



Application of HPX to Tiled GEMM and QR: A Benchmark

September 25, 2019 | Thomas Miethlinger | Jülich Supercomputing Centre

Part I: Introduction

About me

(Thomas Miethlinger)

- Study: Master Physics
- Johannes Kepler University of Linz
- Institute for Theoretical Physics
Department Many Particle Systems
- Research:
 - Quantum fluids
 - Complex fluids
 - Non-equilibrium statistical mechanics

About the GSP

- Supervisor: Dr. Edoardo Di Napoli
- Co-Supervisor: Dr. Xinzhe Wu
- SimLab Quantum Materials
- Research:
 - Development and maintenance of numerical libraries
 - Design and implementation of high-performance algorithms
 - Development of new mathematical and computational models within a methodological frameworkin the scope of computational materials science and quantum materials.

Part II: Introduction to HPX

Current situation in high performance computing (HPC)

Currently, speed-up in computing does not stem from higher CPU frequency, but increased parallelism. However, we already face the following challenges in HPC:

- Ease of programming
- Inability to handle dynamically changing workloads
- Scalability
- Efficient utilization of system resources

⇒ a need for a new execution model: ParalleX, which is implemented by HPX

ParalleX

ParalleX is a new parallel execution model that offers an alternative to the conventional computation models(e.g. message passing):

- Split-phase transaction model
- Message-driven
- Distributed shared memory
- Multi-threaded
- Futures synchronization
- Local Control Objects (LCOs)
- ...

ParalleX focusses on latency hiding instead of latency avoidance.

About HPX

- High Performance ParallelX (HPX) is the first runtime system implementation of the ParallelX execution model.
- Development: STE||AR group
Louisiana State University
LSU Center for Computation and Technology
- Released as open source under the Boost Software License
- Current version: HPX V1.3.0, released on 23.05.2019
- Aims to be a **C++ standards conforming implementation** of the Parallelism and Concurrency proposals for C++ 17/20/23/...
- This means: HPX is a C++ library that supports **dynamic adaptive resource management** and **lightweight task programming and scheduling** within the context of a **global address space**.

On learning HPX

An opinion of a non-CS/HPC student

Learning curve on of HPX is quite steep - in the first days quite some dedication, effort and endurance is needed¹.

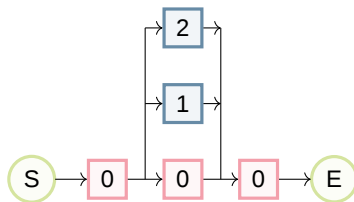
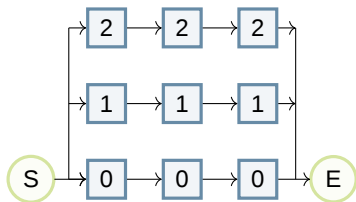
- Probably the easiest way in the beginning: watch [this nice playlist](#) in 1.25x speed on the youtube channel of [cscsch](#) (Swiss National Supercomputing Centre)
- Be aware that the [API reference](#) is not complete
- Be aware that there exist at least 5 different “Hello, World!” examples²:
 - `hpx/examples/hello_world_component/*`: 3 files; 28, 30 & 55 lines
 - `hpx/examples/quickstart/hello_world_1.cpp`; 22 lines
 - `hpx/examples/quickstart/hello_world_2.cpp`; 24 lines
 - `hpx/examples/quickstart/hello_world_distributed.cpp`; 156 lines
 - `tutorials/examples/01_hello_world/hello_world.cpp`; 71 lines

¹Why is the HPX code repo so big and complicated?

²Paths are with respect to <https://github.com/STELLAR-GROUP/>

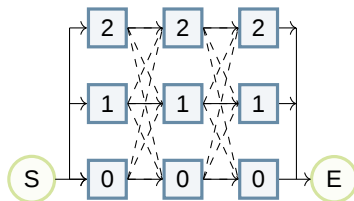
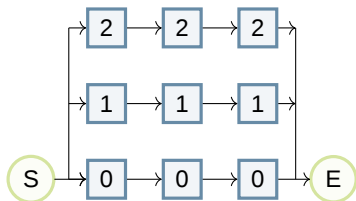
Comparison of HPX and OpenMP

| HPX | OpenMP |
|---|---|
| C++ library Core language: <code>hpx::C++</code> Task-based parallelism AGAS (active global address space) | Compiler extension to C and Fortran <code>#pragma omp directives</code> Parallel regions (fork-join model) shared memory |



Comparison of HPX and MPI

| HPX | MPI |
|---|---|
| C++ library Core language: <code>hpx::C++</code> Task-based parallelism AGAS (active global address space) | Interface specification for C and Fortran Core language: <code>MPI_C</code> , <code>MPI_F08</code> Single program, multiple data (SPMD) Explicit message passing |



HPX: Tasks and Threads

- HPX: Task-based parallelism
- Split up big problem into smaller tasks
- Tasks are worked off as HPX (lightweight) Threads by the OS Threads
- Task size is crucial: not too small and not too big
- Number of tasks can even be as high as $O(10^8)$



HPX: Example Program

```
double calc_area(hpx::future<double> future_r, hpx::future<double> future_pi)
{
    double r = future_r.get();
    double pi = future_pi.get();
    return r * rmake * pi;
}

int hpx_main(variables_map& vm) // In hpx_main the HPX environment is loaded
{
    hpx::future<double> future_r = hpx::make_ready_future(vm["r"].as<double>());
    hpx::future<double> future_pi = hpx::async([](){ return 4.0 * atan(1.0); });
    hpx::future<double> future_area = hpx::dataflow(&calc_area, future_r, future_pi);
    return hpx::finalize(); // Area can be obtained by: future_area.get()
}

int main(int argc, char * argv[]) // Start program by: ./area --r=...
{
    options_description.add_options()("r", value<double>()->default_value(1.0), "Radius: r");
    return hpx::init(options_description, argc, argv); // hpx::init calls hpx_main
}
```

Part III: Overview of numerical linear algebra and its applications

Part IV: GEMM

Part V: QR



Application of HPX to Tiled GEMM and QR: A Benchmark

September 25, 2019 | Thomas Miethlinger | Jülich Supercomputing Centre