Parallel Programming Abstractions for Productivity, Scalability, and Performance Portability

Andreas Knüpfer, Bert Wesarg, Denis Hünich

ZIH, TU Dresden

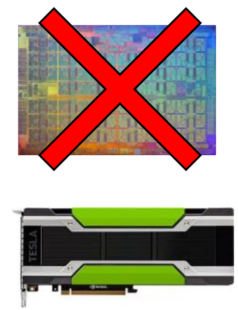
<andreas.knuepfer@tu-dresden.de>

Center for Information Services & High Performance Computing

Overview

- Introduction: The situation of HPC and HPC programming today
  - PGAS
- Programming abstraction libraries
  - Kokkos
  - Alpaka
  - DASH
  - HPX
  - Comparisons and Combinations
- Summary

The World of Parallel Computing in 2018



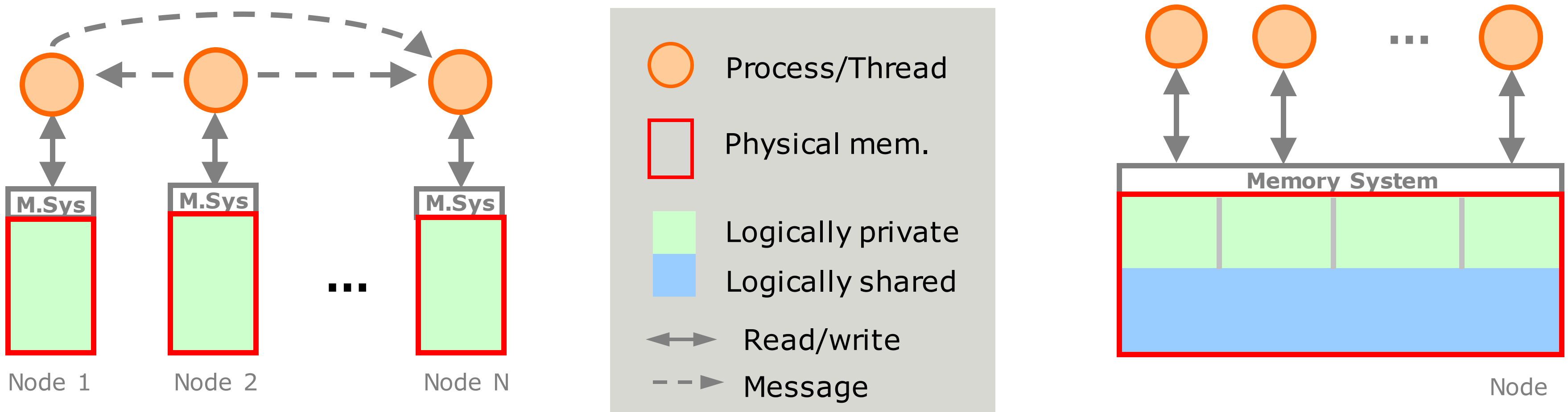
Supercomputer	Cluster	Manycore	Server	Notebook	Mobile
100 000+ cores	1000s of cores		10s of cores	2-4 cores	2-8 cores

Distributed Memory (DM) | Shared Memory (SM)

DM programming: MPI, Charm++, …

SM programming: OpenMP, Pthreads, Cilk, TBB, …

Shared Memory vs. Distributed Memory Programming



Message Passing

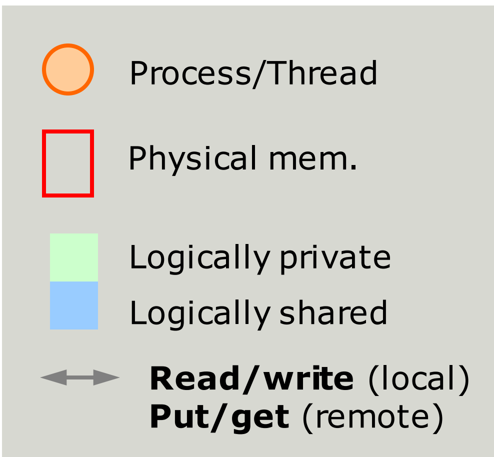
- \+ Performance, runs everywhere
- − Productivity

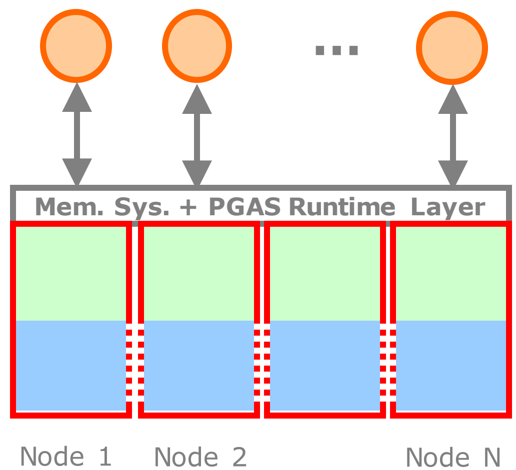
Threading

- \+ Productivity
- − Locality control, limited to SM hardware

PGAS – Combining the Advantages of both Approaches

PGAS: Partitioned Global Address Space





PGAS *Languages*

Chapel, CoArray Fortran, UPC, ...

PGAS *Libraries*

Global Arrays (GA), GASPI, OpenShmem, MPI3.0 RMA

Locality control, runs everywhere, performance and productivity

So you'd like to write parallel HPC codes in C++?

HPC programming today

- Large scale parallelism
- Heterogeneous architectures
- Hybrid parallelism --> multiple sources of complexity

- MPI+X as dominating parallel programming model
- Node-level model X strictly needed for portability and performance portability

- *What if you bet on the wrong one?*

MPI disregards C++

- Data distribution, data transfers, and synchronization deeply entangled
- The MPI C++ bindings deprecated in MPI 2.2 and removed in MPI 3.0*
- In C++ MPI codes you actually need to use MPI's C API
- C++ concepts like STL containers, iterators, and even basic data types are incompatible with MPI!

*http://blogs.cisco.com/performance/the-mpi-c-bindings-what-happened-and-why

Which alternatives are there?

... in terms of pure C++ options

- No new language or language extension
- With “native C++ look & feel”, STL compatibility

Node-local parallelism and architecture abstraction:

- Kokkos
- Alpaka

Highly parallel inter-node parallelism:

- DASH
- HPX

Node-local parallelism and architecture abstraction:

KOKKOS

Kokkos - C++ Performance Portability Programming EcoSystem

- By Sandia National Labs since 2014
- Abstraction for kernel execution and data structure management
- “No more difficult than OpenMP”
- Available math kernels: BLAS, Sparse BLAS, and Graph Kernels
- Pre-compiled library approach
    - Default backend selected at build time of library
    - Application needs to link against backend specific library
- In active development and dissemination
- Encouraged for SNL codes, actively supported by their “Kokkos Ninjas”

https://github.com/kokkos

Kokkos - C++ Performance Portability Programming EcoSystem

- Write kernel as functor with members to access needed data

```
struct Kernel
{
    void operator()( const size_t index ) const
    {
        // body
    }
    // data members
}
```

- … or use C++ lambda functions
- Concurrency and ordering of parallel iterations is not guaranteed

Kokkos - Code example

```
for (size_t i = 0; i < N; ++i) {
    // loop body
}
```

- Serial

```
#pragma omp parallel for
for (size_t i = 0; i < N; ++i) {
    // loop body
}
```

- OpenMP

```
Kokkos::parallel_for(N, [=](const size_t i) {
    // loop body
});
```

- Kokkos, also has `parallel_reduce`

Kokkos - Data structure management

- Kokkos requires to use the provided N-dimensional array datatype
- Example: 3 dimensional array with 1 fix dimension in CUDA memory

```
Kokkos::View<double**[8], LayoutLeft, CudaSpace>
    device_memory("device memory", N1, N2);
```

- Host mirror for initialization:

```
auto host_mirror =  Kokkos::create_mirror_view(device_memory);
// init
Kokkos::deep_copy(device_memory, host_mirror);
```

Kokkos - C++ Performance Portability Programming EcoSystem

- Advanced capabilities
  - Atomic operations
  - Hierarchical patterns
  - Dynamic directed acyclic graph of tasks pattern
  - Plug-in points for extensibility (e.g., debuggers and tools)

- Assessment
  - Low entry point
  - Nevertheless full fledged eco-system
  - Own N-dimensional array restrict usage outside of Kokkos

Node-local parallelism and architecture abstraction:

ALPAKA

Alpaka - C++ Node-level Programming Abstraction Library

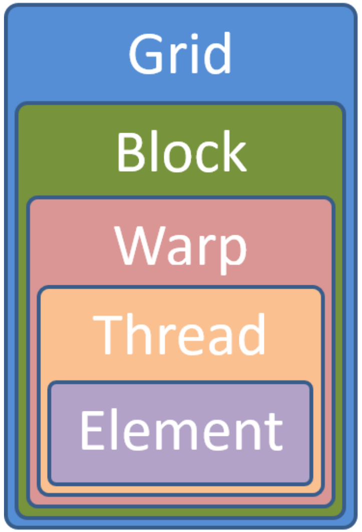
- By Helmholtz-Zentrum Dresden – Rossendorf since 2015
- Single source C++ kernels
- Abstract hierarchical redundant parallelism model
  - Grid, Block, Warp, Thread, SIMD vectorization
- Data structure agnostic memory model
- Header-only library approach
  - Backend selection at compile time of application

```
using Acc1 = alpaka::acc::AccGpuCudaRt<…>;
using Acc2 = alpaka::acc::AccCpuSerial<…>;
```

- In active development and dissemination

https://github.com/ComputationalRadiationPhysics/alpaka

HZDR
HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF



Alpaka - C++ Node-level Programming Abstraction Library

- Write kernel as functor with arguments to access needed data

```
struct Kernel {

    template<typename TAcc, …>
    ALPAKA_FN_ACC void operator()(
        TAcc const &acc, /* kernel arguments */ ) const {

        auto N = alpaka::workdiv::getWorkDiv<
            alpaka::Grid, alpaka::Thread>(acc)[0u];
        auto i = alpaka::idx::getIdx<
            alpaka::Grid, alpaka::Thread>(acc)[0u];
        // body
    }
}
```

Alpaka - Code example

```
using Host     = alpaka::acc::AccCpuSerial<…>;
using Acc      = alpaka::acc::AccGpuCudaRt<…>;
using DevAcc   = alpaka::dev::Dev<Acc>;
using PltfAcc  = alpaka::pltf::Pltf<DevAcc>;
using Stream   =
alpaka::stream::StreamCudaRtSync;

DevAcc
devAcc(alpaka::pltf::getDevByIdx<PltfAcc>(0u));

Stream stream(devAcc);

Kernel kernel;
auto task(alpaka::exec::create<Acc>(
     …, kernel /* kernel arguments */));
alpaka::stream::enqueue(stream, entry);
alpaka::wait::wait(stream);
```

- The host CPU is also an accelerator
- Abstraction is based on type system and deduction
- Instantiate (first) device
- Stream concept (like in CUDA) for asynch. execution
- Create, enqueue, wait stream task

Alpaka - Data management

- Alpaka kernels are data structure agnostic, i.e., only plain pointers
- Allocation and copying routines are provided

```
using Dim = alpaka::dim::DimInt<1>;
alpaka::vec::Vec<Dim, size_t> extends(42);

std::array<double, 42> plainBuffer;

alpaka::mem::view::ViewPlainPtr<
    DevHost, double, Dim, size_t> hostMem(
        plainBuffer.data, devHost, extends);
alpaka::mem::buf::Buf<DevAcc, double, Dim, size_t>
devMem(alpaka::mem::buf::alloc<double, size_t>(devAcc, extends));

// init plainBuffer

alpaka::mem::view::copy(stream, devMem, hostMem, extends);
```

- Declare extends of buffer
- Externally allocated buffer
- Memory view on host
- Allocate device buffer
- Streams for the copying task

Alpaka - C++ Node-level Programming Abstraction Library

- Advanced capabilities
  - Atomic operations
  - Asynchronous streams for data copying and kernel execution

- Assessment
  - Much boilerplate code needed
  - Explicit programming without any hidden defaults
  - Nevertheless full fledged abstraction system

KOKKOS VS. ALPAKA

Comparison Kokkos vs. Alpaka

- Commonalities
  - C++ single-source design
  - Simple backend selection
  - Some productivity overhead:
    May need to program more than necessary
    to fit any single backend programming model alone
  - Similar performance results for OpenMP backends

- Differences
  - Kokkos' own data management simplifies abstract programming
  - Alpaka's data agnostic management allows
    integration with other frameworks
  - Alpaka mostly faster on CUDA

Highly parallel inter-node parallelism:

DASH

DASH - Background

- By LMU Munich, HLRS Stuttgart, and TU Dresden
- DASH started in 2013 under the DFG Priority Programme 1648 "Software for Exascale Computing" (SPPEXA), currently in phase 2

https://www.dash-project.org/

dash
www.dash-project.org
Distributed Data Structures and Parallel Algorithms

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

DFG Deutsche Forschungsgemeinschaft
SPPEXA

Bundesministerium für Bildung und Forschung

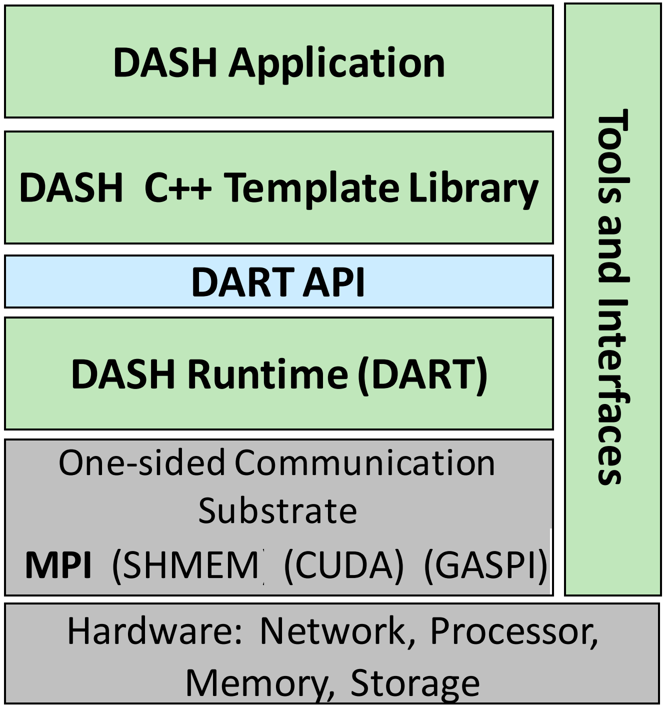
HLRS

TECHNISCHE UNIVERSITÄT DRESDEN

DASH - Overview

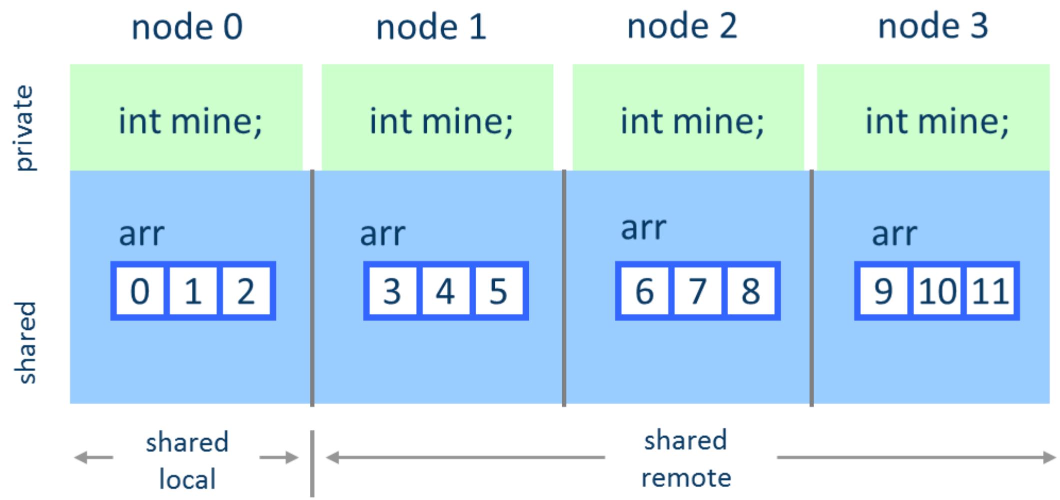
- C++ template library for application programmers
- The DASH Run Time Library (DART) for implementing DASH on top of different substrates

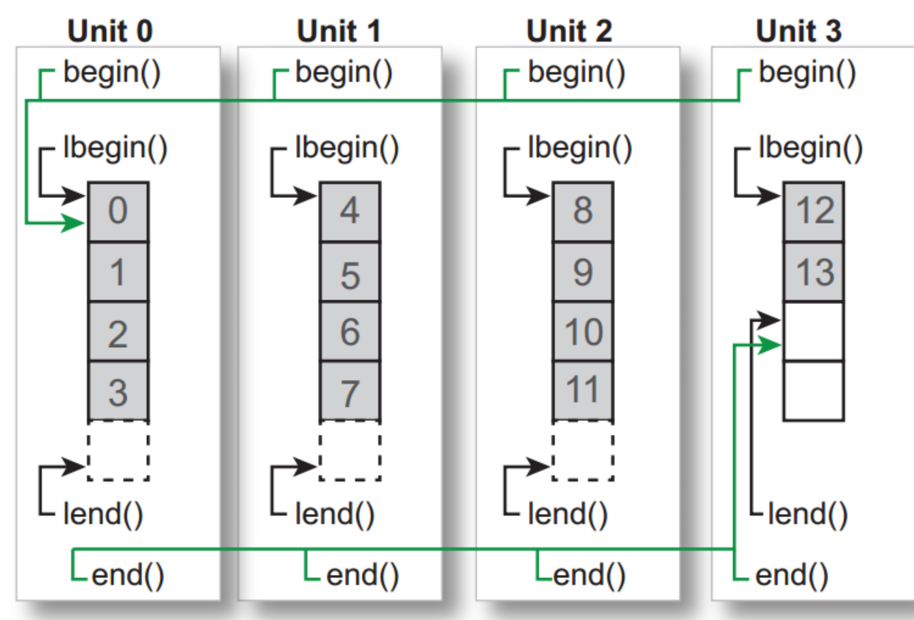
- Distributed data container classes
- Similar to the C++ STL container classes, compatible
- Built-in knowledge about distribution
- Algorithms similar to STL to work on distributed containers



DASH – Architecture

- DASH Array or n-dim. array
- Global random access with **`begin()`, `end()`** and **`[]`**
  … via slow element-wise get
- Dedicated local access with **`myarray.local.begin() / .end()`** and **`.local[]`** … direct and fast
- Configurable data distribution patterns in n dimensions
- STL-like algorithms considering actual data distribution patterns





DASH – Code Example

```
int main( int argc, char* argv[] ) {
    …
    dash::Array<int> arr(100);

    for ( auto it = arr.lbegin(); it != arr.lend(); ++it ) {
        *it= dash::myid(); // local access only
    }

    arr.barrier(); // Team barrier

    if ( 0 == dash::myid() ) { // only unit 0
        for ( const auto& el: arr ) { // global iterator
            cout<<(int)el<<" "; // remote access if not local
        } cout<<endl;
    }
    …
}
```

DASH – Further Features

Static Containers

- Array
- Matrix / NArray

Dynamic Containers

- Set / MultiSet
- List

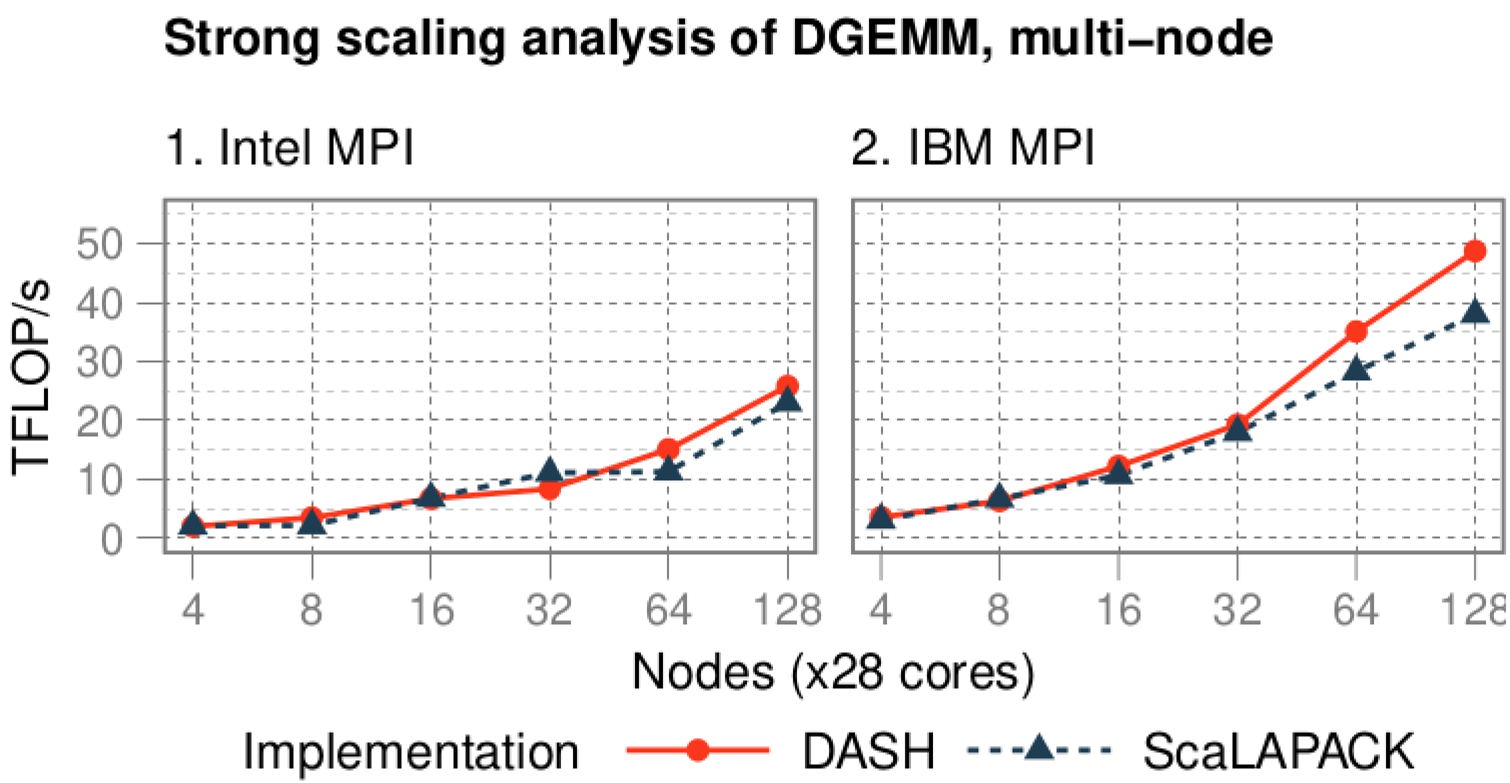
DASH Algorithms

- copy, fill, for_each, generate, reduce, transform, generate, max_element, min_element
- Halo Wrapper
- Tasking

DASH Concepts

- Dimensional
- DistributionSpec : Dimensional
- CartesianSpace = SizeSpec
- CartesianIndexSpace : CartesianSpace
- TeamSpec : CartesianSpace
- Pattern (TilePattern, BlockPattern)
- Pattern Iterator, Pattern Block Iterator
- Container
- Global Iterator / Pointer
- Global Asynchronous Reference

DASH – Performance and Scalability



Highly parallel inter-node parallelism:

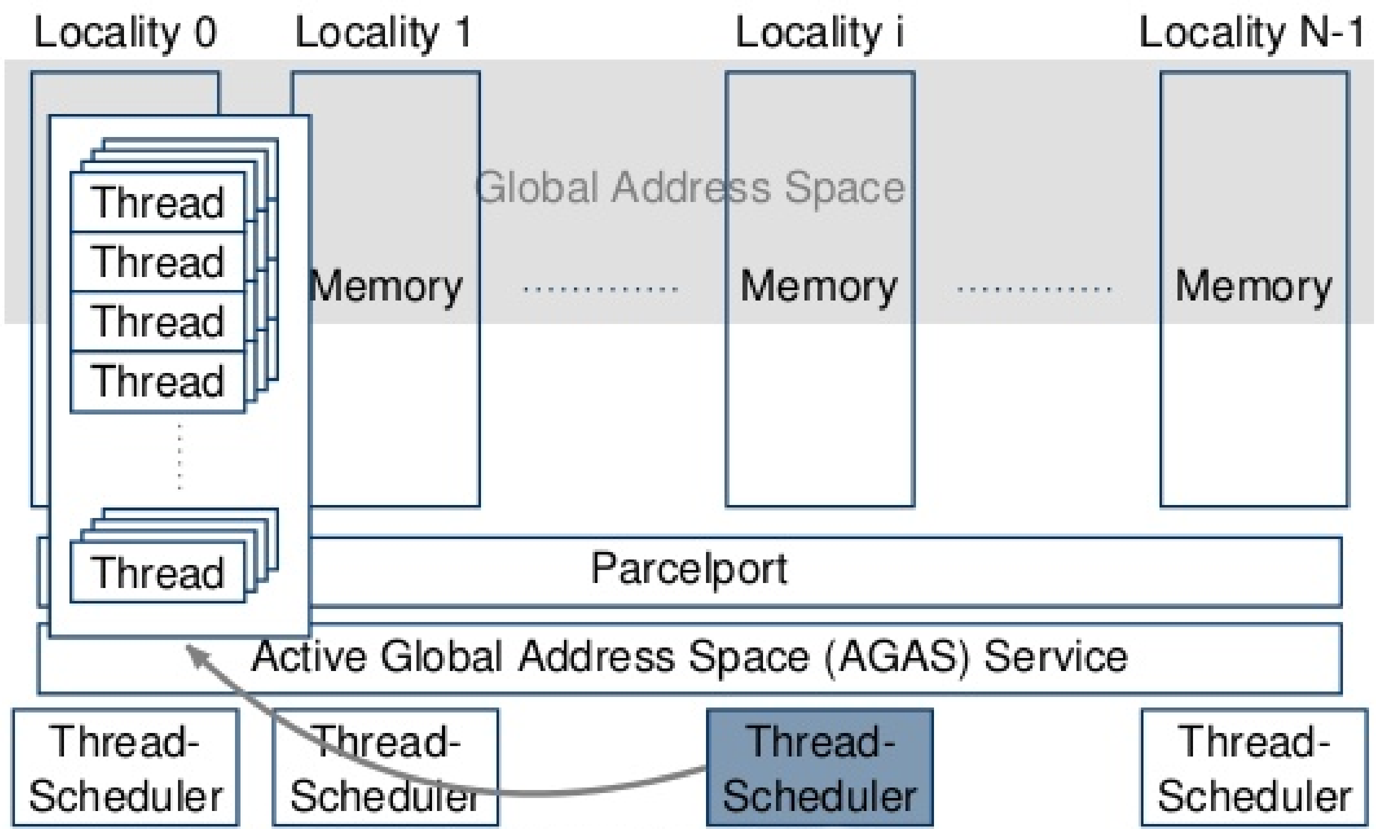
HPX

HPX - Background

- High Performance ParalleX, based on ParalleX concept
- Development began in 2007 by the ParallelX group at the Center for Computation and Technology (CCT) (Louisiana State University)
- In 2011 the STE||AR Group @ LSU (Systems Technology, Emergent Parallelism, and Algorithm Research)
- General Purpose C++ Runtime System
- ParalleX Execution Modell
  - Very light-weight tasks for latency hiding
  - Flexible task migration between compute nodes
- C++11/14 compliant API

http://stellar.cct.lsu.edu/projects/hpx/

HPX – Architecture



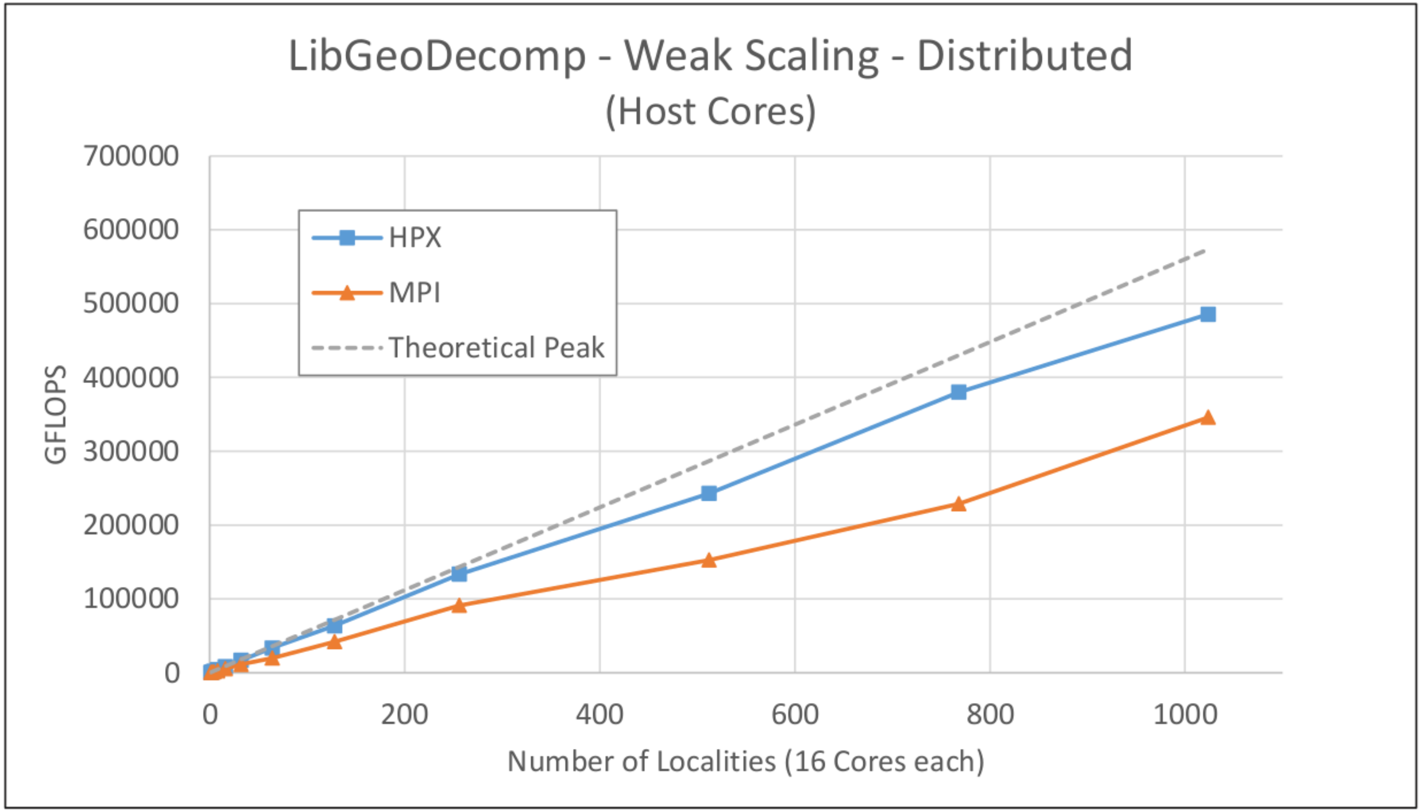
Source: C++ on its way to exascale and beyond -- The HPX Parallel Runtime System (thomas.heller@cs.fau.de) January 21, 2016

```
// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments SPMD
HPX_REGISTER_PARTITIONED_VECTOR(double);
…
// By default, number of segments equal to the current number of localities
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) {
        *lit = count;
    }
}
```

HPX – Further Features

- Parallel alternatives for C++ STL algorithm
    - Like fill, copy, sort, rotate, transform, …
    - Closely working with C++ standards committee
- Numeric parallel algorithms
    - Like reduce, transform_reduce, …
- Parallel for loops
- Execution policies for parallel algorithms and loops
- Dynamic Memory Management
- Performance counters, GPU execution policies, segmented containers

HPX – Performance and Scalability



DASH VS. HPX

Comparison HPX vs. DASH

Commonalities

- C++11/14 compliant
- Integrate well with STL

DASH

- SPMD style, simpler API, conceptually closer to conventional HPC programming
- Focus on data distribution, container classes, algorithms
- Uses backend parallelization models

HPX

- Brings aspects for both, node-level and highly scalable parallelism
- Own ParalleX concept, focus on thread/task based programming
- First steps more difficult, much boilerplate code

COMBINATIONS

Possbile Combinations

- HPX pretty much standalone
- Kokkos and Alpaka for node-level programming
  - Need multi-node framework in addition
  - MPI still the "top dog", but still hard to program

- DASH + Kokkos:
  - Competing in terms of datatype abstractions

- DASH + Alpaka
  - Orthogonal concepts
  - CUDA required data restructuring
  - --> MEPHISTO Project

SUMMARY

Summary

With Kokkos, Alpaka, DASH, and HPX there are interesting HPC programming models besides MPI+X -- worth considering for future projects

- Allow native C++ style programming with all the newer C++ 11/14/17 features
- Abstract away node-level parallel programming decision
- Increased programmer productivity, yet also some learning curve
- All presented projects are Open Source projects

They are **not**:

- … new or extended (non-standard) programming languages
- … relying on non-standard or proprietary libraries, compilers, … which might jeopardize the future of your project when discontinued