015.
- Daniel Bourgeois ([Center for Computation and Technology \(CCT\)](#)⁵⁷¹⁵) who contributed to *HPX* the implementation of several parallel algorithms (as proposed by [N4313](#)⁵⁷¹⁶).
- Anuj Sharma and Christopher Bross ([Department of Computer Science 3 - Computer Architecture](#)⁵⁷¹⁷), who worked on *HPX* in the context of the [Google Summer of Code](#)⁵⁷¹⁸ program 2014.
- Martin Stumpf ([Department of Computer Science 3 - Computer Architecture](#)⁵⁷¹⁹), who rebuilt our contiguous testing infrastructure (see the [HPX Buildbot Website](#)⁵⁷²⁰). Martin is also working on [HPXCL](#)⁵⁷²¹ (mainly all work related to [OpenCL](#)⁵⁷²²) and implementing an *HPX* backend for [POCL](#)⁵⁷²³, a portable computing language solution based on [OpenCL](#)⁵⁷²⁴.
- Grant Mercer ([University of Nevada, Las Vegas](#)⁵⁷²⁵), who helped creating many of the parallel algorithms (as proposed by [N4313](#)⁵⁷²⁶).

⁵⁷¹⁰ <https://www.lsu.edu>

⁵⁷¹¹ <https://www.conan.io/>

⁵⁷¹² <https://www.lsu.edu>

⁵⁷¹³ <https://uoregon.edu/>

⁵⁷¹⁴ <https://github.com/STELLAR-GROUP/hpxcl/>

⁵⁷¹⁵ <https://www.cct.lsu.edu>

⁵⁷¹⁶ <http://wg21.link/n4313>

⁵⁷¹⁷ <https://www3.cs.fau.de>

⁵⁷¹⁸ <https://developers.google.com/open-source/soc/>

⁵⁷¹⁹ <https://www3.cs.fau.de>

⁵⁷²⁰ <http://rostam.cct.lsu.edu/>

⁵⁷²¹ <https://github.com/STELLAR-GROUP/hpxcl/>

⁵⁷²² <https://www.khronos.org/opencl/>

⁵⁷²³ <https://portablecl.org/>

⁵⁷²⁴ <https://www.khronos.org/opencl/>

⁵⁷²⁵ <https://www.unlv.edu>

⁵⁷²⁶ <http://wg21.link/n4313>

- Damond Howard (Louisiana State University (LSU)⁵⁷²⁷), who works on HPXCL⁵⁷²⁸ (mainly all work related to CUDA⁵⁷²⁹).
- Christoph Junghans (Los Alamos National Lab), who helped making our buildsystem more portable.
- Antoine Tran Tan (Laboratoire de Recherche en Informatique, Paris), who worked on integrating HPX as a backend for NT2⁵⁷³⁰. He also contributed an implementation of an API similar to Fortran co-arrays on top of HPX.
- John Biddiscombe (Swiss National Supercomputing Centre⁵⁷³¹), who helped with the BlueGene/Q port of HPX, implemented the parallel sort algorithm, and made several other contributions.
- Erik Schnetter (Perimeter Institute for Theoretical Physics), who greatly helped to make HPX more robust by submitting a large amount of problem reports, feature requests, and made several direct contributions.
- Mathias Gaunard (Metascale), who contributed several patches to reduce compile time warnings generated while compiling HPX.
- Andreas Buhr, who helped with improving our documentation, especially by suggesting some fixes for inconsistencies.
- Patricia Grubel (New Mexico State University⁵⁷³²), who contributed the description of the different HPX thread scheduler policies and is working on the performance analysis of our thread scheduling subsystem.
- Lars Viklund, whose wit, passion for testing, and love of odd architectures has been an amazing contribution to our team. He has also contributed platform specific patches for FreeBSD and MSVC12.
- Agustin Berge, who contributed patches fixing some very nasty hidden template meta-programming issues. He rewrote large parts of the API elements ensuring strict conformance with the C++ ISO Standard.
- Anton Bikineev for contributing changes to make using boost::lexical_cast safer, he also contributed a thread safety fix to the iostreams module. He also contributed a complete rewrite of the serialization infrastructure replacing Boost.Serialization inside HPX.
- Pyry Jakkola, who contributed the Mac OS build system and build documentation on how to build HPX using Clang and libc++.
- Mario Mulansky, who created an HPX backend for his Boost.Odeint library, and who submitted several test cases allowing us to reproduce and fix problems in HPX.
- Rekha Raj, who contributed changes to the description of the Windows build instructions.
- Jeremy Kemp how worked on an HPX OpenMP backend and added regression tests.
- Alex Nagelberg for his work on implementing a C wrapper API for HPX.
- Chen Guo, helvihartmann, Nicholas Pezolano, and John West who added and improved examples in HPX.
- Joseph Kleinhenz, Markus Elfring, Kirill Kropivnyansky, Alexander Neundorf, Bryant Lam, and Alex Hirsch who improved our CMake.
- Tapasweni Pathak, Praveen Velliengiri, Jean-Loup Tastet, Michael Levine, Aalekh Nigam, HadrienG2, Prayag Verma, Islada, Alex Myczko, and Avyav Kumar who improved the documentation.
- Jayesh Badwaik, J. F. Bastien, Christoph Garth, Christopher Hinz, Brandon Kohn, Mario Lang, Maikel Nadolski, pierrele, hendrx, Dekken, woodmeister123, xaguilar, Andrew Kemp, Dylan Stark, Matthew Anderson, Jeremy Wilke, Jiazheng Yuan, CyberDrudge, david8dixon, Maxwell Reeser, Raffaele Solca, Marco Ippolito,

⁵⁷²⁷ <https://www.lsu.edu>⁵⁷²⁸ <https://github.com/STELLAR-GROUP/hpxcl/>⁵⁷²⁹ https://www.nvidia.com/object/cuda_home_new.html⁵⁷³⁰ <https://www.numscale.com/nt2/>⁵⁷³¹ <https://www.cscs.ch>⁵⁷³² <https://www.nmsu.edu>

Jules Penuchot, Weile Wei, Severin Strobl, Kor de Jong, albestro, Jeff Trull, Yuri Victorovich, and Gregor Daiß who contributed to the general improvement of *HPX*.

*HPX Funding Acknowledgements*⁵⁷³³ lists current and past funding sources for *HPX*. Special thanks to [Google Summer of Code](#)⁵⁷³⁴ and [Google Season of Docs](#)⁵⁷³⁵ for the continuous support they provide which helps us enhance both our code and our documentation.

⁵⁷³³ <https://hpx.stellar-group.org/funding-acknowledgements/>

⁵⁷³⁴ <https://developers.google.com/open-source/soc/>

⁵⁷³⁵ <https://developers.google.com/season-of-docs>

**CHAPTER
THREE**

INDEX

- genindex

INDEX

Symbols

```
--hpx:affinity arg
    command line option, 95
--hpx:agas arg
    command line option, 94
--hpx:app-config arg
    command line option, 96
--hpx:attach-debugger arg
    command line option, 96
--hpx:bind arg
    command line option, 95
--hpx:config arg
    command line option, 96
--hpx:connect
    command line option, 94
--hpx:console
    command line option, 94
--hpx:cores arg
    command line option, 95
--hpx:debug-agas-log [arg]
    command line option, 96
--hpx:debug-app-log [arg]
    command line option, 96
--hpx:debug-clp
    command line option, 96
--hpx:debug-hpx-log [arg]
    command line option, 96
--hpx:debug-parcel-log [arg]
    command line option, 96
--hpx:debug-timing-log [arg]
    command line option, 96
--hpx:dump-config
    command line option, 96
--hpx:dump-config-initial
    command line option, 96
--hpx:endnodes
    command line option, 94
--hpx:exit
    command line option, 96
--hpx:expect-connecting-localities
    command line option, 95
--hpx:help
    command line option, 94
--hpx:high-priority-threads arg
    command line option, 95
--hpx:hpx arg
    command line option, 94
--hpx:ifprefix arg
    command line option, 94
--hpx:ifsuffix arg
    command line option, 94
--hpx:iftransform arg
    command line option, 95
--hpx:ignore-batch-env
    command line option, 95
--hpx:info
    command line option, 94
--hpx:ini arg
    command line option, 96
--hpx:list-component-types
    command line option, 96
--hpx:list-counter-infos
    command line option, 97
--hpx:list-counters
    command line option, 97
--hpx:list-symbolic-names
    command line option, 96
--hpx:localities arg
    command line option, 95
--hpx:no-csv-header
    command line option, 97
--hpx:node arg
    command line option, 95
--hpx:nodofile arg
    command line option, 94
--hpx:nodes arg
    command line option, 94
--hpx:numa-sensitive
    command line option, 95
--hpx:options-file arg
    command line option, 94
--hpx:print-bind
    command line option, 95
--hpx:print-counter
```

```
    command line option, 97
--hpx:print-counter-at arg
    command line option, 97
--hpx:print-counter-destination
    command line option, 97
--hpx:print-counter-format
    command line option, 97
--hpx:print-counter-interval
    command line option, 97
--hpx:print-counter-reset
    command line option, 97
--hpx:print-counters-locally
    command line option, 97
--hpx:pu-offset
    command line option, 95
--hpx:pu-step
    command line option, 95
--hpx:queuing arg
    command line option, 95
--hpx:reset-counters
    command line option, 97
--hpx:run-agas-server
    command line option, 94
--hpx:run-agas-server-only
    command line option, 94
--hpx:run-hpx-main
    command line option, 94
--hpx:threads arg
    command line option, 95
--hpx:use-process-mask
    command line option, 95
--hpx:version
    command line option, 94
--hpx:worker
    command line option, 94
```

A

Action, [214](#)
Active Global Address Space, [213](#)
AGAS, [213](#)
AMPLIFIER_ROOT:PATH
 command line option, 54
applier (*C++ type*), [912](#)

B

BOOST_ROOT:PATH
 command line option, 53
BREATHE_APIDOC_ROOT:PATH
 command line option, [939](#)

C

command line option
--hpx:affinity arg, [95](#)
--hpx:agas arg, [94](#)

```
--hpx:app-config arg, 96
--hpx:attach-debugger arg, 96
--hpx:bind arg, 95
--hpx:config arg, 96
--hpx:connect, 94
--hpx:console, 94
--hpx:cores arg, 95
--hpx:debug-agas-log [arg], 96
--hpx:debug-app-log [arg], 96
--hpx:debug-clp, 96
--hpx:debug-hpx-log [arg], 96
--hpx:debug-parcel-log [arg], 96
--hpx:debug-timing-log [arg], 96
--hpx:dump-config, 96
--hpx:dump-config-initial, 96
--hpx:endnodes, 94
--hpx:exit, 96
--hpx:expect-connecting-localities,
    95
--hpx:help, 94
--hpx:high-priority-threads arg, 95
--hpx:hpx arg, 94
--hpx:ifprefix arg, 94
--hpx:ifsuffix arg, 94
--hpx:iftransform arg, 95
--hpx:ignore-batch-env, 95
--hpx:info, 94
--hpx:ini arg, 96
--hpx:list-component-types, 96
--hpx:list-counter-infos, 97
--hpx:list-counters, 97
--hpx:list-symbolic-names, 96
--hpx:localities arg, 95
--hpx:no-csv-header, 97
--hpx:node arg, 95
--hpx:nodofile arg, 94
--hpx:nodes arg, 94
--hpx:numa-sensitive, 95
--hpx:options-file arg, 94
--hpx:print-bind, 95
--hpx:print-counter, 97
--hpx:print-counter-at arg, 97
--hpx:print-counter-destination, 97
--hpx:print-counter-format, 97
--hpx:print-counter-interval, 97
--hpx:print-counter-reset, 97
--hpx:print-counters-locally, 97
--hpx:pu-offset, 95
--hpx:pu-step, 95
--hpx:queuing arg, 95
--hpx:reset-counters, 97
--hpx:run-agas-server, 94
--hpx:run-agas-server-only, 94
--hpx:run-hpx-main, 94
```

```
--hpx:threads arg, 95
--hpx:use-process-mask, 95
--hpx:version, 94
--hpx:worker, 94
AMPLIFIER_ROOT:PATH, 54
BOOST_ROOT:PATH, 53
BREATHE_APIDOC_ROOT:PATH, 939
DOXYGEN_ROOT:PATH, 939
HDF5_ROOT:PATH, 54
HPX_COROUTINES_WITH_SWAP_CONTEXT_EMULATION:BOOL, 47
49
HPX_DATASTRUCTURES_WITH_ADAPT_STD_TUPLE:BOOL, 47
53
HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPARE_WITH_EXAMPLES, 37
53
HPX_ITERATOR_SUPPORT_WITH_BOOST_ITERATOR_HPX_AWARE_EXAMPLES_MAF5_BOOST, 48
53
HPX_SERIALIZATION_WITH_ALL_TYPES_ARE_BITWISE_EQUIVALENT:BOOL, 48
53
HPX_SERIALIZATION_WITH_ALLOW_CONST_TUPLE_MEMBERS:EXAMPLES_TBB:BOOL, 48
53
HPX_SERIALIZATION_WITH_ALLOW_RAW_POINTER_SERIALIZATION:BOOL,
53
HPX_SERIALIZATION_WITH_BOOST_TYPES:BOOL, HPX_WITH_FAULT_TOLERANCE:BOOL, 45
53
HPX_SERIALIZATION_WITH_SUPPORTS_ENDIANESCAPE_WITH_FETCH_LCI:BOOL, 48
53
HPX_TOPOLOGY_WITH_ADDITIONAL_HWLOC_TESTING:GCC_VERSION_CHECK:BOOL, 46
53
HPX_WITH_AGAS_DUMP_REFCNT_ENTRIES:BOOL, 37
50
HPX_WITH_APEX, 37
HPX_WITH_APEX:BOOL, 51
HPX_WITH_ASIO_TAG:STRING, 47
HPX_WITH_ATTACH_DEBUGGER_ON_TEST_FAILURE:WITH_IO_COUNTERS:BOOL, 48
52
HPX_WITH_AUTOMATIC_SERIALIZATION_REGISTRATION_NOTIFY:BOOL, 51
45
HPX_WITH_BENCHMARK_SCRIPTS_PATH:PATH, 45
HPX_WITH_BUILD_BINARY_PACKAGE:BOOL,
45
HPX_WITH_CHECK_MODULE_DEPENDENCIES:BOOL, HPX_WITH_MAX_CPU_COUNT:STRING, 49
45
HPX_WITH_COMPILE_ONLY_TESTS:BOOL, 47
HPX_WITH_COMPILER_WARNINGS:BOOL, 45
HPX_WITH_COMPILER_WARNINGS_AS_ERRORS:BOOL, 46
45
HPX_WITH_COMPRESSION_BZIP2:BOOL, 45
HPX_WITH_COMPRESSION_SNAPPY:BOOL, 45
HPX_WITH_COMPRESSION_ZLIB:BOOL, 45
HPX_WITH_COROUTINE_COUNTERS:BOOL, 49
HPX_WITH_CUDA, 37
HPX_WITH_CUDA:BOOL, 45
HPX_WITH_CXX_STANDARD, 37
HPX_WITH_CXX_STANDARD:STRING, 45
HPX_WITH_DATAPAR_VC:BOOL, 45
HPX_WITH_DEPRECATED_WARNINGS:BOOL,
45
HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL,
45
HPX_WITH_DISTRIBUTED_RUNTIME:BOOL,
47
HPX_WITH_DOCUMENTATION_OUTPUT_FORMATS:STRING,
47
HPX_WITH_DYNAMIC_HPX_MAIN:BOOL, 45
HPX_WITH_EXAMPLES:BOOL, 47
HPX_WITH_EXAMPLES_OPENMP:BOOL, 48
HPX_WITH_THREADS:BOOL, 48
HPX_WITH_EXECUTABLE_PREFIX:STRING,
48
HPX_WITH_FAIL_COMPILE_TESTS:BOOL, 48
HPX_WITH_FAULT_TOLERANCE:BOOL, 45
HPX_WITH_FETCH_ASIO:BOOL, 48
HPX_WITH_FULL_RPATH:BOOL, 46
HPX_WITH_GENERIC_CONTEXT_COROUTINES,
46
HPX_WITH_GENERIC_CONTEXT_COROUTINES:BOOL,
46
HPX_WITH_HIDDEN_VISIBILITY:BOOL, 46
HPX_WITH_HIP:BOOL, 46
HPX_WITH_IO_POOL:BOOL, 49
HPX_WITH_LCI_TAG:STRING, 48
HPX_WITH_LOGGING:BOOL, 46
HPX_WITH_MALLOC, 37
HPX_WITH_MALLOC:STRING, 46
HPX_WITH_MAX_CPU_COUNT, 37
HPX_WITH_MAX_NUMA_DOMAIN_COUNT:STRING,
49
HPX_WITH_MODULES_AS_STATIC_LIBRARIES:BOOL,
49
HPX_WITH_NETWORKING:BOOL, 51
HPX_WITH_NICE_THREADLEVEL:BOOL, 46
HPX_WITH_PAPI:BOOL, 51
HPX_WITH_PARALLEL_TESTS_BIND_NONE:BOOL,
52
HPX_WITH_PARCEL_COALESCING:BOOL, 46
HPX_WITH_PARCEL_PROFILING:BOOL, 51
```

HPX_WITH_PARCELPORT_ACTION_COUNTERS:BOOL
51

HPX_WITH_PARCELPORT_COUNTERS:BOOL,
51

HPX_WITH_PARCELPORT_LCI:BOOL, 51

HPX_WITH_PARCELPORT_LIBFABRIC:BOOL,
51

HPX_WITH_PARCELPORT_MPI, 37

HPX_WITH_PARCELPORT_MPI:BOOL, 51

HPX_WITH_PARCELPORT_TCP, 37

HPX_WITH_PARCELPORT_TCP:BOOL, 51

HPX_WITH_PKGCONFIG:BOOL, 46

HPX_WITH_PRECOMPILED_HEADERS:BOOL,
46

HPX_WITH_RUN_MAIN_EVERYWHERE:BOOL,
46

HPX_WITH_SANITIZEZERS:BOOL, 52

HPX_WITH_SCHEDULER_LOCAL_STORAGE:BOOL,
49

HPX_WITH_SPINLOCK_DEADLOCK_DETECTION:BOOL
49

HPX_WITH_SPINLOCK_POOL_NUM:STRING,
49

HPX_WITH_STACKOVERFLOW_DETECTION:BOOL,
46

HPX_WITH_STACKTRACES:BOOL, 50

HPX_WITH_STACKTRACES_DEMANGLE_SYMBOLS:BOOL
50

HPX_WITH_STACKTRACES_STATIC_SYMBOLS:BOOL
50

HPX_WITH_STATIC_LINKING:BOOL, 46

HPX_WITH_TESTS, 37

HPX_WITH_TESTS:BOOL, 48

HPX_WITH_TESTS_BENCHMARKS:BOOL, 48

HPX_WITH_TESTS_DEBUG_LOG:BOOL, 52

HPX_WITH_TESTS_DEBUG_LOG_DESTINATION:STRING
52

HPX_WITH_TESTS_EXAMPLES:BOOL, 48

HPX_WITH_TESTS_EXTERNAL_BUILD:BOOL,
48

HPX_WITH_TESTS_HEADERS:BOOL, 48

HPX_WITH_TESTS_MAX_THREADS_PER_LOCALITY:STRING
52

HPX_WITH_TESTS_REGRESSIONS:BOOL, 48

HPX_WITH_TESTS_UNIT:BOOL, 48

HPX_WITH_THREAD_BACKTRACE_DEPTH:STRING,
50

HPX_WITH_THREAD_BACKTRACE_ON_SUSPENSION:BOOL
50

HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES:BOOL
50

HPX_WITH_THREAD_CUMULATIVE_COUNTS:BOOL,
50

HPX_WITH_THREAD_DEBUG_INFO:BOOL, 52

HPX_WITH_THREAD_DESCRIPTION_FULL:BOOL,
52

HPX_WITH_THREAD_GUARD_PAGE:BOOL, 52

HPX_WITH_THREAD_IDLE_RATES:BOOL, 50

HPX_WITH_THREAD_LOCAL_STORAGE:BOOL,
50

HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF:BOOL,
50

HPX_WITH_THREAD_QUEUE_WAITTIME:BOOL,
50

HPX_WITH_THREAD_STACK_MMAP:BOOL, 50

HPX_WITH_THREAD_STEALING_COUNTS:BOOL,
50

HPX_WITH_THREAD_TARGET_ADDRESS:BOOL,
50

HPX_WITH_TIMER_POOL:BOOL, 50

HPX_WITH_TOOLS:BOOL, 48

HPX_WITH_UNITY_BUILD:BOOL, 46

HPX_WITH_VALGRIND:BOOL, 52

HPX_WITH_VERIFY_LOCKS:BOOL, 52

HPX_WITH_VERIFY_LOCKS_BACKTRACE:BOOL,
52

HPX_WITH_VIM_YCM:BOOL, 46

HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD:STRI
46

HWLOC_ROOT:PATH, 54

API_ROOT:PATH, 54

SPHINX_ROOT:PATH, 939

Component, 214

D

DOXYGEN_ROOT:PATH
command line option, 939

F

Format, value (*C++ function*), 604, 605

G

get_memory_page_size (*C++ function*), 751

H

HDF5_ROOT:PATH
command line option, 54

hpx (*C++ type*), 268, 273, 274, 278, 280, 282, 287, 289,
290, 294, 296, 303, 305, 316, 320, 329, 330,
332, 334, 337, 348, 351, 354, 361, 366, 368,
373, 384, 388, 389, 393, 396, 401, 412, 418,
420, 426, 441, 446, 452, 461, 464, 468, 472,
476, 479, 481, 484, 486, 489, 493, 496, 500,
506, 516, 520, 524, 528, 531, 536, 539, 542,
545–548, 551, 555–559, 561, 563, 565, 567,
569, 571, 572, 575–578, 584, 588, 590–592,
594–596, 598, 602, 606, 609, 613, 618–624,

627–633, 638, 640, 642, 644–646, 648, 650–
 656, 658, 661, 662, 664–666, 671, 673, 675–
 677, 679, 680, 685–688, 695, 697–700, 702,
 704, 705, 707–709, 712–714, 717, 719, 724,
 725, 732, 734, 736, 739, 743–745, 751, 752,
 755, 759, 773, 778, 779, 782, 785, 786, 789,
 791, 796, 804–808, 810, 812, 813, 815–817,
 819, 821–823, 825, 827–829, 833, 834, 836,
 839–841, 843, 844, 846, 848, 851, 853, 855,
 857, 858, 860, 861, 864, 866, 867, 870–873,
 875, 882, 888, 892, 895, 896, 898, 899, 901,
 905, 910, 912–917, 919, 922–929, 931, 932
 hpx::actions (*C++ type*), 755
 hpx::actions::basic_action (*C++ struct*), 755
 hpx::agas (*C++ type*), 759, 773, 836
 hpx::agas::addressing_service (*C++ struct*), 759
 hpx::agas::addressing_service::~addressing_service (*C++ function*), 760
 hpx::agas::addressing_service::action_prx (*C++ member*), 772
 hpx::agas::addressing_service::addressing_service::get_cache_get_entry (*C++ function*), 761
 hpx::agas::addressing_service::bind_asyn (*C++ function*), 760
 hpx::agas::addressing_service::adjust_loh (*C++ function*), 760
 hpx::agas::addressing_service::begin_migh (*C++ function*), 771
 hpx::agas::addressing_service::bind_asyn (*C++ function*), 764
 hpx::agas::addressing_service::bind_local (*C++ function*), 764
 hpx::agas::addressing_service::bind_postprx (*C++ function*), 772
 hpx::agas::addressing_service::bind_rangeprx (*C++ function*), 765
 hpx::agas::addressing_service::bind_rangeprx (*C++ function*), 764
 hpx::agas::addressing_service::bootstrapprx (*C++ function*), 760
 hpx::agas::addressing_service::caching_ (*C++ member*), 772
 hpx::agas::addressing_service::clear_cachprx (*C++ function*), 771
 hpx::agas::addressing_service::componentprx (*C++ type*), 760
 hpx::agas::addressing_service::componentprx (*C++ member*), 772
 hpx::agas::addressing_service::console_chprx (*C++ member*), 771
 hpx::agas::addressing_service::console_chprx (*C++ member*), 771
 hpx::agas::addressing_service::decref (*C++ function*), 769
 hpx::agas::addressing_service::enable_releasprx (*C++ function*), 771
 hpx::agas::addressing_service::garbage_collect (*C++ function*), 760
 hpx::agas::addressing_service::garbage_collect_non_ (*C++ function*), 760
 hpx::agas::addressing_service::get_cache_entries (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_entry (*C++ function*), 771
 hpx::agas::addressing_service::get_cache_erase_entry (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_erase_entry (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_evictions (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_get_entry (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_insertion (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_insertion (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_misses (*C++ function*), 761
 hpx::agas::addressing_service::get_cache_update_entry (*C++ function*), 761
 hpx::agas::addressing_service::get_colocation_idas (*C++ function*), 768
 hpx::agas::addressing_service::get_component_idas (*C++ function*), 763
 hpx::agas::addressing_service::get_component_typeas (*C++ function*), 763
 hpx::agas::addressing_service::get_console_localityas (*C++ function*), 762
 hpx::agas::addressing_service::get_id_rangeas (*C++ function*), 764
 hpx::agas::addressing_service::get_local_componentas (*C++ function*), 761
 hpx::agas::addressing_service::get_local_localityas (*C++ function*), 760
 hpx::agas::addressing_service::get_local_localityas (*C++ function*), 761
 hpx::agas::addressing_service::get_local_primary_na (*C++ function*), 761
 hpx::agas::addressing_service::get_local_symbol_na

(C++ function), 761
 (C++ type), 760
hpx::agas::addressing_service::get_local_addressing_service::iterate_types_function
 (C++ function), 762
hpx::agas::addressing_service::get_num_local_addresses::addressing_service::launch_bootstrap
 (C++ function), 763
hpx::agas::addressing_service::get_num_local_addresses::addressing_service::locality_member
 (C++ function), 762
hpx::agas::addressing_service::get_num_local_addresses::addressing_service::locality_ns_member
 (C++ function), 763
hpx::agas::addressing_service::get_num_local_addresses::addressing_service::mark_as_migrated
 (C++ function), 763
hpx::agas::addressing_service::get_num_local_addresses::addressing_service::max_refcnt_requests_member
 (C++ function), 763
hpx::agas::addressing_service::get_num_local_addresses::addressing_service::migrated_objects_mt_member
 (C++ function), 763
hpx::agas::addressing_service::get_primary_addressing_service::migrated_objects_taken_member
 (C++ function), 761
hpx::agas::addressing_service::get_runtime_addressing_service::migrated_objects_taken_type
 (C++ function), 761
hpx::agas::addressing_service::get_status::addressing_service::mutex_type
 (C++ function), 760
hpx::agas::addressing_service::get_symbol::addressing_service::on_symbol_namespace_function
 (C++ function), 761
hpx::agas::addressing_service::gva_cache::addressing_service::pre_cache_endpoints_member
 (C++ member), 771
hpx::agas::addressing_service::gva_cache::addressing_service::primary_ns_member
 (C++ member), 771
hpx::agas::addressing_service::gva_cache::addressing_service::range_caching_type
 (C++ type), 760
hpx::agas::addressing_service::has_resolve_addressing_service::refcnt_requests_member
 (C++ function), 761
hpx::agas::addressing_service::HPX_NON_CHEAPABLE::addressing_service::refcnt_requests_count_member
 (C++ function), 760
hpx::agas::addressing_service::inref::addressing_service::refcnt_requests_mt_member
 (C++ function), 768
hpx::agas::addressing_service::inref_as_type::addressing_service::refcnt_requests_type_member
 (C++ function), 768
hpx::agas::addressing_service::initialize::addressing_service::register_console_function
 (C++ function), 760
hpx::agas::addressing_service::is_bootstrapped::addressing_service::register_factory_function
 (C++ function), 760
hpx::agas::addressing_service::is_connected::addressing_service::register_locality_function
 (C++ function), 760
hpx::agas::addressing_service::is_console::addressing_service::register_name_function
 (C++ function), 760
hpx::agas::addressing_service::is_local_address::addressing_service::register_name_async_function
 (C++ function), 767
hpx::agas::addressing_service::is_local_address::addressing_service::register_server_instance_function
 (C++ function), 767
hpx::agas::addressing_service::iterate_id::addressing_service::remove_cache_entry_function
 (C++ function), 769
hpx::agas::addressing_service::iterate_name::addressing_service::remove_resolved_locality_function
 (C++ type), 760
hpx::agas::addressing_service::iterate_type::addressing_service::resolve_async

(C++ function), 768
`hpx::agas::addressing_service::resolve_chp`~~hdx~~`egas::addressing_service::unbind_range_local`
(C++ function), 768
`hpx::agas::addressing_service::resolve_fhp`~~x~~`:a`~~gas~~`::addressing_service::unmark_as_migrated`
(C++ function), 768
`hpx::agas::addressing_service::resolve_fhp`~~x~~`:l`~~ags~~`::addressing_service::unregister_locality`
(C++ function), 768
`hpx::agas::addressing_service::resolve_fhp`~~x~~`:p`~~ag~~`a`~~ppro~~`addressing_service::unregister_name`
(C++ function), 772
`hpx::agas::addressing_service::resolve_lhp`~~x~~`:l`~~ags~~`::addressing_service::unregister_name_asy`
(C++ function), 767, 768
`hpx::agas::addressing_service::resolve_lhp`~~x~~`:l`~~ags~~`::addressing_service::update_cache_entry`
(C++ function), 761
`hpx::agas::addressing_service::resolve_lhp`~~x~~`:l`~~ags~~`::was_object_migrated`
(C++ function), 760
`hpx::agas::addressing_service::resolve_nhp`~~x~~`:a`~~gas~~`::was_object_migrated`
(C++ function), 770
`hpx::agas::addressing_service::resolve_nhp`~~x~~`:a`~~gas~~`::begin_migration` (C++ function),
838
`hpx::agas::addressing_service::resolved_hpx`~~ai~~`g`~~as~~`::bind` (C++ function), 837
(C++ member), 772
`hpx::agas::bind_gid_local` (C++ function),
838
`hpx::agas::addressing_service::resolved_localities`~~hpx~~`:m`~~tx~~`_`
(C++ member), 772
`hpx::agas::bind_range_local` (C++ function),
838
`hpx::agas::addressing_service::resolved_localities`~~hpx~~`:t`~~yp~~`_type`
(C++ type), 760
`hpx::agas::bootstrap_primary_namespace_gid`
`hpx::agas::addressing_service::rts_lva_` (C++ function), 773
(C++ member), 772
`hpx::agas::bootstrap_primary_namespace_id`
`hpx::agas::addressing_service::runtime_type` (C++ function), 773
(C++ member), 772
`hpx::agas::decref` (C++ function), 838
`hpx::agas::addressing_service::send_refchp`~~x~~`r`~~e~~`q`~~u~~`est`~~s~~`destroy_component` (C++ func-
(C++ function), 772
`hpx::agas::send_refchp`~~x~~`r`~~e~~`q`~~u~~`est`~~s~~`eady`~~m~~`igration` (C++ function), 838
(C++ function), 772
`hpx::agas::find_symbols` (C++ function), 839
`hpx::agas::addressing_service::send_refchp`~~x~~`r`~~e~~`q`~~u~~`est`~~s~~`g`~~a~~~~b~~~~h~~~~o~~~~c~~~~l~~~~l~~`ect` (C++ function),
838
`hpx::agas::addressing_service::send_refchp`~~x~~`r`~~e~~`q`~~u~~`est`~~s~~`g`~~a~~~~b~~~~h~~`_c`~~o~~`l`~~l~~`ect`
(C++ function), 773
`hpx::agas::get_all_locality_ids` (C++ function), 838
(C++ member), 772
`hpx::agas::set_localhpx`~~c~~~~a~~~~g~~`:get_colocation_id` (C++ func-
(C++ function), 760
`hpx::agas::set_status`~~hpx~~`:a`~~gas~~`::get_component_id` (C++ function),
839
`hpx::agas::addressing_service::start_shu`~~nd~~~~wn~~`gas::get_component_type_name` (C++ func-
(C++ function), 771
`hpx::agas::addressing_service::state_` hpx::agas::get_console_locality (C++ func-
(C++ member), 772
`hpx::agas::get_locality_id` (C++ function),
838
`hpx::agas::symbol_ns`~~hpx~~`:a`~~gas~~`::get_locality` (C++ function), 837
(C++ member), 772
`hpx::agas::get_locality_id` (C++ function),
837
`hpx::agas::addressing_service::synchronize_with`~~hpx~~`:a`~~gas~~`::sync_incref`
(C++ function), 761
`hpx::agas::get_next_id` (C++ function), 838
`hpx::agas::addressing_service::unbind_lohpx`~~x~~`:a`~~gas~~`::get_num_localities` (C++ func-
(C++ function), 765
`hpx::agas::addressing_service::unbind_raghx`~~a~~`gas::get_num_overall_threads` (C++

`function), 837`
`hpx::agas::get_num_threads (C++ function), 837`
`hpx::agas::get_primary_ns_lva (C++ function), 839`
`hpx::agas::get_runtime_support_lva (C++ function), 839`
`hpx::agas::get_symbol_ns_lva (C++ function), 839`
`hpx::agas::inref (C++ function), 838`
`hpx::agas::is_console (C++ function), 836`
`hpx::agas::is_local_address_cached (C++ function), 837`
`hpx::agas::is_local_lva_encoded_address (C++ function), 837`
`hpx::agas::mark_as_migrated (C++ function), 838`
`hpx::agas::on_symbol_namespace_event (C++ function), 838`
`hpx::agas::register_factory (C++ function), 839`
`hpx::agas::register_name (C++ function), 836`
`hpx::agas::replenish_credits (C++ function), 838`
`hpx::agas::resolve (C++ function), 837`
`hpx::agas::resolve_cached (C++ function), 837`
`hpx::agas::resolve_local (C++ function), 837`
`hpx::agas::resolve_name (C++ function), 836`
`hpx::agas::server (C++ type), 773`
`hpx::agas::server::primary_namespace (C++ struct), 774`
`hpx::agas::server::primary_namespace::alhpx::gas::server::primary_namespace::counter_data (C++ function), 775`
`hpx::agas::server::primary_namespace::bahpx::gas::server::primary_namespace::counter_data (C++ type), 774`
`hpx::agas::server::primary_namespace::behpx::gas::server::primary_namespace::counter_data (C++ function), 775`
`hpx::agas::server::primary_namespace::bihpx::dxggidgas::server::primary_namespace::counter_data (C++ function), 775`
`hpx::agas::server::primary_namespace::cohpx::gas::server::primary_namespace::counter_data (C++ function), 775`
`hpx::agas::server::primary_namespace::cohpx::nneatgtypegasserver::primary_namespace::counter_data (C++ type), 774`
`hpx::agas::server::primary_namespace::cohhpx::ragdasasserver::primary_namespace::counter_data (C++ struct), 776`
`hpx::agas::server::primary_namespace::cohhpx::ragdasasserver::primary_namespace::counter_data (C++ member), 777`
`hpx::agas::server::primary_namespace::cohhpx::ragdasasserver::sepvecoupremadytnamespace::counter_data (C++ struct), 777`
`hpx::agas::server::primary_namespace::cohhpx::ragdasasserver::sepvecoupremadytnamespace::counter_data (C++ function), 778`
`hpx::agas::server::primary_namespace::cohhpx::ragdasasserver::sepvecoupremadytnamespace::counter_data (C++ member), 778`

```

hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::increment
    (C++ function), 777
                                         (C++ function), 776
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::countem_cre
    (C++ function), 777
                                         (C++ function), 775
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::instance_na
    (C++ function), 777
                                         (C++ member), 776
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspace::locality_
    (C++ member), 777
                                         (C++ member), 776
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::migrating_obi
    (C++ function), 777
                                         (C++ member), 776
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::comigration_tak
    (C++ function), 777
                                         (C++ type), 775
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::editmigrat_mat
    (C++ function), 777
                                         (C++ member), 776
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::timesyluemespaceunmutex_type
    (C++ function), 777
                                         (C++ type), 774
hpx::agas::server::primary_namespace::cohperdata::sérveemprimarycaspate::migratingnext_id
    (C++ function), 777
                                         (C++ member), 776
hpx::agas::server::primary_namespace::cohperdata::seeelveggidary_naspace::primary_na
    (C++ member), 777
                                         (C++ function), 775
hpx::agas::server::primary_namespace::cohperdata::senbndgidary_naspace::refcnt_table_
    (C++ member), 777
                                         (C++ type), 774
hpx::agas::server::primary_namespace::cohperdata::server::primary_naspace::refcnts_
    (C++ member), 775
                                         (C++ member), 776
hpx::agas::server::primary_namespace::dehpxeagasrederver::primary_naspace::register_ser
    (C++ function), 775
                                         (C++ function), 775
hpx::agas::server::primary_namespace::dehpxeagasweerver::primary_naspace::resolve_free_
    (C++ function), 776
                                         (C++ function), 776
hpx::agas::server::primary_namespace::endpxiggasonserver::primary_naspace::resolve_gid
    (C++ function), 775
                                         (C++ function), 775
hpx::agas::server::primary_namespace::fihpxzagas::server::primary_naspace::resolve_gid_
    (C++ function), 775
                                         (C++ function), 776
hpx::agas::server::primary_namespace::frhpxcongenerve_nanc::primary_naspace::resolved_type
    (C++ function), 776
                                         (C++ type), 774
hpx::agas::server::primary_namespace::frhpxenaggs::server::primary_naspace::set_local_lo
    (C++ struct), 778
                                         (C++ function), 775
hpx::agas::server::primary_namespace::frhpxenaggs::fseeventrprimary_naspace::unbind_gid
    (C++ function), 778
                                         (C++ function), 775
hpx::agas::server::primary_namespace::frhpxenaggs::gidver::primary_naspace::unregister_se
    (C++ member), 778
                                         (C++ function), 775
hpx::agas::server::primary_namespace::frhpxenaggs::gidver::primary_naspace::wait_for_mig
    (C++ member), 778
                                         (C++ function), 776
hpx::agas::server::primary_namespace::frhpxenaggs::isavlabity_naspace_service_name
    (C++ member), 778
                                         (C++ member), 774
hpx::agas::server::primary_namespace::frhpxenaggs::salaband(C++ function), 838
    (C++ type), 775
                                         hpx::agas:::unbind_gid_local (C++ function),
hpx::agas::server::primary_namespace::free_entry838list_type
    (C++ type), 775
                                         hpx::agas:::unbind_range_local (C++ func
hpx::agas::server::primary_namespace::gva_tableion838type
    (C++ type), 774
                                         hpx::agas:::unmark_as_migrated (C++ func
hpx::agas::server::primary_namespace::gva_tableion839
    (C++ type), 774
                                         hpx::agas:::unregister_name (C++ function),
hpx::agas::server::primary_namespace::gvas836
    (C++ member), 776
                                         hpx::agas:::update_cache_entry (C++ func

```

tion), 837
hpx::agas::was_object_migrated (*C++ function*), 838
hpx::annotated_function (*C++ function*), 717
hpx::applier (*C++ type*), 910, 912
hpx::applier::applier (*C++ class*), 910
hpx::applier::applier::~applier (*C++ function*), 910
hpx::applier::applier::applier (*C++ function*), 910
hpx::applier::applier::get_localities (*C++ function*), 911
hpx::applier::applier::get_locality_id (*C++ function*), 911
hpx::applier::applier::get_raw_localities (*C++ function*), 911
hpx::applier::applier::get_raw_locality (*C++ function*), 910
hpx::applier::applier::get_raw_remote_lobs (*C++ function*), 911
hpx::applier::applier::get_remote_localities (*C++ function*), 911
hpx::applier::applier::get_runtime_support_gid (*C++ type*), 911
hpx::applier::applier::get_runtime_support_raw (*C++ type*), 911
hpx::applier::applier::get_thread_manager (*C++ function*), 910
hpx::applier::applier::HPX_NON_COPYABLE (*C++ function*), 910
hpx::applier::applier::init (*C++ function*), 910
hpx::applier::applier::initialize (*C++ function*), 910
hpx::applier::applier::runtime_support_ib (*C++ member*), 912
hpx::applier::applier::thread_manager_ (*C++ member*), 912
hpx::applier::get_applier (*C++ function*), 912
hpx::applier::get_applier_ptr (*C++ function*), 912
hpx::assertion (*C++ type*), 556
hpx::assertion::assertion_handler (*C++ type*), 556
hpx::assertion::set_assertion_handler (*C++ function*), 557
hpx::assertion_failure (*C++ enumerator*), 608
hpx::async_execute (*C++ member*), 633
hpx::async_execute_t (*C++ struct*), 633
hpx::async_execute_t::tagFallbackInvoke (*C++ function*), 634
hpx::bad_action_code (*C++ enumerator*), 607
hpx::bad_component_type (*C++ enumerator*), 607
hpx::bad_function_call (*C++ enumerator*), 609
hpx::bad_parameter (*C++ enumerator*), 607
hpx::bad_plugin_type (*C++ enumerator*), 609
hpx::bad_request (*C++ enumerator*), 607
hpx::bad_response_type (*C++ enumerator*), 608
hpx::barrier (*C++ class*), 705
hpx::barrier::arrive (*C++ function*), 706
hpx::barrier::arrive_and_drop (*C++ function*), 706
hpx::barrier::arrive_and_wait (*C++ function*), 706
hpx::barrier::arrived_ (*C++ member*), 707
hpx::barrier::barrier (*C++ function*), 706
hpx::barrier::completion_ (*C++ member*), 707
hpx::barrier::cond_ (*C++ member*), 707
hpx::barrier::expected_ (*C++ member*), 707
hpx::barrier::mtx_ (*C++ member*), 707
hpx::barrier::phase_ (*C++ member*), 707
hpx::barrier::wait (*C++ function*), 706
hpx::barrier<OnCompletion>::arrival_token
hpx::barrier<OnCompletion>::mutex_type
hpx::barrier::binary_semaphore (*C++ type*), 707
hpx::broken.promise (*C++ enumerator*), 608
hpx::broken.task (*C++ enumerator*), 608
hpx::bulk_async_execute (*C++ member*), 633
hpx::bulk_async_execute_t (*C++ struct*), 634
hpx::bulk_async_execute_t::tagFallbackInvoke (*C++ function*), 635
hpx::bulk_sync_execute (*C++ member*), 633
hpx::bulk_sync_execute_t (*C++ struct*), 635
hpx::bulk_sync_execute_t::tagFallbackInvoke (*C++ function*), 635
hpx::bulk_then_execute (*C++ member*), 633
hpx::bulk_then_execute_t (*C++ struct*), 635
hpx::bulk_then_execute_t::tagFallbackInvoke (*C++ function*), 636
hpx::collectives (*C++ type*), 805–808, 812, 815, 817, 821, 822, 825, 828
hpx::collectives::all_gather (*C++ function*), 805, 806
hpx::collectives::all_reduce (*C++ function*), 806, 807
hpx::collectives::all_to_all (*C++ function*), 807, 808
hpx::collectives::broadcast_from (*C++ function*), 812, 813
hpx::collectives::broadcast_to (*C++ function*), 812
hpx::collectives::create_channel_communicator (*C++ function*), 815

hpx::components::colocating_distribution::policy::components::component_first_dynamic
 (C++ function), 856
hpx::components::colocating_distribution::policy::components::component_invalid(C++
 enumerator), 842
hpx::components::colocating_distribution::policy::components::component_last (C++
 enumerator), 842
hpx::components::colocating_distribution::policy::components::resumption_match::C++
 enumerator), 842
hpx::components::colocating_distribution::policy::components::resumption_match::C++
 enumerator), 842
hpx::components::colocating_distribution::policy::components::component_plain_function
 (C++ function), 855
hpx::components::colocating_distribution::policy::components::component_registering::C++
 enumerator), 842
hpx::components::colocating_distribution::policy::components::component_registry
 (C++ struct), 898
hpx::components::colocating_distribution::policy::components::component_registry::get_component_...
 (C++ function), 856
hpx::components::colocating_distribution::policy::components::component_registry::register_compo...
 (C++ function), 856
hpx::components::colocating_distribution::policy::components::component_registry_base
 (C++ struct), 676
hpx::components::commandline_options_provider::components::component_registry_base::~componen...
 (C++ type), 840
hpx::components::commandline_options_provider::components::component_registry_base::get_compon...
 (C++ function), 676
hpx::components::component_agas_components::component_registry_base::register_compon...
 (C++ enumerator), 842
hpx::components::component_agas_locality::components::component_runtime_support
 (C++ enumerator), 842
hpx::components::component_agas_primary::components::component_startup_shutdown::get_sh...
 (C++ enumerator), 842
hpx::components::component_agas_symbol_name::components::component_startup_shutdown::get_st...
 (C++ enumerator), 842
hpx::components::component_barrier(C++ enumerator), 842
hpx::components::component_base_lco(hpx::components::component_startup_shutdown_base::C++ struct), 679
hpx::components::component_base_lco_with::components::component_startup_shutdown_base::C++ function), 679
hpx::components::component_base_lco_with::components::component_startup_shutdown_base::C++ function), 679
hpx::components::component_commandline(hpx::components::component_upper_bound::C++ struct), 839
hpx::components::component_commandline::hpx::components::component_upper_bound
 (C++ function), 842
hpx::components::component_commandline::hpx::components::component_upper_bound::binpacking_counter_name
 (C++ function), 840
hpx::components::component_commandline_broker::components::default_distribution_policy
 (C++ struct), 675
hpx::components::component_commandline_broker::components::command_id_deserializer::distribution_policy::apply
 (C++ function), 675
hpx::components::component_commandline_broker::components::command_id_deserializer::distribution_policy::apply
 (C++ function), 675
hpx::components::component_deleter_type(hpx::components::default_distribution_policy::async...
 (C++ type), 842
hpx::components::component_enum_type(hpx::components::default_distribution_policy::async...
 (C++ enum), 842

hpx::components::default_distribution::factory_enabled (C++ enumerator), 842
hpx::components::default_distribution::factory_satisfactory (C++ type), 901
hpx::components::default_distribution::factory_satisfactory::enumerator (C++ enum), 842
hpx::components::default_distribution::factory_satisfactory::get_base_type (C++ function), 843
hpx::components::default_distribution::factory_satisfactory::get_component_base_name (C++ function), 843
hpx::components::default_distribution::factory_satisfactory::get_component_name (C++ function), 843
hpx::components::default_distribution::factory_satisfactory::get_component_type (C++ function), 843
hpx::components::default_distribution::factory_satisfactory::get_component_type_name (C++ function), 842
hpx::components::default_distribution::factory_satisfactory::get_derived_type (C++ function), 843
hpx::components::default_layout (C++ member), 899 hpx::components::instance_count (C++ function), 842
hpx::components::deleter (C++ function), 842 hpx::components::instead (C++ type), 843
hpx::components::derived_component_type hpx::components::intrusive_ptr_add_ref (C++ function), 846
hpx::components::detail_adl_barrier hpx::components::intrusive_ptr_release (C++ function), 846
hpx::components::detail_adl_barrier::deshpay::bemphentsaimsgmam(C++ function), 918 ntrols_lifeti (C++ struct), 845, 846 hpx::components::server (C++ type), 912
hpx::components::detail_adl_barrier::deshpay::bemphentsaiservmanagedbyobjmptnastlifetime_com (C++ struct), 845, 846 (C++ function), 912
hpx::components::detail_adl_barrier::inithpx::components::detail_adl_barrier::inithpx::components::detail_adl_barrier::inithpx::components::detail_adl_barrier::inithpx::components::detail_adl_barrier::manage_lif(, 920 (C++ function), 845-847 hpx::components::stubs::runtime_support::bulk_create
hpx::components::detail_adl_barrier::manage_lif(, 920 (C++ function), 845-847 hpx::components::stubs::runtime_support::bulk_create
hpx::components::detail_adl_barrier::manage_lif(, 919 managed_object_controls_lifet (C++ struct), 845, 847 hpx::components::stubs::runtime_support::call_start
hpx::components::detail_adl_barrier::manage_lif(, 921 managed_object_is_lifetime_com (C++ struct), 845, 847 hpx::components::stubs::runtime_support::call_start
hpx::components::enabled (C++ function), 842 (C++ function), 921
hpx::components::enumerate_instance_count (C++ function), 842 hpx::components::stubs::runtime_support::copy_create (C++ function), 921
hpx::components::factory_check (C++ enu- hpx::components::stubs::runtime_support::copy_create (C++ function), 920
merator), 842
hpx::components::factory_disabled (C++ enumerator), 842 hpx::components::stubs::runtime_support::create_com (C++ function), 919

```
hpx::components::stubs::runtime_support::create(C++ type)858 sync  
    (C++ function), 919 hpx::components::target_distribution_policy::async_  
hpx::components::stubs::runtime_support::create(C++ type)858 colocated  
    (C++ function), 920 hpx::components::target_distribution_policy::bulk_  
hpx::components::stubs::runtime_support::create(C++ function)857 located_async  
    (C++ function), 920 hpx::components::target_distribution_policy::create_  
hpx::components::stubs::runtime_support::create(C++ function)857 counter  
    (C++ function), 922 hpx::components::target_distribution_policy::get_ne  
hpx::components::stubs::runtime_support::create(C++ function)858 counter_async  
    (C++ function), 922 hpx::components::target_distribution_policy::get_nu  
hpx::components::stubs::runtime_support::garbag(C++ function), 858  
    (C++ function), 922 hpx::components::target_distribution_policy::operator  
hpx::components::stubs::runtime_support::garbag(C++ function)857 nc  
    (C++ function), 922 hpx::components::target_distribution_policy::target_  
hpx::components::stubs::runtime_support::garbag(C++ function)857 blocking  
    (C++ function), 922 hpx::components::types_are_compatible  
hpx::components::stubs::runtime_support::get_cd(C++ function), 843  
    (C++ function), 922 hpx::components::unwrapping_result_policy  
hpx::components::stubs::runtime_support::get_cd(C++ function)858  
    (C++ function), 922 hpx::components::unwrapping_result_policy::apply  
hpx::components::stubs::runtime_support::load_d(C++ function), 859  
    (C++ function), 921 hpx::components::unwrapping_result_policy::apply_ch  
hpx::components::stubs::runtime_support::load_d(C++ function)859 nc  
    (C++ function), 921 hpx::components::unwrapping_result_policy::async  
hpx::components::stubs::runtime_support::migrat(C++ function)859  
    (C++ function), 921 hpx::components::unwrapping_result_policy::async_c  
hpx::components::stubs::runtime_support::migrat(C++ function)859 sync  
    (C++ function), 921 hpx::components::unwrapping_result_policy::async_re  
hpx::components::stubs::runtime_support::remove(C++ function)859 section_cache_async  
    (C++ function), 922 hpx::components::unwrapping_result_policy::async_re  
hpx::components::stubs::runtime_support::shutdd(C++ type), 859  
    (C++ function), 921 hpx::components::unwrapping_result_policy::get_ne  
hpx::components::stubs::runtime_support::shutdd(C++ function), 859  
    (C++ function), 921 hpx::components::unwrapping_result_policy::unwrappi  
hpx::components::stubs::runtime_support::shutdd(C++ function), 859  
    (C++ function), 921 hpx::compute (C++ type), 848, 851  
hpx::components::stubs::runtime_support::hpx::permcompute::host (C++ type), 851  
    (C++ function), 921 hpx::compute::host::block_executor (C++  
hpx::components::stubs::runtime_support::terminat(C++ function)851  
    (C++ function), 922 hpx::compute::host::block_executor::async_execute  
hpx::components::stubs::runtime_support::terminat(C++ function)851  
    (C++ function), 921 hpx::compute::host::block_executor::block_executor  
hpx::components::target (C++ member), 857 (C++ function), 851  
hpx::components::target_distribution_pol    (C++ struct), 857 (C++ function), 852  
hpx::components::target_distribution_pol    (C++ function), 858 (C++ function), 852  
hpx::components::target_distribution_pol    (C++ function), 858 (C++ member), 852  
hpx::components::target_distribution_pol    (C++ function), 858 (C++ member), 852  
hpx::components::target_distribution_pol    (C++ function), 858 (C++ function), 852  
hpx::components::target_distribution_pol
```


(C++ function), 708
hpx::counting_semaphore_var::operator=
 (C++ function), 708
hpx::counting_semaphore_var::signal
 (C++ function), 708
hpx::counting_semaphore_var::signal_all
 (C++ function), 708
hpx::counting_semaphore_var::try_wait
 (C++ function), 708
hpx::counting_semaphore_var::wait (C++
 function), 708
hpx::counting_semaphore_var<Mutex,
 N>::mutex_type (C++ type), 708
hpx::cuda (C++ type), 575
hpx::cuda::experimental (C++ type), 575
hpx::cuda::experimental::cuda_executor
 (C++ struct), 575
hpx::cuda::experimental::cuda_executor::hpx::distributed::latch (C++ func-
 tion), 824
hpx::cuda::experimental::cuda_executor::hpx::distributed::latch::try_wait (C++
 function), 825
hpx::cuda::experimental::cuda_executor::hpx::distributed::latch::wait (C++
 function), 825
hpx::cuda::experimental::cuda_executor::hpx::distributed::promise (C++ class), 785
 (C++ function), 575
hpx::cuda::experimental::cuda_executor::cuda_exec (C++ function), 788, 790
hpx::cuda::experimental::cuda_executor::post (C++ function), 788, 790
hpx::cuda::experimental::cuda_executor_base
 (C++ struct), 576
hpx::cuda::experimental::cuda_executor_base::cu (C++ function), 789, 790
 (C++ function), 576
hpx::cuda::experimental::cuda_executor_base::definition (C++ member), 576
hpx::cuda::experimental::cuda_executor_base::event_model (C++ member), 576
hpx::cuda::experimental::cuda_executor_base::get (C++ function), 789, 790
hpx::cuda::experimental::cuda_executor_base::swap (C++ function), 789
hpx::cuda::experimental::cuda_executor_bhp::duplicate_component_address (C++
 enumerator), 607
hpx::cuda::experimental::cuda_executor_bhp::duplicate_component_id (C++ enumera-
 tor), 608
hpx::custom_exception_info_handler_type
 (C++ type), 613
hpx::deadlock (C++ enumerator), 608
hpx::default_ (C++ enumerator), 678
hpx::diagnostic_information (C++ function),
 680
hpx::disconnect (C++ function), 862, 863
hpx::distributed (C++ type), 785, 789, 810, 823
hpx::distributed::barrier (C++ class), 810
 (C++ function), 811
hpx::distributed::barrier::synchronize
 (C++ function), 811
hpx::distributed::barrier::wait (C++
 function), 811
hpx::distributed::latch (C++ class), 823
hpx::distributed::latch::arrive_and_wait
 (C++ function), 824
hpx::distributed::latch::base_type (C++
 type), 825
hpx::distributed::latch::count_down
 (C++ function), 824
hpx::distributed::latch::count_down_and_wait
 (C++ function), 824
hpx::distributed::latch::is_ready (C++
 function), 824
hpx::distributed::latch::latch (C++ func-
 tion), 824
hpx::distributed::latch::try_wait (C++
 function), 825
hpx::distributed::promise::~promise
hpx::distributed::promise::operator=
 (C++ function), 788, 790
hpx::distributed::promise::promise (C++
 function), 788, 790
hpx::distributed::promise::set_value
hpx::distributed::promise::swap (C++
 function), 789, 790
hpx::distributed::promise<void,
 C++ type>::base_type
 (C++ class), 788, 789
hpx::util::unused_type>
hpx::util::unused_type>::base_type
hpx::util::unused_type>::get (C++ type), 789, 790
hpx::util::unused_type>::swap (C++ function), 789

610–612
`hpx::error_code::exception_ (C++ member), 612`
`hpx::error_code::get_message (C++ function), 612`
`hpx::error_code::operator= (C++ function), 612`
`hpx::exception (C++ class), 615`
`hpx::exception::~exception (C++ function), 616`
`hpx::exception::exception (C++ function), 616`
`hpx::exception::get_error (C++ function), 616`
`hpx::exception::get_error_code (C++ function), 616`
`hpx::exception_list (C++ class), 619`
`hpx::exception_list::begin (C++ function), 619`
`hpx::exception_list::end (C++ function), 619`
`hpx::exception_list::iterator (C++ type), 619`
`hpx::exception_list::size (C++ function), 619`
`hpx::execution (C++ type), 273, 620, 621, 627, 628, 631, 638, 642, 644–646, 651, 652`
`hpx::execution::auto_chunk_size (C++ struct), 620`
`hpx::execution::auto_chunk_size::auto_chhpx::execution::experimental::tag_fallback_invoke (C++ function), 621`
`hpx::execution::dynamic_chunk_size (C++ struct), 621`
`hpx::execution::dynamic_chunk_size::dynahpx::execution::experimental::task_group (C++ class), 621`
`hpx::execution::experimental (C++ type), 273, 638, 642, 644, 645, 651, 652`
`hpx::execution::experimental::annotating (C++ struct), 642`
`hpx::execution::experimental::annotating (C++ function), 642`
`hpx::execution::experimental::dynamic (C++ enumerator), 645`
`hpx::execution::experimental::fork_join (C++ class), 645`
`hpx::execution::experimental::fork_join (C++ function), 645`
`hpx::execution::experimental::is_nothrow (C++ member), 639`
`hpx::execution::experimental::is_receive (C++ struct), 639`
`hpx::execution::experimental::is_receive (C++ struct), 640`
`hpx::execution::experimental::is_receive (C++ member), 639`
`hpx::execution::experimental::is_receiver_v (C++ member), 639`
`hpx::execution::experimental::loop_schedule (C++ enum), 645`
`hpx::execution::experimental::operator<< (C++ function), 645`
`hpx::execution::experimental::scheduler_executor (C++ function), 651`
`hpx::execution::experimental::scheduler_executor (C++ struct), 651`
`hpx::execution::experimental::scheduler_executor:: (C++ function), 651`
`hpx::execution::experimental::scheduler_executor:: (C++ function), 651`
`hpx::execution::experimental::set_error (C++ function), 639`
`hpx::execution::experimental::set_error (C++ member), 639`
`hpx::execution::experimental::set_stopped (C++ function), 638`
`hpx::execution::experimental::set_stopped (C++ member), 639`
`hpx::execution::experimental::set_value (C++ function), 638`
`hpx::execution::experimental::set_value (C++ member), 639`
`hpx::execution::experimental::static_ (C++ enumerator), 645`
~~hpx::execution::experimental::tag_invoke (C++ function), 642~~
~~hpx::execution::experimental::tag_invoke (C++ function), 644~~
~~hpx::execution::experimental::task_group (C++ class), 273~~
~~hpx::execution::experimental::task_group::~task_group (C++ function), 273~~
~~hpx::execution::experimental::task_group::add_exception (C++ function), 273~~
~~hpx::execution::experimental::task_group::on_error (C++ member), 274~~
~~hpx::execution::experimental::task_group::has_arrived (C++ member), 274~~
~~hpx::execution::experimental::task_group::latch_ (C++ member), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ struct), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ function), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ member), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ member), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ function), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ function), 274~~
~~hpx::execution::experimental::task_group::on_exit (C++ function), 274~~

hpx::execution::experimental::task_group::filesystem (C++ type), 653
(C++ function), 273
hpx::execution::experimental::task_group::basename (C++ function),
(C++ function), 273
hpx::execution::experimental::task_group::serial::size (C++ member), 653
(C++ function), 273
hpx::execution::experimental::task_group::shared::state_type (C++ type), 273
hpx::execution::experimental::task_group::initial_path (C++ function),
(C++ member), 274
hpx::execution::experimental::task_group::state (C++ function), 653
hpx::execution::experimental::task_group::shared::state_type (C++ type), 273
hpx::execution::experimental::task_group::find_all_from_basename (C++ function),
hpx::execution::experimental::task_group::wait (C++ function), 830
(C++ function), 273
hpx::execution::experimental::task_group::find_all_localities (C++ function), 913,
hpx::execution::experimental::thread_pool_scheduler (C++ struct), 652
hpx::execution::experimental::thread_pool_scheduler::find_from_basename (C++ function), 830,
hpx::execution::experimental::thread_pool_scheduler (C++ function), 652
hpx::execution::experimental::thread_pool_scheduler::find_here (C++ function), 914
hpx::execution::guided_chunk_size (C++ struct), 627
hpx::execution::guided_chunk_size::guided_chunk (C++ function), 913, 915
hpx::execution::guided_chunk_size (C++ function), 627
hpx::execution::num_cores (C++ struct), 627
hpx::execution::num_cores::num_cores (C++ function), 628
hpx::execution::parallel_execution_tag (C++ struct), 638
hpx::execution::parallel_executor (C++ type), 646
hpx::execution::parallel_policy_executor (C++ struct), 646
hpx::execution::parallel_policy_executor::future_can_not_be_cancelled (C++ enumerator), 608
(C++ function), 646, 647
hpx::execution::parallel_policy_executor::future_does_not_support_cancellation (C++ enumerator), 608
hpx::execution::parallel_policy_executor<Policy> (C++ type), 646
hpx::execution::parallel_policy_executor<Policy>::executor_parameters_type (C++ type), 646
hpx::execution::persistent_auto_chunk_size (C++ struct), 628
hpx::execution::persistent_auto_chunk_size::get_error (C++ function), 681
(C++ function), 628
hpx::execution::sequenced_execution_tag (C++ struct), 638
hpx::execution::sequenced_executor (C++ struct), 652
hpx::execution::static_chunk_size (C++ struct), 631
hpx::execution::static_chunk_size::static_chunk_size (C++ function), 680
(C++ function), 631
hpx::execution::unsequenced_execution_tag (C++ struct), 638
hpx::experimental (C++ type), 329, 330
hpx::experimental::induction (C++ function), 329
hpx::experimental::reduction (C++ function), 330
hpx::filesystem::basename (C++ function), 653
hpx::filesystem::canonical (C++ function), 653
hpx::filesystem::initial_path (C++ function), 653
hpx::filesystem::error (C++ enumerator), 609
hpx::filesystem::shared::size (C++ function), 861, 862
hpx::find_all_from_basename (C++ function), 830
hpx::find_all_localities (C++ function), 913
hpx::find_from_basename (C++ function), 830,
hpx::find_locality (C++ function), 916
hpx::find_remote_localities (C++ function), 915
hpx::functional (C++ type), 668
hpx::functional::unwrap (C++ struct), 668
hpx::functional::unwrap_all (C++ struct), 668
hpx::functional::unwrap_n (C++ struct), 668
hpx::future_already_retrieved (C++ enumerator), 608
hpx::future_can_not_be_cancelled (C++ enumerator), 608
hpx::future_does_not_support_cancellation (C++ enumerator), 608
hpx::get_colocation_id (C++ function), 778,
hpx::get_error (C++ function), 613, 614
hpx::get_error_backtrace (C++ function), 682
hpx::get_error_config (C++ function), 684
hpx::get_error_file_name (C++ function), 614
hpx::get_error_function_name (C++ function), 614
hpx::get_error_host_name (C++ function), 681
hpx::get_error_line_number (C++ function), 615
hpx::get_error_locality_id (C++ function), 680
hpx::get_error_os_thread (C++ function), 682
hpx::get_error_process_id (C++ function), 681
hpx::get_error_state (C++ function), 684
hpx::get_error_thread_description (C++ function), 683
hpx::get_error_thread_id (C++ function), 683
hpx::get_error_what (C++ function), 613

hpx::get_hpx_category (*C++ function*), 610
 hpx::get_hpx_rethrow_category (*C++ function*), 610
 hpx::get_initial_num_localities (*C++ function*), 686
 hpx::get_local_worker_thread_num (*C++ function*), 732, 733
 hpx::get_locality_id (*C++ function*), 685
 hpx::get_locality_name (*C++ function*), 685, 916
 hpx::get_lva (*C++ struct*), 844
 hpx::get_lva::call (*C++ function*), 779, 780, 844
 hpx::get_lva<lcos::base_lco const> (*C++ struct*), 779, 780
 hpx::get_lva<lcos::base_lco> (*C++ struct*), 779
 hpx::get_num_localities (*C++ function*), 686, 917
 hpx::get_num_worker_threads (*C++ function*), 696
 hpx::get_os_thread_count (*C++ function*), 687
 hpx::get_os_thread_data (*C++ function*), 695
 hpx::get_ptr (*C++ function*), 834, 835
 hpx::get_runtime_instance_number (*C++ function*), 696
 hpx::get_runtime_mode_from_name (*C++ function*), 678
 hpx::get_runtime_mode_name (*C++ function*), 678
 hpx::get_runtime_support_ptr (*C++ function*), 899
 hpx::get_system_uptime (*C++ function*), 696
 hpx::get_thread_name (*C++ function*), 687
 hpx::get_thread_on_error_func (*C++ function*), 700
 hpx::get_thread_on_start_func (*C++ function*), 700
 hpx::get_thread_on_stop_func (*C++ function*), 700
 hpx::get_thread_pool_num (*C++ function*), 733
 hpx::get_worker_thread_num (*C++ function*), 732
 hpx::init (*C++ function*), 864–866
 hpx::init_params (*C++ struct*), 866
 hpx::init_params::cfg (*C++ member*), 867
 hpx::init_params::desc_cmdline (*C++ member*), 867
 hpx::init_params::mode (*C++ member*), 867
 hpx::init_params::rp_callback (*C++ member*), 867
 hpx::init_params::rp_mode (*C++ member*), 867
 hpx::init_params::shutdown (*C++ member*), 867
 hpx::init_params::startup (*C++ member*), 867
 hpx::internal_server_error (*C++ enumerator*), 607
 hpx::invalid (*C++ enumerator*), 678
 hpx::invalid_data (*C++ enumerator*), 608
 hpx::invalid_status (*C++ enumerator*), 607
 hpx::is_async_execution_policy (*C++ struct*), 632
 hpx::is_async_execution_policy_v (*C++ member*), 632
 hpx::is_execution_policy (*C++ struct*), 632
 hpx::is_execution_policy_v (*C++ member*), 632
 hpx::is_parallel_execution_policy (*C++ struct*), 632
 hpx::is_parallel_execution_policy_v (*C++ member*), 632
 hpx::is_running (*C++ function*), 696
 hpx::is_sequenced_execution_policy (*C++ struct*), 633
 hpx::is_sequenced_execution_policy_v (*C++ member*), 632
 hpx::is_starting (*C++ function*), 696
 hpx::is_stopped (*C++ function*), 696
 hpx::is_stopped_or_shutting_down (*C++ function*), 696
 hpx::kernel_error (*C++ enumerator*), 608
 hpx::last (*C++ enumerator*), 678
 hpx::latch (*C++ class*), 709
 hpx::latch::~latch (*C++ function*), 710
 hpx::latch::arrive_and_wait (*C++ function*), 710
 hpx::latch::cond_ (*C++ member*), 711
 hpx::latch::count_down (*C++ function*), 710
 hpx::latch::counter_ (*C++ member*), 711
 hpx::latch::HPX_NON_COPYABLE (*C++ function*), 710
 hpx::latch::latch (*C++ function*), 710
 hpx::latch::mtx_ (*C++ member*), 711
 hpx::latch::mutex_type (*C++ type*), 711
 hpx::latch::notified_ (*C++ member*), 711
 hpx::latch::try_wait (*C++ function*), 710
 hpx::latch::wait (*C++ function*), 710
 hpx::launch (*C++ struct*), 557
 hpx::launch::apply (*C++ member*), 558
 hpx::launch::async (*C++ member*), 558
 hpx::launch::deferred (*C++ member*), 558
 hpx::launch::fork (*C++ member*), 558
 hpx::launch::launch (*C++ function*), 557
 hpx::launch::select (*C++ member*), 558
 hpx::launch::sync (*C++ member*), 558

hpx::lcos (C++ type), 655, 658, 661, 662, 708, 711, (C++ function), 784
780, 782, 785, 786, 813, 816, 819, 827 hpx::lcos::base_lco_with_value::set_value_nonvirt (C++ function), 783

hpx::lcos::base_lco (C++ class), 780 hpx::lcos::base_lco::~base_lco (C++ function), 780

hpx::lcos::base_lco::base_type_holder (C++ type), 780 hpx::lcos::base_lco::connect (C++ function), 780

hpx::lcos::base_lco::connect_nonvirt (C++ function), 781 hpx::lcos::base_lco::disconnect (C++ function), 780

hpx::lcos::base_lco::disconnect_nonvirt (C++ function), 781 hpx::lcos::base_lco::disconnect_nonvirt (C++ member), 781

hpx::lcos::base_lco::finalize (C++ function), 780 hpx::lcos::base_lco::get_component_type (C++ function), 782

hpx::lcos::base_lco::HPX_DEFINE_COMPONENT_DIRECTIVE (C++ function), 781 hpx::lcos::base_lco::set_component_type (C++ function), 782

hpx::lcos::base_lco::set_event (C++ function), 780 hpx::lcos::base_lco::set_event_nonvirt (C++ function), 780

hpx::lcos::base_lco::set_exception (C++ function), 780 hpx::lcos::base_lco::set_exception_nonvirt (C++ function), 780

hpx::lcos::base_lco::wrapping_type (C++ type), 780 hpx::lcos::base_lco_with_value (C++ class), 782

hpx::lcos::base_lco_with_value::~base_lco_with_value (C++ function), 784, 785 hpx::lcos::base_lco_with_value::~base_lco_with_value (C++ function), 784

hpx::lcos::base_lco_with_value::get_component_type (C++ function), 784 hpx::lcos::base_lco_with_value::get_value (C++ function), 784, 785

hpx::lcos::base_lco_with_value::get_value_nonvirt (C++ function), 783 hpx::lcos::base_lco_with_value::get_value_nonvirt (C++ function), 783

hpx::lcos::base_lco_with_value::HPX_DEFINE_COMPONENT_DIRECTIVE (C++ function), 783 hpx::lcos::base_lco_with_value::set_component_type (C++ function), 784

hpx::lcos::base_lco_with_value::set_event (C++ function), 784 hpx::lcos::base_lco_with_value::set_event (C++ function), 784

hpx::lcos::base_lco_with_value::set_event_nonvirt (C++ function), 784 hpx::lcos::base_lco_with_value::set_event_nonvirt (C++ function), 784

hpx::lcos::base_lco_with_value::set_value (C++ member), 660 hpx::lcos::base_lco_with_value::set_value (C++ member), 660

hpx::lcos::base_lco_with_value::set_value_nonvirt (C++ function), 784 hpx::lcos::base_lco::base_and_gate (C++ struct), 658

hpx::lcos::base_lco::base_and_gate (C++ function), 658 hpx::lcos::local (C++ type), 658, 661, 662, 708, 711

hpx::lcos::base_lco::base_and_gate::and_gate (C++ function), 658 hpx::lcos::local::and_gate (C++ type), 658

hpx::lcos::base_lco::base_and_gate::base_type (C++ type), 659 hpx::lcos::local::and_gate::base_and_gate (C++ type), 658

hpx::lcos::base_lco::base_and_gate::get_future (C++ function), 658 hpx::lcos::local::and_gate::get_future (C++ function), 658

hpx::lcos::base_lco::base_and_gate::get_shared_future (C++ function), 658 hpx::lcos::local::and_gate::get_shared_future (C++ function), 658

hpx::lcos::base_lco::base_and_gate::operator= (C++ function), 658 hpx::lcos::local::and_gate::operator= (C++ function), 658

hpx::lcos::base_lco::base_and_gate::set (C++ function), 658 hpx::lcos::local::and_gate::set (C++ function), 658

hpx::lcos::base_lco::base_and_gate::syncronize (C++ function), 658 hpx::lcos::local::base_and_gate (C++ type), 658

hpx::lcos::base_lco::base_and_gate::base_and_gate (C++ type), 659 hpx::lcos::local::base_and_gate::base_and_gate (C++ type), 659

hpx::lcos::base_lco::base_and_gate::condition_list_type (C++ type), 660 hpx::lcos::local::base_and_gate::conditions_ (C++ type), 660

member), 709

hpx::lcos::local::event::mtx_ (C++ member), 709

hpx::lcos::local::event::mutex_type (C++ type), 709

hpx::lcos::local::event::occurred (C++ function), 709

hpx::lcos::local::event::reset (C++ function), 709

hpx::lcos::local::event::set (C++ function), 709

hpx::lcos::local::event::set_locked (C++ function), 709

hpx::lcos::local::event::wait (C++ function), 709

hpx::lcos::local::event::wait_locked (C++ function), 709

hpx::lcos::local::instead (C++ type), 708

hpx::lcos::local::latch (C++ class), 711

hpx::lcos::local::latch::~latch (C++ function), 711

hpx::lcos::local::latch::abort_all (C++ function), 712

hpx::lcos::local::latch::count_down_and_wait (C++ function), 711

hpx::lcos::local::latch::count_up (C++ function), 712

hpx::lcos::local::latch::HPX_NON_COPYABLE (C++ function), 711

hpx::lcos::local::latch::is_ready (C++ function), 712

hpx::lcos::local::latch::latch (C++ function), 711

hpx::lcos::local::latch::reset (C++ function), 712

hpx::lcos::local::latch::reset_if_needed (C++ function), 712

hpx::lcos::local::trigger (C++ struct), 663

hpx::lcos::local::trigger::base_type (C++ type), 664

hpx::lcos::local::trigger::operator= (C++ function), 663

hpx::lcos::local::trigger::synchronize (C++ function), 663

hpx::lcos::local::trigger::trigger (C++ function), 663

hpx::lcos::packaged_action (C++ class), 786

hpx::lcos::packaged_action::apply (C++ function), 786, 788

hpx::lcos::packaged_action::apply_cb (C++ function), 786, 788

hpx::lcos::packaged_action::apply_deferred (C++ function), 787

hpx::lcos::packaged_action::apply_deferred_cb (C++ type), 577

hpx::lcos::packaged_action::apply_p (C++ function), 787

hpx::lcos::packaged_action::apply_p_cb (C++ function), 787

hpx::lcos::packaged_action::do_apply (C++ function), 787

hpx::lcos::packaged_action::do_apply_cb (C++ function), 787

hpx::lcos::packaged_action::packaged_action (C++ function), 786, 788

hpx::lcos::packaged_action<Action, Result, false> (C++ class), 786

hpx::lcos::packaged_action<Action, Result, false>::action_type (C++ type), 787

hpx::lcos::packaged_action<Action, Result, false>::base_type (C++ type), 787

hpx::lcos::packaged_action<Action, Result, false>::remote_result_type (C++ type), 787

hpx::lcos::packaged_action<Action, Result, true> (C++ class), 787

hpx::lcos::packaged_action<Action, Result, true>::action_type (C++ type), 788

hpx::length_error (C++ enumerator), 609

hpx::lightweight (C++ enumerator), 618

hpx::lightweight (C++ member), 618

hpx::lightweight_rethrow (C++ member), 618

hpx::local (C++ enumerator), 678

hpx::lock_error (C++ enumerator), 607

hpx::make_error_code (C++ function), 609

hpx::make_success_code (C++ function), 610

hpx::migration (C++ function), 377–379

hpx::migration_needs_retry (C++ enumerator), 609

hpx::min_element (C++ function), 373–375

hpx::minmax_element (C++ function), 380–383

hpx::mpi (C++ type), 576, 577

hpx::mpi::experimental (C++ type), 576, 577

hpx::mpi::experimental::executor (C++ struct), 576

hpx::mpi::experimental::executor::async_execute (C++ function), 577

hpx::mpi::experimental::executor::communicator (C++ member), 577

hpx::mpi::experimental::executor::execution_category (C++ type), 577

hpx::mpi::experimental::executor::executor (C++ function), 577

hpx::mpi::experimental::executor::executor_parameter (C++ type), 577

```

hpx::mpi::experimental::executor::in_flight          hpx::parallel::execution::executor_execution_category
    (C++ function), 577                                (C++ struct), 851, 852
hpx::mpi::experimental::transform_mpi               hpx::parallel::execution::executor_parameters_join
    (C++ member), 577                                 (C++ struct), 623
hpx::mpi::experimental::transform_mpi_t             hpx::parallel::execution::executor_parameters_join
    (C++ struct), 577                                 (C++ struct), 624
hpx::naming (C++ type), 843, 871                  hpx::parallel::execution::executor_parameters_join
hpx::naming::operator<< (C++ function), 843        (C++ type), 624
hpx::naming::unmanaged (C++ function), 871         hpx::parallel::execution::executor_parameters_join
hpx::network_error (C++ enumerator), 607           (C++ type), 624
hpx::no_registered_console (C++ enumerator), 607   hpx::parallel::execution::extract_executor_parameter
                                                    (C++ struct), 641
hpx::no_state (C++ enumerator), 608                 hpx::parallel::execution::extract_executor_parameter
                                                    (C++ type), 640
hpx::no_success (C++ enumerator), 607              hpx::parallel::execution::extract_executor_parameter
                                                    Enable>::type (C++ type), 641
hpx::not_implemented (C++ enumerator), 607         hpx::parallel::execution::extract_executor_parameter
                                                    std::void_t<typename
hpx::null_thread_id (C++ enumerator), 608           Executor:::executor_parameters_type>>
hpx::operator& (C++ function), 618                  (C++ struct), 640, 641
hpx::out_of_memory (C++ enumerator), 607            hpx::parallel::execution::extract_executor_parameter
                                                    std::void_t<typename
hpx::out_of_range (C++ enumerator), 609              Executor:::executor_parameters_type>>::type
hpx::parallel (C++ type), 268, 330, 331, 418, 484, (C++ type), 640, 641
    542, 545–548, 551, 622–624, 629, 630, 640, hpx::parallel::execution::extract_has_variable_chunk
    642, 644, 648, 650, 697, 739, 743, 744, 852, (C++ member), 641
    860, 928                                         hpx::parallel::execution::extract_invokes_testing_
hpx::parallel::execution (C++ type), 622–          (C++ member), 641
    624, 629, 630, 640, 642, 644, 648, 650, 697, hpx::parallel::execution::extract_parallel::execution::get_chunk_size
    739, 743, 744, 852, 860                         (C++ member), 624
hpx::parallel::execution::async_execute_hpx::parallel::execution::get_parallel::execution::tag_fallba
    (C++ member), 740                                 (C++ function), 624
hpx::parallel::execution::async_execute_hpx::parallel::execution::get_parallel::execution::tag_fallba
    (C++ struct), 740                               (C++ struct), 624
hpx::parallel::execution::async_execute_hpx::parallel::execution::get_parallel::execution::tag_fallba
    (C++ function), 740                           (C++ function), 624
hpx::parallel::execution::create_reboundhpx::parallel::execution::get_parallel::execution::tag_fallba
    (C++ member), 630                             (C++ function), 622
hpx::parallel::execution::create_reboundhpx::parallel::execution::get_parallel::execution::tag_fallba
    (C++ struct), 630                           (C++ function), 622
hpx::parallel::execution::create_reboundhpx::parallel::execution::get_parallel::execution::has_pending_closures
    (C++ function), 631                         (C++ member), 622
hpx::parallel::execution::current_executhpx::parallel::execution::has_pending_closures_t
    (C++ type), 643                           (C++ struct), 622
hpx::parallel::execution::distribution_phpx::parallel::execution::has_pending_closures_t::t
    (C++ class), 860                         (C++ function), 623
hpx::parallel::execution::distribution_phpx::parallel::execution::has_pending_closures_t::t
    (C++ function), 860                         (C++ function), 623
hpx::parallel::execution::distribution_phpx::parallel::execution::has_pending_closures_t::t
    (C++ member), 861                         (C++ struct), 697
hpx::parallel::execution::executor_execuhpx::parallel::execution::io_pool_executor
    (C++ type), 851, 853                      (C++ function), 697

```

hpx::parallel::execution::io_thread_poolhpx::parallel::execution::parallel_policy_executor
(C++ enumerator), 697 (C++ type), 648, 649

hpx::parallel::execution::is_executor_parallel::parallel::execution::parallel_policy_executor
(C++ type), 640 (C++ type), 648, 649

hpx::parallel::execution::is_executor_parallel::parallel::execution::parallel_policy_executor
(C++ member), 641 (C++ type), 649

hpx::parallel::execution::is_timed_executorhpx::tparallel::execution::parallel_policy_executor
(C++ type), 744 (C++ type), 649

hpx::parallel::execution::is_timed_executorhpx::vparallel::execution::parallel_timed_executor
(C++ member), 744 (C++ type), 743

hpx::parallel::execution::join_executor_parallel::parallel::execution::parcel_pool_executor
(C++ function), 623 (C++ struct), 697

hpx::parallel::execution::main_pool_executorhpx::parallel::execution::parcel_pool_executor::par
(C++ struct), 697 (C++ function), 698

hpx::parallel::execution::main_pool_executorhpx::parallel::execution::parcel_thread_pool
(C++ function), 697 (C++ enumerator), 697

hpx::parallel::execution::main_thread hpx::parallel::execution::polymorphic_executor::ass
(C++ enumerator), 697 (C++ function), 630

hpx::parallel::execution::make_distribut~~hpx::parallel::execution::polymorphic_executor::asy~~
(C++ function), 860 (C++ function), 629

hpx::parallel::execution::mark_begin_exehpx::parallel::execution::polymorphic_executor::bu
(C++ member), 624 (C++ function), 629

hpx::parallel::execution::mark_begin_exehpx::parallel::execution::polymorphic_executor::bu
(C++ struct), 625 (C++ function), 629

hpx::parallel::execution::mark_begin_exehpx::parallel::execution::polymorphic_executor::bu
(C++ function), 625 (C++ function), 630

hpx::parallel::execution::mark_end_execuhpx::parallel::execution::polymorphic_executor::get
(C++ member), 624 (C++ function), 630

hpx::parallel::execution::mark_end_execuhpx::parallel::execution::polymorphic_executor::get
(C++ struct), 625 (C++ function), 630

hpx::parallel::execution::mark_end_execuhpx::parallel::execution::polymorphic_executor::ope
(C++ function), 625 (C++ function), 629

hpx::parallel::execution::mark_end_of_sch~~hpx::parallel::execution::polymorphic_executor::po~~
(C++ member), 624 (C++ function), 629

hpx::parallel::execution::mark_end_of_sch~~hpx::parallel::execution::polymorphic_executor::po~~
(C++ struct), 625 (C++ function), 629

hpx::parallel::execution::mark_end_of_sch~~hpx::parallel::execution::polymorphic_executor::re~~
(C++ function), 626 (C++ function), 629

hpx::parallel::execution::maximal_numberhpx::parallel::execution::polymorphic_executor::syn
(C++ member), 624 (C++ function), 629

hpx::parallel::execution::maximal_numberhpx::parallel::execution::polymorphic_executor::the
(C++ struct), 626 (C++ function), 629

hpx::parallel::execution::maximal_numberhpx::parallel::execution::polymorphic_executor<T>
(C++ function), 626 (C++ class), 629

hpx::parallel::execution::parallel_execuhpx::aggregated::execution::polymorphic_executor<T>
(C++ type), 649 (C++ type), 630

hpx::parallel::execution::parallel_policy~~hpx::parallel::execution::polymorphic_executor<T>~~
(C++ struct), 649 (C++ type), 629

hpx::parallel::execution::parallel_policy~~hpx::parallel::execution::polymorphic_executor<T>~~
(C++ function), 648–650 (C++ type), 630

hpx::parallel::execution::parallel_policy~~hpx::parallel::execution::polymorphic_executor_aggr~~
(C++ function), 648–650 (C++ member), 740

hpx::parallel::execution::parallel_policy~~hpx::parallel::execution::polymorphic_executor_aggr~~
(C++ struct), 648, 649 (C++ struct), 741

hpx::parallel::execution::post_after_t::tag_fal(C++ function), 698
hpx::parallel::execution::post_at (C++ member), 740
hpx::parallel::execution::post_at_t (C++ struct), 741
hpx::parallel::execution::post_at_t::tag_fallba(C++ member), 622
(C++ function), 742
hpx::parallel::execution::processing_units_coun(C++ struct), 623
(C++ member), 624
hpx::parallel::execution::processing_units_coun(C++ member), 740
(C++ struct), 626
hpx::parallel::execution::processing_units_coun(C++ struct), 742
(C++ function), 626
hpx::parallel::execution::rebind_executor (C++ function), 742
(C++ struct), 631
hpx::parallel::execution::rebind_executor<ExPol(C++ member), 740
Executor, Parameters>::type (C++ type), 631
hpx::parallel::execution::reset_thread_dhpkipatadmel::execution::sync_execute_at_t::tag_fal(C++ member), 624
(C++ function), 743
hpx::parallel::execution::reset_thread_dhpkipatadmel::execution::tag_fallback_invoke(C++ struct), 626
(C++ function), 644
hpx::parallel::execution::reset_thread_dhpkipatadmel::tag_fallback_invoke(C++ function), 627
(C++ function), 644
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::timed_executor (C++ class), 650
(C++ struct), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::timed_executor::async_exec(C++ type), 650
(C++ function), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::timed_executor::base_exec(C++ type), 650
(C++ function), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::thimed_executor::exec(C++ member), 651
(C++ member), 744
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::archimed_ekeeshold::execute_a(C++ member), 651
(C++ member), 744
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::archimed_ekeeshold::default_o(C++ member), 651
(C++ type), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::archimed_ekeeshold::parameter(C++ member), 651
(C++ type), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::thimed_executor::post(C++ member), 651
(C++ function), 744
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::timed_executor::sync_exec(C++ member), 651
(C++ function), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::timed_executor::timed_exec(C++ member), 651
(C++ function), 743
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::thimed_executor::post(C++ function), 650
(C++ struct), 698
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::thimed_executor::timed_executor(C++ member), 651
(C++ function), 698
hpx::parallel::execution::restricted_thrhpad::pperaekedutexecution::thimed_executor::timed_executor(C++ member), 651
(C++ enumerator), 697
hpx::parallel::execution::sequenced_timehpxepatafel::execution::with_processing_units_cou(C++ type), 743
(C++ member), 624
hpx::parallel::execution::service_executhpox::parallel::execution::with_processing_units_cou

(C++ struct), 627
hpx::parallel::util (C++ type), 542, 545–548,
 551
hpx::parallel::util::concat (C++ function),
 548
hpx::parallel::util::construct (C++ func-
 tion), 543
hpx::parallel::util::construct_object
 (C++ function), 542
hpx::parallel::util::destroy (C++ func-
 tion), 543
hpx::parallel::util::destroy_object
 (C++ function), 542
hpx::parallel::util::destroy_range (C++
 function), 549
hpx::parallel::util::full_merge (C++
 function), 544, 549
hpx::parallel::util::full_merge4 (C++
 function), 545
hpx::parallel::util::get_in2_element
 (C++ function), 551
hpx::parallel::util::get_pair (C++ func-
 tion), 551
hpx::parallel::util::get_second_element
 (C++ function), 551
hpx::parallel::util::get_subrange (C++
 function), 551
hpx::parallel::util::get_third_element
 (C++ function), 552
hpx::parallel::util::half_merge (C++
 function), 544, 550
hpx::parallel::util::in_fun_result (C++
 struct), 552
hpx::parallel::util::in_fun_result::operator
 in_fun_result<I2, F2> (C++ func-
 tion), 552
hpx::parallel::util::in_fun_result::seri-
 (C++ function), 552
hpx::parallel::util::in_in_out_result
 (C++ struct), 552
hpx::parallel::util::in_in_out_result::operator
 in_in_out_result<II1, II2, O1> (C++ func-
 tion), 553
hpx::parallel::util::in_in_out_result::seri-
 (C++ function), 553
hpx::parallel::util::in_in_result (C++
 struct), 553
hpx::parallel::util::in_in_result::operator
 in_in_result<II1, II2> (C++ func-
 tion), 553
hpx::parallel::util::in_in_result::seri-
 (C++ function), 553
hpx::parallel::util::in_out_out_result
 (C++ struct), 553
 hpx::parallel::util::in_out_out_result::operator
 in_out_out_result<II, O01, O02>
 (C++ function), 553
 hpx::parallel::util::in_out_out_result::serialize
 (C++ function), 553
 hpx::parallel::util::in_out_result (C++
 struct), 554
 hpx::parallel::util::in_out_result::operator
 in_out_result<I2, O2> (C++ func-
 tion), 554
 hpx::parallel::util::in_out_result::serialize
 (C++ function), 554
 hpx::parallel::util::in_place_merge
 (C++ function), 545, 550
 hpx::parallel::util::in_place_merge_uncontiguous
 (C++ function), 544, 550
 hpx::parallel::util::init (C++ function),
 543, 549
 hpx::parallel::util::init_move (C++ func-
 tion), 543, 548
 hpx::parallel::util::is_mergeable (C++
 function), 549
 hpx::parallel::util::less_range (C++
 function), 545
 hpx::parallel::util::make_in_out_out_result
 (C++ function), 552
 hpx::parallel::util::make_subrange (C++
 function), 552
 hpx::parallel::util::merge_flow (C++
 function), 551
 hpx::parallel::util::merge_level4 (C++
 function), 546
 hpx::parallel::util::merge_vector4 (C++
 function), 547
 hpx::parallel::util::min_max_result
 (C++ struct), 554
 hpx::parallel::util::min_max_result::operator
 min_max_result<T2> (C++ function),
 554
 hpx::parallel::util::min_max_result::serialize
 (C++ function), 554
 hpx::parallel::util::nbits32 (C++ func-
 tion), 547
 hpx::parallel::util::nbits64 (C++ func-
 tion), 547
 hpx::parallel::util::range (C++ type), 548
 hpx::parallel::util::tmsb (C++ member),
 548
 hpx::parallel::util::uninit_full_merge
 (C++ function), 544, 549
 hpx::parallel::util::uninit_full_merge4
 (C++ function), 546
 hpx::parallel::util::uninit_merge_level4
 (C++ function), 547

```

hpx::parallel::util::uninit_move  (C++ function), 543, 548
hpx::parallel::v1::minmax_element_result (C++ type), 928
hpx::parallel::v1::reduce_by_key   (C++ function), 418
hpx::parallel::v1::sort_by_key   (C++ function), 485
hpx::parallel::v1::sort_by_key_result (C++ type), 485
hpx::parallel::v2 (C++ type), 270
hpx::parallel::v2::define_task_block (C++ function), 268
hpx::parallel::v2::define_task_block_restore (C++ function), 269
hpx::parallel::v2::identity (C++ member), 332
hpx::parallel::v2::task_block (C++ class), 270
hpx::parallel::v2::task_block::get_execution_policy (C++ function), 271
hpx::parallel::v2::task_block::id_ (C++ member), 272
hpx::parallel::v2::task_block::policy (C++ function), 272
hpx::parallel::v2::task_block::policy_ (C++ member), 272
hpx::parallel::v2::task_block::run (C++ function), 271
hpx::parallel::v2::task_block::tasks_ (C++ member), 272
hpx::parallel::v2::task_block::wait (C++ function), 272
hpx::parallel::v2::task_block<ExPolicy>::execut (C++ member), 271
hpx::parallel::v2::task_canceled_exception (C++ class), 272
hpx::parallel::v2::task_canceled_exception (C++ function), 273
hpx::parcelset (C++ type), 872
hpx::parcelset::parcel_write_handler_type (C++ type), 872
hpx::parcelset::parcelport_background_moh (C++ enum), 873
hpx::parcelset::parcelport_background_moh (C++ enumerator), 873
hpx::parcelset::parcelport_background_moh (C++ enumerator), 873
hpx::parcelset::parcelport_background_moh (C++ enumerator), 873
hpx::parcelset::parcelport_background_moh (C++ enumerator), 873
hpx::performance_counters (C++ type), 873, 875, 882, 888, 892
hpx::parallel::performance_counters::add_counter_type (C++ function), 878, 885
hpx::parallel::performance_counters::agas_counter_discoverer (C++ function), 875
hpx::parallel::performance_counters::agas_raw_counter_creator (C++ function), 874
hpx::parallel::performance_counters::aggregating (C++ enumerator), 876, 877, 882, 883
hpx::parallel::performance_counters::average_base (C++ enumerator), 876, 877, 882, 883
hpx::parallel::performance_counters::average_count (C++ enumerator), 876, 877, 882, 883
hpx::parallel::performance_counters::average_timer (C++ enumerator), 876, 877, 882, 883
hpx::parallel::performance_counters::complement_counter_info (C++ function), 885
hpx::parallel::performance_counters::counter_aggregating (C++ member), 886
hpx::parallel::performance_counters::counter_average_base (C++ member), 886
hpx::parallel::performance_counters::counter_average_count (C++ member), 886
hpx::parallel::performance_counters::counter_average_timer (C++ member), 886
hpx::parallel::performance_counters::counter_elapsed_time (C++ member), 887
hpx::parallel::performance_counters::counter_histogram (C++ member), 887
hpx::parallel::performance_counters::counter_info (C++ struct), 879
hpx::parallel::performance_counters::counter_info::counter_info (C++ function), 879
hpx::parallel::performance_counters::counter_info::fullname_ (C++ member), 879
hpx::parallel::performance_counters::counter_info::execut (C++ member), 879
hpx::parallel::performance_counters::counter_info::help_text_ (C++ member), 879
hpx::parallel::performance_counters::counter_info::serialize (C++ member), 879
hpx::parallel::performance_counters::counter_info::tas (C++ member), 879
hpx::parallel::performance_counters::counter_info::version_ (C++ member), 879
hpx::parallel::performance_counters::counter_info::status_ (C++ member), 879
hpx::parallel::performance_counters::counter_info::unit_of_measurement_ (C++ member), 879
hpx::parallel::performance_counters::counter_info::version (C++ member), 879
hpx::parallel::performance_counters::counter_monotonically_increasing (C++ member), 886
hpx::parallel::performance_counters::counter_path_elements (C++ struct), 879
hpx::parallel::performance_counters::counter_path_elements::counter_path_elements (C++ type), 880
hpx::parallel::performance_counters::counter_path_elements::counter_path_elements (C++ function), 880

```

```
hpx::performance_counters::counter_path_hp::perf::im::a::ce::on::d::e::rs::counter_value::serialize
    (C++ member), 880                                (C++ function), 887
hpx::performance_counters::counter_path_hp::perf::im::a::ce::on::d::e::rs::counter_value::status_
    (C++ member), 880                                (C++ member), 887
hpx::performance_counters::counter_path_hp::perf::pa::an::te::in::sta::es::cb::a::n::a::l::e::value::time_
    (C++ member), 880                                (C++ member), 887
hpx::performance_counters::counter_path_hp::perf::pa::an::te::in::sta::es::c::u::n::t::er::value::value_
    (C++ member), 880                                (C++ member), 887
hpx::performance_counters::counter_path_hp::perf::pa::an::te::in::sta::es::c::a::n::t::er::values::array
    (C++ member), 880                                (C++ struct), 887
hpx::performance_counters::counter_path_hp::perf::f::or::m::a::i::c::e::counters::counter_values::array::co
    (C++ function), 880                                (C++ member), 888
hpx::performance_counters::counter_path_hp::perf::f::or::m::a::i::c::e::counters::counter_values::array::co
    (C++ member), 880                                (C++ function), 888
hpx::performance_counters::counter_path_hp::perf::f::or::m::a::i::c::e::counters::counter_values::array::co
    (C++ member), 880                                (C++ function), 888
hpx::performance_counters::counter_prefix hpx::performance_counters::counter_values::array::so
    (C++ member), 879                                (C++ member), 888
hpx::performance_counters::counter_prefix hpx::perf::ormance_counters::counter_values::array::so
    (C++ member), 879                                (C++ member), 888
hpx::performance_counters::counter_raw hpx::performance_counters::counter_values::array::se
    (C++ member), 886                                (C++ function), 888
hpx::performance_counters::counter_raw_v hpx::perf::ormance_counters::counter_values::array::st
    (C++ member), 887                                (C++ member), 888
hpx::performance_counters::counter_status hpx::perf::ormance_counters::counter_values::array::ti
    (C++ enum), 877, 883                            (C++ member), 888
hpx::performance_counters::counter_text hpx::perf::ormance_counters::counter_values::array::va
    (C++ member), 886                                (C++ member), 888
hpx::performance_counters::counter_type hpx::perf::ormance_counters::create_counter_func
    (C++ enum), 876, 882                            (C++ type), 875
hpx::performance_counters::counter_type_hp::perf::ormance_counters::default_counter_discover
    (C++ struct), 881                                (C++ function), 873
hpx::performance_counters::counter_type_hp::perf::ormance_counters::discover_counter_func
    (C++ function), 881                                (C++ type), 875
hpx::performance_counters::counter_type_hp::perf::ormance_counters::discover_counter_type
    (C++ member), 881                                (C++ function), 885, 886
hpx::performance_counters::counter_type_hp::perf::ormance_counters::discover_counter_types
    (C++ member), 881                                (C++ function), 885
hpx::performance_counters::counter_type_hp::perf::ormance_counters::discover_counters_full
    (C++ member), 881                                (C++ enumerator), 884
hpx::performance_counters::counter_type_hp::perf::ormance_counters::discover_counters_func
    (C++ function), 881                                (C++ type), 875
hpx::performance_counters::counter_value hpx::perf::ormance_counters::discover_counters_minima
    (C++ struct), 887                                (C++ enumerator), 884
hpx::performance_counters::counter_value hpx::perf::ormance_counters::discover_counters_mode
    (C++ member), 887                                (C++ enum), 884
hpx::performance_counters::counter_value hpx::perf::ormance_counters::elapsed_time
    (C++ function), 887                                (C++ enumerator), 876, 877, 883
hpx::performance_counters::counter_value hpx::perf::ormance_counters::ensure_counter_prefix
    (C++ function), 887                                (C++ function), 878
hpx::performance_counters::counter_value hpx::perf::ormance_counters::expand_counter_info
    (C++ member), 887                                (C++ function), 886
hpx::performance_counters::counter_value hpx::perf::ormance_counters::get_counter
    (C++ member), 887                                (C++ function), 878
```

hpx::performance_counters::get_counter_ahpx::performance_counters::registry
 (C++ function), 886
hpx::performance_counters::get_counter_ihpfx::performance_counters::registry::add_counter
 (C++ function), 886
hpx::performance_counters::get_counter_ihpfx::performance_counters::registry::add_counter_ty
 (C++ function), 885
hpx::performance_counters::get_counter_nhpfx::performance_counters::registry::clear
 (C++ function), 885
hpx::performance_counters::get_counter_phpx::performance_counters::registry::counter_data
 (C++ function), 885
hpx::performance_counters::get_counter_type::hpfx::performance_counters::registry::counter_data:
 (C++ function), 886
hpx::performance_counters::get_counter_type::hpfx::performance_counters::registry::counter_data:
 (C++ function), 878, 884, 885
hpx::performance_counters::get_counter_type::hpfx::performance_counters::registry::counter_data:
 (C++ member), 894
hpx::performance_counters::get_counter_type::hpfx::performance_counters::registry::counter_data:
 (C++ member), 885
hpx::performance_counters::get_full_counhpfx::performance_counters::registry::counter_data:
 (C++ function), 884
hpx::performance_counters::histogram hpx::performance_counters::registry::counter_type_r
 (C++ enumerator), 877, 883
hpx::performance_counters::install_counthpfx::performance_counters::registry::countertypes_
 (C++ function), 889–891
hpx::performance_counters::local_action_hpx::perfmon::counter::registry::create_arithme
 (C++ function), 875
hpx::performance_counters::local_action_hpx::perfmon::counter::registry::create_arithme
 (C++ function), 875
hpx::performance_counters::locality0_couhpfx::perfmon::counter::registry::create_counter
 (C++ function), 874
hpx::performance_counters::locality_counhpfx::perfmon::counter::registry::create_raw_cou
 (C++ function), 873
hpx::performance_counters::locality_numahpx::perfmon::counter::registry::create_raw_cou
 (C++ function), 874
hpx::performance_counters::locality_poolhpfx::perfmon::counter::registry::create_statist
 (C++ function), 873
hpx::performance_counters::locality_poolhpfx::perfmon::counter::registry::discover_count
 (C++ function), 874
hpx::performance_counters::locality_poolhpfx::perfmon::counter::registry::discover_count
 (C++ function), 874
hpx::performance_counters::locality_raw_hpntperfmon::counter::registry::get_counter_cr
 (C++ function), 874
hpx::performance_counters::locality_raw_hpntperfmon::counter::registry::get_counter_d
 (C++ function), 874
hpx::performance_counters::locality_threhpx::perfmon::counter::registry::get_counter_ty
 (C++ function), 874
hpx::performance_counters::monotonicallyhpfx::perfmon::counter::registry::instance
 (C++ enumerator), 876, 877, 882, 883
hpx::performance_counters::operator> hpx::performance_counters::registry::locate_counter
 (C++ function), 884
hpx::performance_counters::operator< hpx::performance_counters::registry::registry
 (C++ function), 884
hpx::performance_counters::raw (C++ enu- hpx::performance_counters::registry::remove_counter
 merator), 876, 877, 882, 883
hpx::performance_counters::raw_values hpx::performance_counters::registry::remove_counter
 (C++ enumerator), 877, 883
hpx::performance_counters::raw_values hpx::performance_counters::registry::remove_counter
 (C++ function), 892

hpx::performance_counters::remove_counter_prefunction), 637
(C++ function), 878 hpx::pre_exception_handler_type (C++
hpx::performance_counters::remove_counter_type type), 613
(C++ function), 886 hpx::promise_already_satisfied (C++ enumerator), 608
(C++ enumerator), 877, 878, 884 hpx::ranges (C++ type), 274, 278, 280, 282, 287,
hpx::performance_counters::status_counter_type unknown), 294, 296, 303, 305, 316, 320, 332,
(C++ enumerator), 878, 884 334, 337, 348, 351, 354, 361, 366, 368, 384,
hpx::performance_counters::status_counter_unknown), 298, 300, 301, 302, 303, 305, 316, 320, 332,
(C++ enumerator), 878, 884 388, 389, 393, 396, 401, 412, 420, 426, 441,
(C++ enumerator), 877, 878, 884 446, 452, 461, 464, 468, 472, 476, 479, 481,
hpx::performance_counters::status_generic_error 486, 489, 493, 496, 500, 506, 520, 524, 528,
(C++ enumerator), 878, 884 531, 536, 539
hpx::performance_counters::status_invalidated (C++ ranges::adjacent_difference (C++
(C++ function), 877, 878, 884 function), 274, 276
hpx::performance_counters::status_is_val(C++ function), 878 275, 276, 279
hpx::performance_counters::status_new_dahpx::ranges::all_of (C++ function), 281
(C++ enumerator), 877, 878, 883, 884 hpx::ranges::any_of (C++ function), 281
hpx::performance_counters::status_valid_hpx::ranges::copy (C++ function), 282, 283
(C++ enumerator), 877, 878, 883, 884 hpx::ranges::copy_if (C++ function), 284, 286
hpx::performance_counters::text (C++ ranges::copy_n (C++ function), 284
enumerator), 876, 877, 882, 883 hpx::ranges::count (C++ function), 287
hpx::plain (C++ enumerator), 618 hpx::ranges::count_if (C++ function), 288
hpx::plain (C++ member), 618 hpx::ranges::destroy (C++ function), 289
hpx::plugins (C++ type), 677, 895, 896 hpx::ranges::destroy_n (C++ function), 289
hpx::plugins::binary_filter_factory hpx::ranges::ends_with (C++ function), 290–
(C++ struct), 895 293
hpx::plugins::binary_filter_factory::~binary_filter_factory (C++ function), 294, 295
(C++ function), 895 hpx::ranges::exclusive_scan (C++ function),
hpx::plugins::binary_filter_factory::binary_factory 296, 302
(C++ function), 895 hpx::ranges::experimental (C++ type), 320
hpx::plugins::binary_filter_factory::create_hpx::ranges::experimental::for_loop
hpx::ranges::experimental::for_loop (C++ function), 321–324
(C++ function), 895
hpx::plugins::binary_filter_factory::glob_hpx::ranges::experimental::for_loop_strided
hpx::ranges::experimental::for_loop_strided (C++ function), 325, 327, 328
(C++ member), 896
hpx::plugins::binary_filter_factory::isehpx::ranges::fill (C++ function), 304
hpx::ranges::fill (C++ function), 304
(C++ member), 896 hpx::ranges::fill_n (C++ function), 304
hpx::plugins::binary_filter_factory::lock_hpx::ranges::find (C++ function), 305, 306
hpx::ranges::find (C++ function), 305, 306
(C++ member), 896 hpx::ranges::find_end (C++ function), 309, 311
hpx::plugins::plugin_registry (C++ hpx::ranges::find_first_of (C++ function),
struct), 896 312, 314
hpx::plugins::plugin_registry::get_plugin_hpx::ranges::find_if (C++ function), 306, 307
(C++ function), 897 hpx::ranges::find_if_not (C++ function), 308,
hpx::plugins::plugin_registry_base (C++ 309
struct), 677 hpx::ranges::for_each (C++ function), 316–318
hpx::plugins::plugin_registry_base::~plugin_hpx::ranges::generate (C++ function), 332, 333
hpx::ranges::baseeach_n (C++ function), 319
(C++ function), 677 hpx::ranges::generate_n (C++ function), 334
hpx::plugins::plugin_registry_base::get_pluginname_hpx::ranges::includes (C++ function), 335, 336
(C++ function), 677 hpx::ranges::inclusive_scan (C++ function),
hpx::plugins::plugin_registry_base::init 337–344, 346
(C++ function), 677
hpx::post (C++ member), 633 hpx::ranges::inplace_merge (C++ function),
hpx::post_t (C++ struct), 636 371, 372
hpx::post_t::tagFallback_invoke (C++ hpx::ranges::is_heap (C++ function), 348, 349

hpx::ranges::is_heap_until (<i>C++ function</i>), 349, 350	478
hpx::ranges::is_partitioned (<i>C++ function</i>), 351–353	hpx::ranges::shift_right (<i>C++ function</i>), 479, 480
hpx::ranges::is_sorted (<i>C++ function</i>), 354–356	hpx::ranges::sort (<i>C++ function</i>), 481–483
hpx::ranges::is_sorted_until (<i>C++ function</i>), 357–359	hpx::ranges::stable_partition (<i>C++ function</i>), 404–407
hpx::ranges::lexicographical_compare (<i>C++ function</i>), 361–364	hpx::ranges::stable_sort (<i>C++ function</i>), 486–488
hpx::ranges::make_heap (<i>C++ function</i>), 366, 367	hpx::ranges::starts_with (<i>C++ function</i>), 490–492
hpx::ranges::merge (<i>C++ function</i>), 368, 369	hpx::ranges::swap_ranges (<i>C++ function</i>), 493–495
hpx::ranges::mismatch (<i>C++ function</i>), 385, 386	hpx::ranges::transform (<i>C++ function</i>), 496–498
hpx::ranges::move (<i>C++ function</i>), 388	hpx::ranges::transform_exclusive_scan (<i>C++ function</i>), 500, 501, 503, 504
hpx::ranges::none_of (<i>C++ function</i>), 280	hpx::ranges::transform_inclusive_scan (<i>C++ function</i>), 506–509, 511–513, 515
hpx::ranges::nth_element (<i>C++ function</i>), 390–392	hpx::ranges::uninitialized_copy (<i>C++ function</i>), 520–522
hpx::ranges::partial_sort (<i>C++ function</i>), 393, 394	hpx::ranges::uninitialized_copy_n (<i>C++ function</i>), 523
hpx::ranges::partial_sort_copy (<i>C++ function</i>), 396–399	hpx::ranges::uninitialized_default_construct (<i>C++ function</i>), 524–526
hpx::ranges::partition (<i>C++ function</i>), 401–403	hpx::ranges::uninitialized_default_construct_n (<i>C++ function</i>), 526, 527
hpx::ranges::partition_copy (<i>C++ function</i>), 408–411	hpx::ranges::uninitialized_fill (<i>C++ function</i>), 528, 529
hpx::ranges::reduce (<i>C++ function</i>), 413–417	hpx::ranges::uninitialized_fill_n (<i>C++ function</i>), 530
hpx::ranges::remove (<i>C++ function</i>), 420–422	hpx::ranges::uninitialized_move (<i>C++ function</i>), 532–534
hpx::ranges::remove_if (<i>C++ function</i>), 423–425	hpx::ranges::uninitialized_move_n (<i>C++ function</i>), 534, 535
hpx::ranges::replace (<i>C++ function</i>), 426–428	hpx::ranges::uninitialized_value_construct (<i>C++ function</i>), 536–538
hpx::ranges::replace_copy (<i>C++ function</i>), 433, 434, 436	hpx::ranges::uninitialized_value_construct_n (<i>C++ function</i>), 538, 539
hpx::ranges::replace_copy_if (<i>C++ function</i>), 437–440	hpx::ranges::unique (<i>C++ function</i>), 540, 541
hpx::ranges::replace_if (<i>C++ function</i>), 429–431	hpx::recursive_mutex (<i>C++ type</i>), 712
hpx::ranges::reverse (<i>C++ function</i>), 441–443	hpx::register_on_exit (<i>C++ function</i>), 696
hpx::ranges::reverse_copy (<i>C++ function</i>), 443–445	hpx::register_pre_shutdown_function (<i>C++ function</i>), 698
hpx::ranges::rotate (<i>C++ function</i>), 446–448	hpx::register_pre_startup_function (<i>C++ function</i>), 699
hpx::ranges::rotate_copy (<i>C++ function</i>), 449–450	hpx::register_shutdown_function (<i>C++ function</i>), 698
hpx::ranges::search (<i>C++ function</i>), 452–455	hpx::register_startup_function (<i>C++ function</i>), 699
hpx::ranges::search_n (<i>C++ function</i>), 456–459	hpx::register_thread (<i>C++ function</i>), 695
hpx::ranges::set_difference (<i>C++ function</i>), 461, 462	hpx::register_thread_on_error_func (<i>C++ function</i>), 701
hpx::ranges::set_intersection (<i>C++ function</i>), 465, 466	hpx::register_thread_on_start_func (<i>C++ function</i>), 701
hpx::ranges::set_symmetric_difference (<i>C++ function</i>), 469, 470	
hpx::ranges::set_union (<i>C++ function</i>), 472, 474	
hpx::ranges::shift_left (<i>C++ function</i>), 476–478	

692
`hpx::runtime::call_startup_functions` (*C++ function*), 694
`hpx::runtime::deinit_global_data` (*C++ function*), 693
`hpx::runtime::deinit_tss_helper` (*C++ function*), 694
`hpx::runtime::enumerate_os_threads` (*C++ function*), 692
`hpx::runtime::exception_` (*C++ member*), 693
`hpx::runtime::finalize` (*C++ function*), 690
`hpx::runtime::get_config` (*C++ function*), 689
`hpx::runtime::get_initial_num_localities` (*C++ function*), 692
`hpx::runtime::get_instance_number` (*C++ function*), 689
`hpx::runtime::get_locality_id` (*C++ function*), 692
`hpx::runtime::get_locality_name` (*C++ function*), 692
`hpx::runtime::get_notification_policy` (*C++ function*), 688
`hpx::runtime::get_num_localities` (*C++ function*), 692
`hpx::runtime::get_num_worker_threads` (*C++ function*), 692
`hpx::runtime::get_os_thread_data` (*C++ function*), 692
`hpx::runtime::get_state` (*C++ function*), 688
`hpx::runtime::get_system_uptime` (*C++ function*), 693
`hpx::runtime::get_thread_manager` (*C++ function*), 690
`hpx::runtime::get_thread_mapper` (*C++ function*), 689
`hpx::runtime::get_thread_pool` (*C++ function*), 691
`hpx::runtime::get_topology` (*C++ function*), 689
`hpx::runtime::here` (*C++ function*), 690
`hpx::runtime::hpx_error_sink_function_type` (*C++ type*), 688
`hpx::runtime::hpx_main_function_type` (*C++ type*), 688
`hpx::runtime::init` (*C++ function*), 693
`hpx::runtime::init_global_data` (*C++ function*), 693
`hpx::runtime::init_tss_ex` (*C++ function*), 694
`hpx::runtime::init_tss_helper` (*C++ function*), 694
`hpx::runtime::instance_number_` (*C++ member*), 693
`hpx::runtime::instance_number_counter_` (*C++ member*), 694
`hpx::runtime::is_networking_enabled` (*C++ function*), 690
`hpx::runtime::main_pool_` (*C++ member*), 693
`hpx::runtime::main_pool_notifier_` (*C++ member*), 693
`hpx::runtime::mtx_` (*C++ member*), 693
`hpx::runtime::notification_policy_type` (*C++ type*), 688
`hpx::runtime::notifier_` (*C++ member*), 693
`hpx::runtime::notify_finalize` (*C++ function*), 694
`hpx::runtime::on_error_func` (*C++ function*), 692
`hpx::runtime::on_error_func_` (*C++ member*), 693
`hpx::runtime::on_exit` (*C++ function*), 688
`hpx::runtime::on_exit_functions_` (*C++ member*), 693
`hpx::runtime::on_exit_type` (*C++ type*), 693
`hpx::runtime::on_start_func` (*C++ function*), 692
`hpx::runtime::on_start_func_` (*C++ member*), 693
`hpx::runtime::on_stop_func` (*C++ function*), 692
`hpx::runtime::on_stop_func_` (*C++ member*), 693
`hpx::runtime::pre_shutdown_functions_` (*C++ member*), 694
`hpx::runtime::pre_startup_functions_` (*C++ member*), 694
`hpx::runtime::register_thread` (*C++ function*), 691
`hpx::runtime::report_error` (*C++ function*), 690
`hpx::runtime::result_` (*C++ member*), 693
`hpx::runtime::resume` (*C++ function*), 690
`hpx::runtime::rethrow_exception` (*C++ function*), 689
`hpx::runtime::rtcfg_` (*C++ member*), 693
`hpx::runtime::run` (*C++ function*), 689
`hpx::runtime::run_helper` (*C++ function*), 693
`hpx::runtime::runtime` (*C++ function*), 688, 693
`hpx::runtime::set_notification_policies` (*C++ function*), 693
`hpx::runtime::set_state` (*C++ function*), 688
`hpx::runtime::shutdown_functions_` (*C++ member*), 694
`hpx::runtime::start` (*C++ function*), 689
`hpx::runtime::starting` (*C++ function*), 688
`hpx::runtime::startup_functions_` (*C++ member*), 694
`hpx::runtime::state_` (*C++ member*), 693

hpx::runtime::stop (*C++ function*), 690
hpx::runtime::stop_called_ (*C++ member*),
 694
hpx::runtime::stop_done_ (*C++ member*), 694
hpx::runtime::stop_helper (*C++ function*),
 694
hpx::runtime::stopped (*C++ function*), 688
hpx::runtime::stopping (*C++ function*), 688
hpx::runtime::suspend (*C++ function*), 690
hpx::runtime::thread_manager_ (*C++ mem-
ber*), 693
hpx::runtime::thread_support_ (*C++ mem-
ber*), 693
hpx::runtime::topology_ (*C++ member*), 693
hpx::runtime::unregister_thread (*C++ func-
tion*), 692
hpx::runtime::wait (*C++ function*), 690
hpx::runtime::wait_condition_ (*C++ mem-
ber*), 694
hpx::runtime::wait_finalize (*C++ function*),
 694
hpx::runtime::wait_helper (*C++ function*),
 693
hpx::runtime_distributed (*C++ class*), 905
hpx::runtime_distributed::~runtime_distr
 hpx::runtime_distributed::get_thread_pool
 (*C++ function*), 908
hpx::runtime_distributed::active_counter
 hpx::runtime_distributed::here (*C++ func-
tion*), 907
 (*C++ member*), 910
hpx::runtime_distributed::add_pre_shutdown
 hpx::runtime_distributed::id_pool_ (*C++
function*), 908
hpx::runtime_distributed::add_pre_startup
 hpx::runtime_distributed::init_global_data
 (*C++ function*), 909
hpx::runtime_distributed::add_shutdown_f
 hpx::runtime_distributed::init_id_pool_range
 (*C++ function*), 908
hpx::runtime_distributed::add_startup_fu
 hpx::runtime_distributed::init_tss_ex
 (*C++ function*), 909
hpx::runtime_distributed::agas_client_
 hpx::runtime_distributed::init_tss_helper
 (*C++ member*), 910
 (*C++ function*), 909
hpx::runtime_distributed::applier_ (*C++
member*), 910
hpx::runtime_distributed::assign_cores
 hpx::runtime_distributed::is_networking_enabled
 (*C++ function*), 907
hpx::runtime_distributed::default_errorsh
 hpx::runtime_distributed::mode_ (*C++
function*), 910
hpx::runtime_distributed::deinit_global_h
 hpx::runtime_distributed::post_main_
 (*C++ function*), 909
hpx::runtime_distributed::deinit_tss_hel
 hpx::runtime_distributed::pre_main_
 (*C++ function*), 910
hpx::runtime_distributed::evaluate_activ
 hpx::runtime_distributed::register_counter_types
 (*C++ function*), 907
hpx::runtime_distributed::finalize (*C++
function*), 906
hpx::runtime_distributed::get_agas_clien
 hpx::runtime_distributed::register_query_counters
 (*C++ function*), 907
 (*C++ function*), 907
hpx::runtime_distributed::register_thread
 hpx::runtime_distributed::register_thread
 (*C++ function*), 908

```

hpx::runtime_distributed::reinit_active_counter(C++ function), 703, 704
    (C++ function), 907
hpx::runtime_distributed::report_error
    (C++ function), 906
hpx::runtime_distributed::reset_active_counters(C++ function), 703, 704
    (C++ function), 907
hpx::runtime_distributed::resume  (C++ function), 906
hpx::runtime_distributed::run  (C++ function), 906, 907
hpx::runtime_distributed::run_helper
    (C++ function), 909
hpx::runtime_distributed::runtime_support
    (C++ member), 910
hpx::runtime_distributed::set_error_sink
    (C++ function), 907
hpx::runtime_distributed::start  (C++ function), 905
hpx::runtime_distributed::start_active_chapter
    (C++ function), 907
hpx::runtime_distributed::stop (C++ function), 906
hpx::runtime_distributed::stop_active_cooperation
    (C++ function), 907
hpx::runtime_distributed::stop_evaluating
    (C++ function), 907
hpx::runtime_distributed::stop_helper
    (C++ function), 906
hpx::runtime_distributed::suspend (C++ function), 906
hpx::runtime_distributed::used_cores_map
    (C++ member), 910
hpx::runtime_distributed::used_cores_map
    (C++ type), 909
hpx::runtime_distributed::wait (C++ function), 905
hpx::runtime_distributed::wait_helper
    (C++ function), 909
hpx::runtime_mode (C++ enum), 678
hpx::segmented (C++ type), 922–929, 931, 932
hpx::segmented::minmax_element_result
    (C++ type), 928
hpx::segmented::tag_invoke (C++ function),
    923–933
hpx::serialization (C++ type), 704
hpx::serialization::base_object
    (C++ function), 704
hpx::serialization::base_object_type
    (C++ struct), 704
hpx::serialization::base_object_type::base_object
    (C++ function), 703, 704
hpx::serialization::base_object_type::d_hpx
    (C++ member), 704, 705
hpx::serialization::base_object_type::HPX_SERIALIZEABLE_MEMBER
    (C++ member), 607
        hpx::serialization::base_object_type::load
            (C++ function), 703, 704
        hpx::serialization::base_object_type::save
            (C++ function), 704
        hpx::serialization::base_object_type::serialize
            (C++ function), 704
        hpx::serialization::base_object_type<Derived,
            Base, std::true_type> (C++ struct),
            703, 704
        hpx::serialization::operator& (C++ function), 704
hpx::serialization::operator>> (C++ function), 704
hpx::serialization::operator<< (C++ function), 704
hpx::serialization_error (C++ enumerator),
    608
hpx::set_custom_exception_info_handler
    (C++ function), 613
hpx::set_error_handlers (C++ function), 688
    hpx::set_lco_error (C++ function), 794–796
hpx::set_co_value (C++ function), 792, 793
    hpx::set_lco_value_unmanaged (C++ function), 792, 794
hpx::set_pre_exception_handler (C++ function), 613
    hpx::shutdown_function_type (C++ type), 698
hpx::sliding_semaphore
    (C++ type), 713
    hpx::sliding_semaphore_var (C++ class), 713
hpx::sliding_semaphore_var::mtx_
    (C++ member), 714
hpx::sliding_semaphore_var::sem_
    (C++ member), 714
hpx::sliding_semaphore_var::set_max_difference
    (C++ function), 713
hpx::sliding_semaphore_var::signal (C++ function), 714
hpx::sliding_semaphore_var::signal_all
    (C++ function), 714
hpx::sliding_semaphore_var::sliding_semaphore_var
    (C++ function), 713
hpx::sliding_semaphore_var::try_wait
    (C++ function), 714
hpx::sliding_semaphore_var::wait  (C++ function), 713
    hpx::sliding_semaphore_var<Mutex>::mutex_type
        (C++ type), 714
        hpx::split_future (C++ function), 559
        hpx::start (C++ function), 867–869
        hpx::startup_function_type (C++ type), 699

```

hpx::stop (*C++ function*), 863
hpx::success (*C++ enumerator*), 607
hpx::suspend (*C++ function*), 870
hpx::sync_execute (*C++ member*), 633
hpx::sync_execute_t (*C++ struct*), 637
hpx::sync_execute_t::tagFallbackInvoke
 (*C++ function*), 637
hpx::task_already_started (*C++ enumerator*),
 608
hpx::task_block_not_active (*C++ enumera-
 tor*), 609
hpx::task_canceled_exception (*C++ enumera-
 tor*), 609
hpx::task_moved (*C++ enumerator*), 608
hpx::terminate (*C++ function*), 862
hpx::then_execute (*C++ member*), 633
hpx::then_execute_t (*C++ struct*), 637
hpx::then_execute_t::tagFallbackInvoke
 (*C++ function*), 638
hpx::this_thread (*C++ type*), 643, 725
hpx::this_thread::get_executor (*C++ func-
 tion*), 643
hpx::this_thread::get_pool (*C++ function*),
 728
hpx::this_thread::suspend (*C++ function*),
 725–727
hpx::thread_cancelled (*C++ enumerator*), 608
hpx::thread_interrupted (*C++ struct*), 617
hpx::thread_not_interruptable (*C++ enu-
 merator*), 608
hpx::thread_resource_error (*C++ enumera-
 tor*), 608
hpx::threads (*C++ type*), 598, 602, 643, 694, 702,
 714, 717, 719, 724, 728, 734, 736, 745
hpx::threads::abort (*C++ enumerator*), 600
hpx::threads::active (*C++ enumerator*), 599
hpx::threads::boost (*C++ enumerator*), 599
hpx::threads::bound (*C++ enumerator*), 599
hpx::threads::create_topology (*C++ func-
 tion*), 745
hpx::threads::current (*C++ enumerator*), 600
hpx::threads::default_ (*C++ enumera-
 tor*),
 599, 600
hpx::threads::depleted (*C++ enumerator*), 599
hpx::threads::enumerate_threads
 (*C++ function*), 703
hpx::threads::get_ctx_ptr (*C++ function*),
 719
hpx::threads::get_default_stack_size
 (*C++ function*), 695
hpx::threads::get_executor (*C++ function*),
 643
hpx::threads::get_idle_core_count
 (*C++ function*), 703
hpx::threads::get_idle_core_mask
 (*C++ function*), 703
hpx::threads::get_memory_page_size
 (*C++ function*), 745
hpx::threads::get_parent_id (*C++ function*),
 719
hpx::threads::get_parent_locality_id
 (*C++ function*), 720
hpx::threads::get_parent_phase
 (*C++ func-
 tion*), 720
hpx::threads::get_pool (*C++ function*), 732
hpx::threads::get_self (*C++ function*), 719
hpx::threads::get_self_component_id
 (*C++ function*), 720
hpx::threads::get_self_id
 (*C++ function*),
 719
hpx::threads::get_self_id_data
 (*C++ func-
 tion*), 719
hpx::threads::get_self_ptr
 (*C++ function*),
 719
hpx::threads::get_self_ptr_checked
 (*C++ function*), 719
hpx::threads::get_self_stacksize
 (*C++ function*), 720
hpx::threads::get_self_stacksize_enum
 (*C++ function*), 720
hpx::threads::get_stack_size
 (*C++ func-
 tion*), 695, 731
hpx::threads::get_stack_size_enum_name
 (*C++ function*), 601
hpx::threads::get_stack_size_name
 (*C++ function*), 695
hpx::threads::get_thread_count
 (*C++ func-
 tion*), 702
hpx::threads::get_thread_description
 (*C++ function*), 724
hpx::threads::get_thread_id_data
 (*C++ function*), 719
hpx::threads::get_thread_interruption_enabled
 (*C++ function*), 730
hpx::threads::get_thread_interruption_requested
 (*C++ function*), 730
hpx::threads::get_thread_lco_description
 (*C++ function*), 724
hpx::threads::get_thread_phase
 (*C++ func-
 tion*), 730
hpx::threads::get_thread_priority
 (*C++ function*), 731
hpx::threads::get_thread_priority_name
 (*C++ function*), 601
hpx::threads::get_thread_state
 (*C++ func-
 tion*), 729
hpx::threads::get_thread_state_ex_name
 (*C++ function*), 601

hpx::threads::thread(C++ enumerator), 600 hpx::threads::thread_data::interrupt
hpx::threads::thread_data(C++ class), 720 (C++ function), 722
hpx::threads::thread_data::~thread_data hpx::threads::thread_data::interruption_enabled
(C++ function), 723 (C++ function), 722
hpx::threads::thread_data::add_thread_ex hpx::threads::thread_data::interruption_point
(C++ function), 722 (C++ function), 722
hpx::threads::thread_data::current_state hpx::threads::thread_data::interruption_requested
(C++ member), 723 (C++ function), 722
hpx::threads::thread_data::destroy(C++ function), 723 hpx::threads::thread_data::is_stackless
(C++ function), 723 (C++ function), 722
hpx::threads::thread_data::destroy_thread hpx::threads::thread_data::is_stackless_
(C++ function), 722 (C++ member), 723
hpx::threads::thread_data::enabled_inter hpx::threads::thread_data::last_worker_thread_num_
(C++ member), 723 (C++ member), 723
hpx::threads::thread_data::exit_funcs_ hpx::threads::thread_data::operator()
(C++ member), 723 (C++ function), 722
hpx::threads::thread_data::free_thread_ex hpx::threads::thread_data::operator=
(C++ function), 722 (C++ function), 721
hpx::threads::thread_data::get_backtrace hpx::threads::thread_data::priority_
(C++ function), 722 (C++ member), 723
hpx::threads::thread_data::get_component hpx::threads::thread_data::queue_
(C++ function), 721 (C++ member), 724
hpx::threads::thread_data::get_descripti hpx::threads::thread_data::ran_exit_funcs_
(C++ function), 722 (C++ member), 723
hpx::threads::thread_data::get_last_work hpx::threads::thread_data::rebind (C++
(C++ function), 722 function), 723
hpx::threads::thread_data::get_lco_descr hpx::threads::thread_data::rebind_base
(C++ function), 722 (C++ function), 723
hpx::threads::thread_data::get_parent_lo hpx::threads::thread_data::requested_interrupt_
(C++ function), 722 (C++ member), 723
hpx::threads::thread_data::get_parent_th hpx::threads::thread_data::restore_state
(C++ function), 722 (C++ function), 721
hpx::threads::thread_data::get_parent_th hpx::threads::thread_data::run_thread_exit_callback
(C++ function), 722 (C++ function), 722
hpx::threads::thread_data::get_priority hpx::threads::thread_data::scheduler_base_
(C++ function), 722 (C++ member), 723
hpx::threads::thread_data::get_queue hpx::threads::thread_data::set_backtrace
(C++ function), 722 (C++ function), 722
hpx::threads::thread_data::get_scheduler hpx::threads::thread_data::set_description
(C++ function), 722 (C++ function), 722
hpx::threads::thread_data::get_stack_siz hpx::threads::thread_data::set_interruption_enabled
(C++ function), 722 (C++ function), 722
hpx::threads::thread_data::get_stack_siz hpx::threads::thread_data::set_last_worker_thread_n
(C++ function), 722 (C++ function), 722
hpx::threads::thread_data::get_state hpx::threads::thread_data::set_lco_description
(C++ function), 721 (C++ function), 722
hpx::threads::thread_data::get_thread_da hpx::threads::thread_data::set_priority
(C++ function), 723 (C++ function), 722
hpx::threads::thread_data::get_thread_id hpx::threads::thread_data::set_state
(C++ function), 723 (C++ function), 721
hpx::threads::thread_data::get_thread_ph hpx::threads::thread_data::set_state_ex
(C++ function), 723 (C++ function), 723
hpx::threads::thread_data::init (C++ hpx::threads::thread_data::set_state_tagged
function), 723 (C++ function), 721

hpx::threads::thread_data::set_thread_data (*C++ function*), 734
 (*C++ function*), 723
 hpx::threads::thread_data::spinlock_pool (*C++ type*), 720
 hpx::threads::thread_data::stacksize_ (*C++ member*), 723
 hpx::threads::thread_data::stacksize_enum_ (*C++ member*), 723
 hpx::threads::thread_data::thread_data (*C++ function*), 721, 723
 hpx::threads::thread_id (*C++ struct*), 603
 hpx::threads::thread_id::get (*C++ function*), 603
 hpx::threads::thread_id::operator bool (*C++ function*), 603
 hpx::threads::thread_id::operator= (*C++ function*), 603
 hpx::threads::thread_id::reset (*C++ function*), 603
 hpx::threads::thread_id::thrd_ (*C++ member*), 603
 hpx::threads::thread_id::thread_id (*C++ function*), 603
 hpx::threads::thread_id::thread_id_repr (*C++ type*), 603
 hpx::threads::thread_id_addrref (*C++ enum*), 603
 hpx::threads::thread_id_ref (*C++ struct*), 604
 hpx::threads::thread_id_ref::detach (*C++ function*), 605
 hpx::threads::thread_id_ref::get (*C++ function*), 605
 hpx::threads::thread_id_ref::noref (*C++ function*), 605
 hpx::threads::thread_id_ref::operator bool (*C++ function*), 605
 hpx::threads::thread_id_ref::operator= (*C++ function*), 604, 605
 hpx::threads::thread_id_ref::reset (*C++ function*), 605
 hpx::threads::thread_id_ref::thrd_ (*C++ member*), 605
 hpx::threads::thread_id_ref::thread_id_repr (*C++ function*), 604, 605
 hpx::threads::thread_id_ref::thread_id_rhp^x
 (*C++ type*), 605
 hpx::threads::thread_id_ref::thread_reprhp^x
 (*C++ type*), 604
 hpx::threads::thread_pool_base (*C++ class*), 734
 hpx::threads::thread_pool_base::resume_dhp^x
 (*C++ function*), 734
 hpx::threads::thread_pool_base::resume_php^x
 (*C++ function*), 734
 (*C++ struct*), 734
 hpx::threads::thread_pool_base::suspend_direct
 (*C++ function*), 734
 hpx::threads::thread_pool_base::suspend_processing_ (*C++ function*), 734
 hpx::threads::thread_pool_init_parameters
 (*C++ struct*), 734
 hpx::threads::thread_pool_init_parameters::affinity_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::index_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::max_backlog_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::max_busy_time_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::max_idle_time_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::mode_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::name_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::network_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::notifier_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::num_threads_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::shutdown_time_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::thread_count_ (*C++ member*), 735
 hpx::threads::thread_pool_init_parameters::thread_id_ (*C++ function*), 735
 hpx::threads::thread_priority (*C++ enum*), 599
 hpx::threads::thread_restart_state (*C++ enum*), 600
 hpx::threads::thread_schedule_hint (*C++ struct*), 601
 hpx::threads::thread_schedule_hint::hint_ (*C++ member*), 602
 hpx::threads::thread_schedule_hint::mode_ (*C++ member*), 602
 hpx::threads::thread_schedule_hint::thread_schedule_ (*C++ function*), 602
 hpx::threads::thread_schedule_hint_mode_ (*C++ enum*), 600
 hpx::threads::thread_schedule_state (*C++ enum*), 599
 hpx::threads::thread_stacksize_ (*C++ enum*), 600

(C++ function), 736
hpx::threads::threadmanager::abort_all_shpspentdeadseathreadmanager::init_tss
(C++ function), 738
hpx::threads::threadmanager::add_remove_hpkedtheeardeatthreadmanager::is_busy
(C++ function), 738
hpx::threads::threadmanager::add_schedulhpxmodhreads::threadmanager::is_idle
(C++ function), 738
hpx::threads::threadmanager::cleanup_terminhpxtatedhreads::threadmanager::mtx_
(C++ member), 739
hpx::threads::threadmanager::create_poolhpx::threads::threadmanager::mutex_type
(C++ function), 736
hpx::threads::threadmanager::default_poohpxtatedhreads::threadmanager::network_background_ca
(C++ function), 736
hpx::threads::threadmanager::default_schaduledhreads::threadmanager::notification_policy_ty
(C++ function), 736
hpx::threads::threadmanager::deinit_tss hpx::threads::threadmanager::notifier_
(C++ function), 738
hpx::threads::threadmanager::enumerate_thptheadshreads::threadmanager::pool_exists
(C++ function), 738
hpx::threads::threadmanager::get_backgrohpdlthreaddsthreadmanager::pool_type
(C++ function), 738
hpx::threads::threadmanager::get_cumulathpexdtheadshreads::threadmanager::pool_vector
(C++ function), 738
hpx::threads::threadmanager::get_idle_colhxcothreads::threadmanager::pools_
(C++ function), 738
hpx::threads::threadmanager::get_idle_colhxmathtreads::threadmanager::print_pools
(C++ function), 738
hpx::threads::threadmanager::get_os_threhpxccothreads::threadmanager::register_thread
(C++ function), 738
hpx::threads::threadmanager::get_os_threhpxhandhreads::threadmanager::register_work
(C++ function), 738
hpx::threads::threadmanager::get_pool hpx::threads::threadmanager::remove_scheduler_mode
(C++ function), 736
hpx::threads::threadmanager::get_pool_numbpxbithreads::threadmanager::report_error
(C++ function), 738
hpx::threads::threadmanager::get_queue_lhpstthreads::threadmanager::reset_thread_distribut
(C++ function), 738
hpx::threads::threadmanager::get_thread_hpntthreade::threadmanager::resume
(C++ function), 737
hpx::threads::threadmanager::get_thread_hpntthreade::threadmanager::rtcfg_
(C++ function), 739
hpx::threads::threadmanager::get_thread_hpntthreadding::threadmanager::run (C++
function), 739
hpx::threads::threadmanager::get_thread_hpntthreaged::threadmanager::scheduler_type
(C++ function), 739
hpx::threads::threadmanager::get_thread_hpntthreagedhreadmanager::set_scheduler_mode
(C++ function), 739
hpx::threads::threadmanager::get_thread_hpntthreagedhreadmanager::status
(C++ function), 739
hpx::threads::threadmanager::get_thread_hpntthreagedhreadmanager::stop (C++
function), 738
hpx::threads::threadmanager::get_used_prbpasssthgeadiststhreadmanager::suspend
(C++ function), 738
hpx::threads::threadmanager::init (C++ hpx::threads::threadmanager::threadmanager

hpx::threads::threadmanager::threads_lookup
 (C++ member), 739
 hpx::threads::threadmanager::wait (C++ function), 737
 hpx::threads::timeout (C++ enumerator), 600
 hpx::threads::topology (C++ struct), 746
 hpx::threads::topology::~topology (C++ function), 746
 hpx::threads::topology::allocate (C++ function), 749
 hpx::threads::topology::allocate_membind
 (C++ function), 749
 hpx::threads::topology::bitmap_to_mask hpx::threads::topology::get_pu_number
 (C++ function), 749
 hpx::threads::topology::core_affinity_mask
 (C++ member), 750
 hpx::threads::topology::core_numbers_ hpx::threads::topology::get_socket_affinity_mask
 (C++ member), 750
 hpx::threads::topology::core_offset hpx::threads::topology::get_socket_number
 (C++ member), 751
 hpx::threads::topology::cpuset_to_nodes
 (C++ function), 749
 hpx::threads::topology::deallocate (C++ function), 749
 hpx::threads::topology::empty_mask (C++ member), 751
 hpx::threads::topology::extract_node_cou
 (C++ function), 750
 hpx::threads::topology::extract_node_mask
 (C++ function), 750
 hpx::threads::topology::get_area_membind
 (C++ function), 749
 hpx::threads::topology::get_core_affinity
 (C++ function), 747
 hpx::threads::topology::get_core_number hpx::threads::topology::init_core_affinity_mask_f
 (C++ function), 748
 hpx::threads::topology::get_cpubind_mask
 (C++ function), 748
 hpx::threads::topology::get_machine_affi
 (C++ function), 746
 hpx::threads::topology::get_numa_domain hpx::threads::topology::init_numa_node_number
 (C++ function), 749
 hpx::threads::topology::get_numa_node_af
 (C++ function), 747
 hpx::threads::topology::get_numa_node_af
 (C++ function), 747
 hpx::threads::topology::get_numa_node_numb
 (C++ function), 746
 hpx::threads::topology::get_number_of_col
 (C++ function), 748
 hpx::threads::topology::get_number_of_col
 (C++ function), 748
 hpx::threads::topology::get_number_of_nu
 (C++ member), 750
 hpx::threads::topology::get_number_of_nu
 (C++ function), 748
 hpx::threads::topology::get_number_of_numa_node_pus
 (C++ function), 748
 hpx::threads::topology::get_number_of_numa_nodes
 (C++ function), 748
 hpx::threads::topology::get_number_of_socket_cores
 (C++ function), 748
 hpx::threads::topology::get_number_of_socket_pus
 (C++ function), 748
 hpx::threads::topology::get_number_of_sockets
 (C++ function), 748
 hpx::threads::topology::get_pu_number
 (C++ function), 748
 hpx::threads::topology::get_service_affinity_mask
 (C++ function), 746
 hpx::threads::topology::get_socket_affinity_mask
 (C++ function), 746
 hpx::threads::topology::init_core_affinity_mask
 (C++ function), 750
 hpx::threads::topology::init_core_affinity_mask_f
 (C++ function), 749
 hpx::threads::topology::init_core_number
 (C++ function), 750
 hpx::threads::topology::init_machine_affinity_mask
 (C++ function), 750
 hpx::threads::topology::init_node_number
 (C++ function), 750
 hpx::threads::topology::init_num_of_pus
 (C++ function), 750
 hpx::threads::topology::init numa_node_affinity_ma
 (C++ function), 750
 hpx::threads::topology::init numa_node_affinity_ma
 (C++ function), 749
 hpx::threads::topology::init numa_node_number
 (C++ function), 750
 hpx::threads::topology::init_socket_affinity_ma
 (C++ function), 750
 hpx::threads::topology::init_socket_affinity_ma
 (C++ function), 749
 hpx::threads::topology::init_socket_number
 (C++ function), 750
 hpx::threads::topology::init_thread_affinity_ma
 (C++ function), 749
 hpx::threads::topology::machine_affinity_ma
 (C++ member), 750
 hpx::threads::topology::mask_to_bitmap

(*C++ function*), 749
hpx::threads::topology::memory_page_sizehpx::traits::is_executor_parameters_v
(*C++ member*), 751
hpx::threads::topology::mutex_type (*C++ type*), 750
hpx::threads::topology::num_of_pus_ (*C++ member*), 750
hpx::threads::topology::numa_node_affinihpx::mask::initialized_value (*C++ enumerator*), 607
(*C++ member*), 750
hpx::threads::topology::numa_node_numberhpx::unknown_component_address (*C++ enumerator*), 607
(*C++ member*), 750
hpx::threads::topology::print_affinity_maphpx::unwrap (*C++ function*), 666
(*C++ function*), 748
hpx::threads::topology::print_hwloc
(*C++ function*), 749
hpx::threads::topology::print_mask_vectohpx::unwrap_all (*C++ function*), 667
(*C++ function*), 749
hpx::threads::topology::print_vector
(*C++ function*), 749
hpx::threads::topology::pu_offset (*C++ member*), 751
hpx::threads::topology::reduce_thread_prhpx::util::priority (*C++ type*), 555, 578, 584, 588, 590–592,
(*C++ function*), 748
hpx::threads::topology::set_area_membind_nodeset::bind_nodes (*C++ function*), 724, 751, 752, 796, 804
(*C++ function*), 749
hpx::threads::topology::set_thread_affinhpkmkl::accept_end (*C++ function*), 555
(*C++ function*), 747
hpx::threads::topology::socket_affinity_masks_ition), 717
(*C++ member*), 750
hpx::threads::topology::socket_numbers_ (*C++ member*), 750
hpx::threads::topology::thread_affinity_maphpx::util::cache (*C++ type*), 578, 584, 588, 590–
(*C++ member*), 750
592, 594–596
hpx::threads::topology::thread_affinity_masksutil::cache::entries (*C++ type*), 588,
(*C++ member*), 750
hpx::threads::topology::topo (*C++ member*), 750
hpx::threads::topology::topo_mtx (*C++ member*), 750
hpx::threads::topology::topology (*C++ function*), 746
hpx::threads::topology::use_pus_as_coreshpx::util::cache::entries::entry::get_size
(*C++ member*), 750
hpx::threads::topology::write_to_log
(*C++ function*), 749
hpx::threads::unknown (*C++ enumerator*), 599, 600
hpx::threads::yes (*C++ enumerator*), 603
hpx::throwmode (*C++ enum*), 618
hpx::throws (*C++ member*), 618
hpx::tolerate_node_faults (*C++ function*), 696
hpx::traits (*C++ type*), 641, 759
hpx::traits::action_remote_result_t
(*C++ type*), 759
hpx::traits::is_executor_parameters_t
(*C++ type*), 641
hpx::traits::is_executor_parameters_v
(*C++ member*), 641
hpx::transform_reduce (*C++ function*), 516, 518
hpx::trigger_lco_event (*C++ function*), 791
hpx::unhandled_exception (*C++ enumerator*), 608
hpx::mask::initialized_value (*C++ enumerator*), 607
hpx::unknown_component_address (*C++ enumerator*), 607
hpx::unwrap (*C++ function*), 666
hpx::unwrap_all (*C++ function*), 667
hpx::unwrap_n (*C++ function*), 667
hpx::unwrapping (*C++ function*), 667
hpx::unwrapping_all (*C++ function*), 668
hpx::unwrapping_n (*C++ function*), 668
hpx::util::accept_begin (*C++ function*), 555
hpx::util::accept_end (*C++ function*), 555
hpx::util::annotated_function (*C++ function*), 555
hpx::util::as_string (*C++ function*), 724
hpx::util::cache (*C++ type*), 578, 584, 588, 590–
592, 594–596
hpx::util::cache::entries::entry::get_size
(*C++ function*), 589
hpx::util::cache::entries::entry::insert
(*C++ function*), 589
hpx::util::cache::entries::entry::remove
(*C++ function*), 589
hpx::util::cache::entries::entry::touch
(*C++ function*), 589
hpx::util::cache::entries::entry::value_ (*C++ member*), 590
hpx::util::cache::entries::entry::value_type
(*C++ type*), 589
hpx::util::cache::entries::fifo_entry
(*C++ class*), 590
hpx::util::cache::entries::fifo_entry::fifo_entry

(C++ function), 590
hpx::util::cache::entries::fifo_entry::g~~hp~~_xcreation_{cache}::local_cache::clear
(C++ function), 591
hpx::util::cache::entries::fifo_entry::i~~hp~~_prtutil::cache::local_cache::current_size_
(C++ function), 590
hpx::util::cache::entries::fifo_entry::i~~hp~~_prt_{tm}l_{time}cache::local_cache::entry_heap_
(C++ member), 591
hpx::util::cache::entries::fifo_entry<Val~~hp~~_x:_{base}:_{type}he>::local_cache::erase
(C++ type), 591
hpx::util::cache::entries::fifo_entry<Val~~hp~~_x:_{time}:_{path}>::local_cache::free_space
(C++ type), 591
hpx::util::cache::entries::lfu_entry hpx::util::cache::local_cache::get_entry
(C++ class), 591
hpx::util::cache::entries::lfu_entry::ge~~hp~~_pcces\$_l_{cou}the::local_cache::get_statistics
(C++ function), 592
hpx::util::cache::entries::lfu_entry::lf~~hp~~_pnt_{util}::cache::local_cache::holds_key
(C++ function), 592
hpx::util::cache::entries::lfu_entry::ref~~hp~~_pcount_{il}::cache::local_cache::insert
(C++ member), 592
hpx::util::cache::entries::lfu_entry::to~~hp~~_x::util::cache::local_cache::insert_policy_<
(C++ function), 592
hpx::util::cache::entries::lfu_entry<Val~~hp~~_x:_{base}:_{type}he>::local_cache::local_cache
(C++ type), 592
hpx::util::cache::entries::lru_entry hpx::util::cache::local_cache::max_size_
(C++ class), 593
hpx::util::cache::entries::lru_entry::ach~~hp~~_x:_{time}::cache::local_cache::reserve
(C++ member), 594
hpx::util::cache::entries::lru_entry::ge~~hp~~_pcces\$_l_t_{ime}ache::local_cache::size
(C++ function), 593
hpx::util::cache::entries::lru_entry::lr~~hp~~_pnt_{util}::cache::local_cache::statistics_<
(C++ function), 593
hpx::util::cache::entries::lru_entry::to~~hp~~_x::util::cache::local_cache::store_<
(C++ function), 593
hpx::util::cache::entries::lru_entry<Val~~hp~~_x:_{base}:_{type}he>::local_cache::update
(C++ type), 593
hpx::util::cache::entries::lru_entry<Val~~hp~~_x:_{time}:_{poa}he>::local_cache::update_if
(C++ type), 593
hpx::util::cache::entries::size_entry hpx::util::cache::local_cache::update_policy_<
(C++ class), 594
hpx::util::cache::entries::size_entry::b~~hp~~_x:_{type}el::cache::local_cache<Key,
(C++ type), 595
hpx::util::cache::entries::size_entry::derived_{_}Entry, UpdatePolicy,
(C++ type), 595
hpx::util::cache::entries::size_entry::get_size_{function}, 584
(C++ function), 594
hpx::util::cache::entries::size_entry::size_ Entry, UpdatePolicy,
(C++ member), 595
hpx::util::cache::entries::size_entry::size_ InsertPolicy, CacheStorage,
hpx::util::cache::entries::size_entry::size_ Statistics>::adapt::adapt (C++ mem-
(C++ function), 594
ber), 584
hpx::util::cache::local_cache (C++ class), hpx::util::cache::local_cache<Key,
578
hpx::util::cache::local_cache::adapt Entry, UpdatePolicy,
(C++ struct), 583
hpx::util::cache::local_cache::capacity InsertPolicy, CacheStorage,
Statistics>::adapt::operator()<
(C++ function), 584

```
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::adapted_update_policy_type
    (C++ type), 583
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::const_iterator (C++ type),
    583
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::entry_type (C++ type),
    578
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::heap_iterator (C++ type),
    583
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::heap_type (C++ type),
    583
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::insert_policy_type
    (C++ type), 578
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::iterator (C++ type),
    583
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::key_type (C++ type),
    578
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::size_type (C++ type),
    579
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::statistics_type
    (C++ type), 579
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::storage_type (C++ type),
    579
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::adapted_update_policy_type
    (C++ type), 579
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::storage_value_type
    (C++ type), 579
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::update_on_exit (C++ type),
    583
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::update_policy_type
    (C++ type), 578
hpx::util::cache::local_cache<Key,
    Entry, UpdatePolicy,
    InsertPolicy, CacheStorage,
    Statistics>::value_type (C++ type),
    579
hpx::util::cache::lru_cache (C++ class),
    584
hpx::util::cache::lru_cache::capacity
    (C++ function), 585
hpx::util::cache::lru_cache::clear (C++ function),
    587
hpx::util::cache::lru_cache::current_size_
    (C++ member), 588
hpx::util::cache::lru_cache::erase (C++ function),
    587
hpx::util::cache::lru_cache::evict (C++ function),
    588
hpx::util::cache::lru_cache::get_entry
    (C++ function), 585, 586
hpx::util::cache::lru_cache::get_statistics
    (C++ function), 587
hpx::util::cache::lru_cache::holds_key
    (C++ function), 585
hpx::util::cache::lru_cache::insert
    (C++ function), 586
hpx::util::cache::lru_cache::insert_nonexist
    (C++ function), 588
hpx::util::cache::lru_cache::lru_cache
    (C++ function), 585
hpx::util::cache::lru_cache::map_
    (C++ member), 588
hpx::util::cache::lru_cache::max_size_
    (C++ member), 588
hpx::util::cache::lru_cache::reserve
    (C++ function), 585
hpx::util::cache::lru_cache::size (C++ function),
    585
hpx::util::cache::lru_cache::statistics_
```


(C++ function), 597
hpx::util::cache::statistics::no_statist
 (C++ function), 597
hpx::util::cache::statistics::no_statist
 (C++ struct), 598
hpx::util::cache::statistics::no_statist
 (C++ function), 598
hpx::util::cache::statistics::update_en
 (C++ enumerator), 597
hpx::util::checkpoint (C++ class), 802
hpx::util::checkpoint::~checkpoint (C++
 function), 802
hpx::util::checkpoint::begin (C++ func
 tion), 802
hpx::util::checkpoint::checkpoint (C++
 function), 802
hpx::util::checkpoint::const_iterator
 (C++ type), 802
hpx::util::checkpoint::data (C++ function),
 802
hpx::util::checkpoint::data_ (C++ mem
 ber), 803
hpx::util::checkpoint::end (C++ function),
 802
hpx::util::checkpoint::operator= (C++
 function), 802
hpx::util::checkpoint::serialize (C++
 function), 803
hpx::util::checkpoint::size (C++ function),
 802
hpx::util::cleanup_ip_address (C++ func
 tion), 555
hpx::util::connect_begin (C++ function), 555
hpx::util::connect_end (C++ function), 555
hpx::util::data_type (C++ enum), 725
hpx::util::data_type_address (C++ enum
 erator), 725
hpx::util::data_type_description (C++
 enumerator), 725
hpx::util::endpoint_iterator_type (C++
 type), 555
hpx::util::functional (C++ type), 654, 669
hpx::util::functional::invoke (C++
 struct), 654
hpx::util::functional::invoke::operator()
 (C++ function), 654
hpx::util::functional::invoke_r (C++
 struct), 654
hpx::util::functional::invoke_r::operator()
 (C++ function), 654
hpx::util::get_endpoint (C++ function), 555
hpx::util::get_endpoint_name (C++ func
 tion), 555
hpx::util::insert_checked (C++ function),
 751
hpx::util::invoke (C++ function), 654
hpx::util::invoke_fused (C++ function), 655
hpx::util::update_invoke_ifused_r (C++
 function), 655
hpx::util::update_invoke_exit (C++ function), 654
hpx::util::io_service_pool (C++ class), 656
hpx::util::io_service_pool::~io_service_pool
 (C++ function), 656
hpx::util::io_service_pool::clear (C++
 function), 656
hpx::util::io_service_pool::clear_locked
 (C++ function), 657
hpx::util::io_service_pool::continue_barrier_
 (C++ member), 658
hpx::util::io_service_pool::get_io_service
 (C++ function), 657
hpx::util::io_service_pool::get_name
 (C++ function), 657
hpx::util::io_service_pool::get_os_thread_handle
 (C++ function), 657
hpx::util::io_service_pool::HPX_NON_COPYABLE
 (C++ function), 656
hpx::util::io_service_pool::init (C++
 function), 657
hpx::util::io_service_pool::initialize_work
 (C++ function), 657
hpx::util::io_service_pool::io_service_pool
 (C++ function), 656
hpx::util::io_service_pool::io_service_ptr
 (C++ type), 657
hpx::util::io_service_pool::io_services_
 (C++ member), 657
hpx::util::io_service_pool::join (C++
 function), 656
hpx::util::io_service_pool::join_locked
 (C++ function), 657
hpx::util::io_service_pool::mtx_ (C++
 member), 657
hpx::util::io_service_pool::next_io_service_
 (C++ member), 657
hpx::util::io_service_pool::notifier_
 (C++ member), 658
hpx::util::io_service_pool::pool_name_
 (C++ member), 658
hpx::util::io_service_pool::pool_name_postfix_
 (C++ member), 658
hpx::util::io_service_pool::pool_size_
 (C++ member), 657
hpx::util::io_service_pool::run (C++
 function), 656
hpx::util::io_service_pool::run_locked
 (C++ function), 657
hpx::util::io_service_pool::size (C++

hpx::util::io_service_pool::stop (C++ function), 656
 hpx::util::io_service_pool::stop_locked (C++ function), 657
 hpx::util::io_service_pool::stopped (C++ function), 656
 hpx::util::io_service_pool::stopped_ (C++ member), 657
 hpx::util::io_service_pool::thread_run (C++ function), 657
 hpx::util::io_service_pool::threads_ (C++ member), 657
 hpx::util::io_service_pool::wait (C++ function), 656
 hpx::util::io_service_pool::wait_barrier_ (C++ member), 658
 hpx::util::io_service_pool::wait_locked (C++ function), 657
 hpx::util::io_service_pool::waiting_ (C++ member), 658
 hpx::util::io_service_pool::work_ (C++ member), 657
 hpx::util::io_service_pool::work_type (C++ type), 657
 hpx::util::operator>> (C++ function), 797
 hpx::util::operator<< (C++ function), 724, 797
 hpx::util::parse_sed_expression (C++ function), 752
 hpx::util::prepare_checkpoint (C++ function), 800, 801
 hpx::util::prepare_checkpoint_data (C++ function), 804
 hpx::util::resolve_hostname (C++ function), 555
 hpx::util::resolve_public_ip_address (C++ function), 555
 hpx::util::restore_checkpoint (C++ function), 801
 hpx::util::restore_checkpoint_data (C++ function), 805
 hpx::util::retrieve_commandline_arguments (C++ function), 695
 hpx::util::save_checkpoint (C++ function), 797–799
 hpx::util::save_checkpoint_data (C++ function), 804
 hpx::util::sed_transform (C++ struct), 752
 hpx::util::sed_transform::command_ (C++ member), 752
 hpx::util::sed_transform::operator bool (C++ function), 752
 hpx::util::sed_transform::operator! (C++ function), 752
 hpx::util::sed_transform::operator() (C++ function), 752
 hpx::util::sed_transform::sed_transform (C++ function), 752
 hpx::util::split_ip_address (C++ function), 555
 hpx::util::thread_description (C++ struct), 724
 hpx::util::thread_description::get_address (C++ function), 725
 hpx::util::thread_description::get_description (C++ function), 725
 hpx::util::thread_description::init_from_alternative (C++ function), 725
 hpx::util::thread_description::kind (C++ function), 725
 hpx::util::thread_description::operator bool (C++ function), 725
 hpx::util::thread_description::thread_description (C++ function), 725
 hpx::util::thread_description::valid (C++ function), 725
 hpx::util::traverse_pack_async (C++ function), 665
 hpx::util::traverse_pack_async_allocator (C++ function), 665
 hpx::util::unwrap (C++ function), 668
 hpx::util::unwrap_all (C++ function), 668
 hpx::util::unwrap_n (C++ function), 668
 hpx::util::unwrapping (C++ function), 668
 hpx::util::unwrapping_all (C++ function), 669
 hpx::util::unwrapping_n (C++ function), 669
 hpx::version_too_new (C++ enumerator), 607
 hpx::version_too_old (C++ enumerator), 607
 hpx::version_unknown (C++ enumerator), 607
 hpx::wait_all (C++ function), 560
 hpx::wait_all_n (C++ function), 561
 hpx::wait_any (C++ function), 561, 562
 hpx::wait_any_n (C++ function), 562
 hpx::wait_each (C++ function), 563, 564
 hpx::wait_each_n (C++ function), 564
 hpx::wait_some (C++ function), 565, 566
 hpx::wait_some_n (C++ function), 566
 hpx::when_all (C++ function), 567
 hpx::when_all_n (C++ function), 568
 hpx::when_any (C++ function), 569
 hpx::when_any_n (C++ function), 570
 hpx::when_any_result (C++ struct), 570
 hpx::when_any_result::futures (C++ member), 570
 hpx::when_any_result::index (C++ member), 570
 hpx::when_each (C++ function), 571

hpx::when_each_n (*C++ function*), 572
hpx::when_some (*C++ function*), 573, 574
hpx::when_some_n (*C++ function*), 574
hpx::when_some_result (*C++ struct*), 575
hpx::when_some_result::futures (*C++ member*), 575
hpx::when_some_result::indices (*C++ member*), 575
hpx::worker (*C++ enumerator*), 678
hpx::yield_aborted (*C++ enumerator*), 608
HPX_ASSERT (*C macro*), 556
HPX_ASSERT_MSG (*C macro*), 556
HPX_CACHE_METHOD_UNSCOPED_ENUM_DEPRECATED_MSG *macro*), 782
(*C macro*), 596
HPX_COROUTINES_WITH_SWAP_CONTEXT_EMULATION:BOOL *macro*, 782
command line option, 49
HPX_COUNTER_TYPE_UNSCOPED_ENUM_DEPRECATED_MSG *macro*), 782
(*C macro*), 882
HPX_DATASTRUCTURES_WITH_ADAPT_STD_TUPLE:BOOL *macro*, 782
command line option, 53
HPX_DECLARE_PLAIN_ACTION (*C macro*), 757
HPX_DEFINE_COMPONENT_ACTION (*C macro*), 756
HPX_DEFINE_COMPONENT_COMMANDLINE_OPTIONS
(*C macro*), 839
HPX_DEFINE_COMPONENT_NAME (*C macro*), 841
HPX_DEFINE_COMPONENT_NAME_ (*C macro*), 841
HPX_DEFINE_COMPONENT_NAME_2 (*C macro*), 841
HPX_DEFINE_COMPONENT_NAME_3 (*C macro*), 841
HPX_DEFINE_COMPONENT_STARTUP_SHUTDOWN
(*C macro*), 840
HPX_DEFINE_GET_COMPONENT_TYPE (*C macro*),
841
HPX_DEFINE_GET_COMPONENT_TYPE_STATIC (*C
macro*), 841
HPX_DEFINE_GET_COMPONENT_TYPE_TEMPLATE
(*C macro*), 841
HPX_DEFINE_PLAIN_ACTION (*C macro*), 757
HPX_DP_LAZY (*C macro*), 606
HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY *macro*), 782
command line option, 53
HPX_INVOKE_R (*C macro*), 653
HPX_ITERATOR_SUPPORT_WITH_BOOST_ITERATOR ~~HPX_ITERATOR_SUPPORT_BASESEMBARIOITIHTVABUEI~~ ID_6
command line option, 53
HPX_PERFORMANCE_COUNTER_V1 (*C macro*), 882
HPX_PLAIN_ACTION (*C macro*), 757
HPX_PLAIN_ACTION_ID (*C macro*), 758
HPX_PP_CAT (*C macro*), 669
HPX_PP_EXPAND (*C macro*), 669
HPX_PP_NARGS (*C macro*), 670
HPX_PP_STRINGIZE (*C macro*), 670
HPX_PP_STRIP_PARENS (*C macro*), 671
HPX_REGISTER_ACTION (*C macro*), 754
HPX_REGISTER_ACTION_DECLARATION
macro), 754
HPX_REGISTER_ACTION_DECLARATION_ (*C
macro*), 754
HPX_REGISTER_ACTION_DECLARATION_1
macro), 754
HPX_REGISTER_ACTION_ID (*C macro*), 755
HPX_REGISTER_BASE_LCO_WITH_VALUE
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_ (*C
macro*), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_1
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_2
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_3
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION2
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_ (*C
macro*), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_1
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_2
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_3
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_DECLARATION_4
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID (*C
macro*), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID2 (*C
macro*), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_ (*C
macro*), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_4
macro), 782
HPX_REGISTER_BASE_LCO_WITH_VALUE_ID_5
macro), 782
HPX_REGISTER_BINARY_FILTER_FACTORY (*C
macro*), 895
HPX_REGISTER_COMMANDLINE_MODULE
macro), 839
HPX_REGISTER_COMMANDLINE_MODULE_DYNAMIC
macro), 839
HPX_REGISTER_COMMANDLINE_OPTIONS
macro), 675
HPX_REGISTER_COMMANDLINE_OPTIONS_DYNAMIC
macro), 675
HPX_REGISTER_COMMANDLINE_REGISTRY
macro), 839

macro), 675

HPX_REGISTER_COMMANDLINE_REGISTRY_DYNAMIC (C macro), 676
(C macro), 675

HPX_REGISTER_COMPONENT (C macro), 897

HPX_REGISTER_COMPONENT_REGISTRY (C macro), 676

HPX_REGISTER_COMPONENT_REGISTRY_DYNAMIC (C macro), 676

HPX_REGISTER_DERIVED_COMPONENT_FACTORY (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_3 (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_4 (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_D (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DM (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DMREG (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DMREGASTER (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DMREGASTER_DM (C macro), 901

HPX_REGISTER_DERIVED_COMPONENT_FACTORY_DMREGASTER_DMREG (C macro), 901

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY (C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_SERIALIZATION_WITH_ALL_TYPES_ARE_BITWISE_SERIALIZATION (C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_SERIALIZATION_WITH_ALLOW_CONST_TUPLE_MEMBERS_BOOL (C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_SERIALIZATION_WITH_ALLOW_RAW_POINTER_SERIALIZATION (C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_SERIALIZATION_WITH_BOOST_TYPES_BOOL (C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_SERIALIZATION_WITH_SUPPORTS_ENDIANESS_BOOL (C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_THROWROW_EXCEPTION (C macro), 619
(C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_DYNAMIC (C macro), 618
(C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_THROWS_IF (C macro), 620
(C macro), 898

HPX_REGISTER_MINIMAL_COMPONENT_REGISTRY_HPX_UNSCOPED_ENUM_DEPRECATED_MSG (C macro), 620
(C macro), 898

HPX_REGISTER_PLUGIN_BASE_REGISTRY (C macro), 677
(C macro), 896

HPX_REGISTER_PLUGIN_REGISTRY (C macro), 896

HPX_REGISTER_PLUGIN_REGISTRY_ (C macro), 896

HPX_REGISTER_PLUGIN_REGISTRY_2 (C macro), 896

HPX_REGISTER_PLUGIN_REGISTRY_4 (C macro), 896

HPX_REGISTER_PLUGIN_REGISTRY_5 (C macro), 896

HPX_REGISTER_PLUGIN_REGISTRY_MODULE (C macro), 677

HPX_REGISTER_PLUGIN_REGISTRY_MODULE_DYNAMIC command line option, 45

HPX_WITH_BUILD_BINARY_PACKAGE:BOOL
 command line option, 45

HPX_WITH_CHECK_MODULE_DEPENDENCIES:BOOL
 command line option, 45

HPX_WITH_COMPILE_ONLY_TESTS:BOOL
 command line option, 47

HPX_WITH_COMPILER_WARNINGS:BOOL
 command line option, 45

HPX_WITH_COMPILER_WARNINGS_AS_ERRORS:BOOL
 command line option, 45

HPX_WITH_COMPRESSION_BZIP2:BOOL
 command line option, 45

HPX_WITH_COMPRESSION_SNAPPY:BOOL
 command line option, 45

HPX_WITH_COMPRESSION_ZLIB:BOOL
 command line option, 45

HPX_WITH_COROUTINE_COUNTERS:BOOL
 command line option, 49

HPX_WITH_CUDA
 command line option, 37

HPX_WITH_CUDA:BOOL
 command line option, 45

HPX_WITH_CXX_STANDARD
 command line option, 37

HPX_WITH_CXX_STANDARD:STRING
 command line option, 45

HPX_WITH_DATAPAR_VC:BOOL
 command line option, 45

HPX_WITH_DEPRECATED_WARNINGS:BOOL
 command line option, 45

HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL
 command line option, 45

HPX_WITH_DISTRIBUTED_RUNTIME:BOOL
 command line option, 47

HPX_WITH_DOCUMENTATION:BOOL
 command line option, 47

HPX_WITH_DOCUMENTATION_OUTPUT_FORMATS:STRING
 command line option, 47

HPX_WITH_DYNAMIC_HPX_MAIN:BOOL
 command line option, 45

HPX_WITH_EXAMPLES
 command line option, 37

HPX_WITH_EXAMPLES:BOOL
 command line option, 47

HPX_WITH_EXAMPLES_HDF5:BOOL
 command line option, 48

HPX_WITH_EXAMPLES_OPENMP:BOOL
 command line option, 48

HPX_WITH_EXAMPLES_QT4:BOOL
 command line option, 48

HPX_WITH_EXAMPLES_QTHREADS:BOOL
 command line option, 48

HPX_WITH_EXAMPLES_TBB:BOOL
 command line option, 48

HPX_WITH_EXECUTABLE_PREFIX:STRING
 command line option, 48

HPX_WITH_FAIL_COMPILE_TESTS:BOOL
 command line option, 48

HPX_WITHFAULT_TOLERANCE:BOOL
 command line option, 45

HPX_WITH_FETCH_ASIO:BOOL
 command line option, 48

HPX_WITH_FETCH_LCI:BOOL
 command line option, 48

HPX_WITH_FULL_RPATH:BOOL
 command line option, 46

HPX_WITH_GCC_VERSION_CHECK:BOOL
 command line option, 46

HPX_WITH_GENERIC_CONTEXT_COROUTINES
 command line option, 37

HPX_WITH_GENERIC_CONTEXT_COROUTINES:BOOL
 command line option, 46

HPX_WITH_HIDDEN_VISIBILITY:BOOL
 command line option, 46

HPX_WITH_HIP:BOOL
 command line option, 46

HPX_WITH_IO_COUNTERS:BOOL
 command line option, 48

HPX_WITH_IO_POOL:BOOL
 command line option, 49

HPX_WITH_ITTNOTIFY:BOOL
 command line option, 51

HPX_WITH_LCI_TAG:STRING
 command line option, 48

HPX_WITHLOGGING:BOOL
 command line option, 46

HPX_WITH_MALLOC
 command line option, 37

HPX_WITH_MALLOC:STRING
 command line option, 46

HPX_WITH_MAX_CPU_COUNT:STRING
 command line option, 37

HPX_WITH_MAX_NUMA_DOMAIN_COUNT:STRING
 command line option, 49

HPX_WITH_MODULES_AS_STATIC_LIBRARIES:BOOL
 command line option, 46

HPX_WITH_NETWORKING:BOOL
 command line option, 51

HPX_WITH_NICE_THREADLEVEL:BOOL
 command line option, 46

HPX_WITH_PAPI:BOOL
 command line option, 51

HPX_WITH_PARALLEL_TESTS_BIND_NONE:BOOL
 command line option, 52

HPX_WITH_PARCEL_COALESCING:BOOL
 command line option, 46

HPX_WITH_PARCEL_PROFILING:BOOL
 command line option, 51

HPX_WITH_PARCELPORT_ACTION_COUNTERS:BOOLHPX_WITH_TESTS_HEADERS:BOOL
 command line option, 51
 command line option, 48

HPX_WITH_PARCELPORT_COUNTERS:BOOL
 command line option, 51

HPX_WITH_PARCELPORT_LCI:BOOL
 command line option, 51

HPX_WITH_PARCELPORT_LIBFABRIC:BOOL
 command line option, 51

HPX_WITH_PARCELPORT_MPI:BOOL
 command line option, 51

HPX_WITH_PARCELPORT_TCP
 command line option, 37

HPX_WITH_PARCELPORT_TCP:BOOL
 command line option, 51

HPX_WITH_PKGCONFIG:BOOL
 command line option, 46

HPX_WITH_PRECOMPILED_HEADERS:BOOL
 command line option, 46

HPX_WITH_RUN_MAIN_EVERYWHERE:BOOL
 command line option, 46

HPX_WITH_SANITIZERS:BOOL
 command line option, 52

HPX_WITH_SCHEDULER_LOCAL_STORAGE:BOOL
 command line option, 49

HPX_WITH_SPINLOCK_DEADLOCK_DETECTION:BOOLHPX_WITH_THREAD_MANAGER_IDLE_BACKOFF:BOOL
 command line option, 49
 command line option, 50

HPX_WITH_SPINLOCK_POOL_NUM:STRING
 command line option, 49

HPX_WITH_STACKOVERFLOW_DETECTION:BOOL
 command line option, 46

HPX_WITH_STACKTRACES:BOOL
 command line option, 50

HPX_WITH_STACKTRACES_DEMANGLE_SYMBOLS:BOOLHPX_WITH_THREAD_QUEUE_WAITTIME:BOOL
 command line option, 50
 command line option, 50

HPX_WITH_STACKTRACES_STATIC_SYMBOLS:BOOLHPX_WITH_TIMER_POOL:BOOL
 command line option, 50
 command line option, 50

HPX_WITH_STATIC_LINKING:BOOL
 command line option, 46

HPX_WITH_TESTS
 command line option, 37

HPX_WITH_TESTS:BOOL
 command line option, 48

HPX_WITH_TESTS_BENCHMARKS:BOOL
 command line option, 48

HPX_WITH_TESTS_DEBUG_LOG:BOOL
 command line option, 52

HPX_WITH_TESTS_DEBUG_LOG_DESTINATION:STRINGHPX_WITH_VIM_YCM:BOOL
 command line option, 52
 command line option, 46

HPX_WITH_TESTS_EXAMPLES:BOOL
 command line option, 48

HPX_WITH_TESTS_EXTERNAL_BUILD:BOOL
 command line option, 48

HPX_WITH_TESTS_HEADERS:BOOL
 command line option, 48

HPX_WITH_TESTS_MAX_THREADS_PER_LOCALITY:STRING
 command line option, 52

HPX_WITH_TESTS_REGRESSIONS:BOOL
 command line option, 48

HPX_WITH_TESTS_UNIT:BOOL
 command line option, 48

HPX_WITH_THREAD_BACKTRACE_DEPTH:STRING
 command line option, 50

HPX_WITH_THREAD_BACKTRACE_ON_SUSPENSION:BOOL
 command line option, 50

HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES:BOOL
 command line option, 50

HPX_WITH_THREAD_CUMULATIVE_COUNTS:BOOL
 command line option, 50

HPX_WITH_THREAD_DEBUG_INFO:BOOL
 command line option, 52

HPX_WITH_THREAD_DESCRIPTION_FULL:BOOL
 command line option, 52

HPX_WITH_THREAD_GUARD_PAGE:BOOL
 command line option, 52

HPX_WITH_THREAD_IDLE_RATES:BOOL
 command line option, 50

HPX_WITH_THREAD_LOCAL_STORAGE:BOOL
 command line option, 50

HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF:BOOL
 command line option, 50

HPX_WITH_THREAD_QUEUE_WAITTIME:BOOL
 command line option, 50

HPX_WITH_THREAD_STACK_MMAP:BOOL
 command line option, 50

HPX_WITH_THREAD_STEALING_COUNTS:BOOL
 command line option, 50

HPX_WITH_THREAD_TARGET_ADDRESS:BOOL
 command line option, 50

HPX_WITH_UNITY_BUILD:BOOL
 command line option, 46

HPX_WITH_VALGRIND:BOOL
 command line option, 52

HPX_WITH_VERIFY_LOCKS:BOOL
 command line option, 52

HPX_WITH_VERIFY_LOCKS_BACKTRACE:BOOL
 command line option, 52

HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD:STRING
 command line option, 46

HWLOC_ROOT:PATH
command line option, 54

L

LCO, **214**
Lightweight Control Object, **214**
Local Control Object, **214**
Locality, **213**

M

make_error_code (*C++ function*), 613

O

operator!= (*C++ function*), 604, 605, 804
operator== (*C++ function*), 604, 605, 804
operator> (*C++ function*), 604, 605
operator>= (*C++ function*), 604, 605
operator>> (*C++ function*), 803
operator< (*C++ function*), 590–592, 594, 595, 604,
605
operator<= (*C++ function*), 604, 605
operator<< (*C++ function*), 604, 605, 746, 803

P

PAPI_ROOT:PATH
command line option, 54
Parcel, **214**
Process, **214**

R

restore_checkpoint (*C++ function*), 803

S

SPHINX_ROOT:PATH
command line option, 939
std (*C++ type*), **606**, **790**
std::hash::operator () (*C++ function*), **602**, **606**
std::hash<::hpx::threads::thread_id_ref>
 (*C++ struct*), **602**, **606**
std::hash<::hpx::threads::thread_id>
 (*C++ struct*), **602**, **606**
std::uses_allocator<hpx::distributed::promise<R>,
 Allocator> (*C++ struct*), **789**, **790**

T

tag_dispatch (*C++ function*), 647
tag_fallback_invoke (*C++ function*), 577, 623
tag_invoke (*C++ function*), 558, 623, 647, 648