Application of HPX to Tiled GEMM and QR: A Benchmark

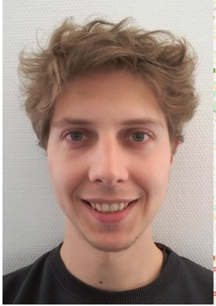
October 1, 2019 | Thomas Miethlinger | Johannes Kepler University Linz

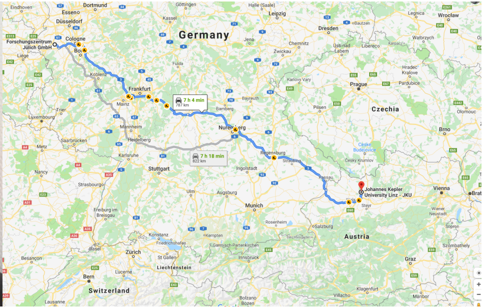
Part I: Introduction

About me

(Thomas Miethlinger)

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

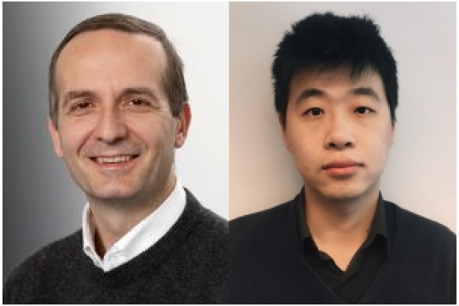




About the Supervisors

- 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 framework

in the scope of computational materials science and quantum materials.



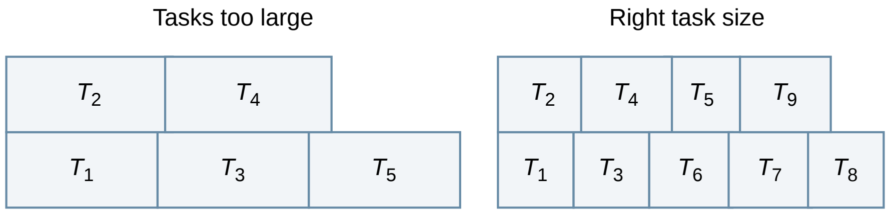
Part II: Introduction to HPX

About HPX

- High Performance ParalleX (HPX) is the first runtime system implementation of the ParalleX 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**.

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 $\mathcal{O}(10^8)$

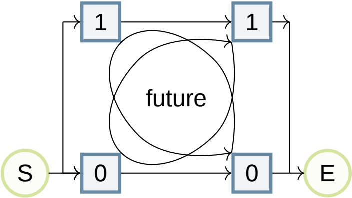


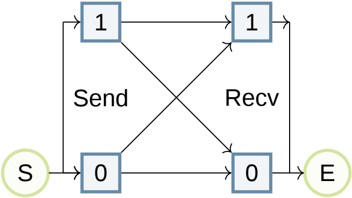
Comparison of HPX and OpenMP

HPX	OpenMP
C++ library	Compiler extension to C and Fortran
Core language: `hpx::C++`	`#pragma omp` directives
Task-based parallelism	Parallel regions (fork-join model)

Comparison of HPX and MPI

HPX	MPI
C++ library	Interface specification for C and Fortran
Core language: `hpx::C++`	Core language: MPI_C, MPI_F08
Task-based parallelism	Single program, multiple data (SPMD)
Message driven	Message passing





The HPX API

Selection: Classes

Class	Description
`hpx::thread`	Low level thread of control
`hpx::mutex`	Low level synchronization facility
`hpx::lcos::local::condition_variable`	Signal a condition
`hpx::future`, `hpx::shared_future`	Asynchronous result transport (receiving end)
`hpx::promise`, `hpx::lcos::local::promise`	Asynchronous result transport (producing end)
`hpx::lcos::packaged_task`	Asynchronous result transport (producing end)
`hpx::function`	Type erased function object
`hpx::tuple`	Tuple
...	

The HPX API

Selection: Functions

Functions	Description
`hpx::async`	Spawning tasks (returns a future)
`hpx::make_ready_future`	Spawning tasks (returns a ready future)
`hpx::bind`	Binding Parameters to callables
`hpx::apply`	Signal a condition
`future::{is_ready, valid, has_exception}`	Query state of future
`future::get`	Return computed result of future
`future::then`	Continuations of futures
`hpx::when_all, hpx::when_any, hpx::when_n`	Waiting on one or more futures (non blocking)
`hpx::wait_all, hpx::wait_any, hpx::wait_n`	Waiting on one or more futures (blocking)
`hpx::dataflow`	Shortcut to hpx::when_all(...).then(...)
`...`	

Part III: Introduction to Numerical Linear Algebra, GEMM and QR

JÜLICH
Forschungszentrum

Introduction: Numerical Linear Algebra

Numerical linear algebra is a subfield of numerical analysis and linear algebra, and it plays an integral role in computational problem solving. There exist many several algorithms for common problems, a few well-known are:

- Solving systems of linear equations
- Eigenvalue problem
- Matrix inversion problem
- Least-squares problem

which may be using one of the following matrix operations/decompositions:

- Matrix multiplications
- LU decomposition
- QR decomposition
- Spectral decomposition
- Singular value decomposition

GEMM

Applications of Numerical Linear Algebra

GEMM - GEneral Matrix Multiply

- Basic binary operation in Linear Algebra, which has numerous applications in mathematics, science and engineering.
- More fundamental applications of matrix multiplications include
  - Systems of Linear Algebraic Equations (SLAE) can be expressed as a single matrix equation, e.g. $Ax = y$.
  - Linear map between two vector spaces U and V over the same field F.
- Motivation: A large amount (>70%) of runtime in ChASE ((E.D. Napoli, 2019)) routine is used for GEMM.
- Let the field F be $\mathbb{R}$ or $\mathbb{C}$, $A = \left(a_{ij}\right) \in F^{m \times n}, B = \left(b_{jk}\right) \in F^{n \times p}$. Then,

$$C = \left(c_{ik}\right) = AB \in F^{m \times p}, \tag{1}$$

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \tag{2}$$

GEMM

GEMM and GEMM of Blocked Matrices

$\Longrightarrow$ most simple implementation consists of 3 nested for-loops:

```
for 0 <= i < m, 0 <= j < n, 0 <= k < p, do: C[i][k] += A[i][j] * B[j][k]
```

Better approach: Discretize matrices into blocks, perform GEMM **block-wise**
Let the field F again be $\mathbb{R}$ or $\mathbb{C}$, $A = \left(A_{ij}\right) \in F^{M \times N \times m \times n}$, $B = \left(B_{jk}\right) \in F^{N \times P \times n \times p}$.
Then:

$$C = \left(C_{ik}\right) = AB \in F^{M \times P \times m \times p}, \tag{3}$$

$$C_{ik} = \left(c_{ik,i'k'}\right) = \sum_{j=1}^{n} A_{ij} B_{jk}, \tag{4}$$

$$c_{ik,i'k'} = \sum_{j=1}^{N} \sum_{j'=1}^{n} a_{ij,i'j'} b_{jk,j'k'} \tag{5}$$

GEMM of Blocked Matrices

A Small Example

Let $m = n = p = M = N = P = 2$:

$$A \cdot B = C$$

$$\begin{bmatrix} A_{00} & A_{10} \\ A_{01} & A_{11} \end{bmatrix} \cdot \begin{bmatrix} B_{00} & B_{10} \\ B_{01} & B_{11} \end{bmatrix} = \begin{bmatrix} C_{00} & C_{10} \\ C_{01} & C_{11} \end{bmatrix}$$

GEMM of Blocked Matrices

A Small Example

Let $m = n = p = M = N = P = 2$:

$$\boxed{A_{ij}} \cdot \boxed{B_{jk}} = \boxed{A_{ij}B_{jk}}$$

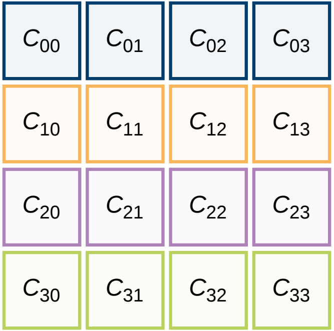
$$\begin{bmatrix} a_{ij,00} & a_{ij,10} \\ a_{ij,01} & a_{ij,11} \end{bmatrix} \cdot \begin{bmatrix} b_{jk,00} & b_{jk,10} \\ b_{jk,01} & b_{jk,11} \end{bmatrix} = \begin{bmatrix} \cdots & \cdots \\ \cdots & \cdots \end{bmatrix}$$

GEMM of Blocked Matrices

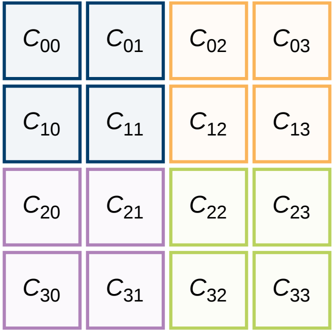
OpenMP and HPX foreach Row

Let $M = N = 4$, number of threads $\mathcal{T} = 4$:

OMP Row Tiled & HPX foreach Row



OMP Cell Tiled



GEMM of Blocked Matrices

HPX future C_{ik} and HPX future $A_{ij}B_{jk}$

HPX future $f_{ik} \rightarrow C_{ik}$

...	$f_{ik} \rightarrow C_{ik}$	...

$$C_{ik} = f_{ik}.\texttt{get()}$$

HPX future $f_{ijk} \rightarrow A_{ij}B_{jk}$

...	$f_{ijk} \rightarrow A_{ij}B_{jk}$	...

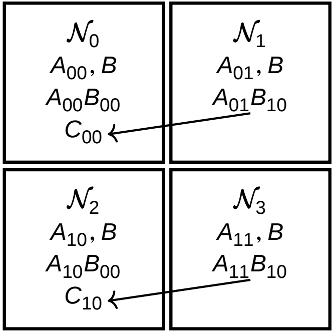
$$C_{ik} = \sum_j f_{ijk}.\texttt{get()}$$

GEMM of Blocked Matrices

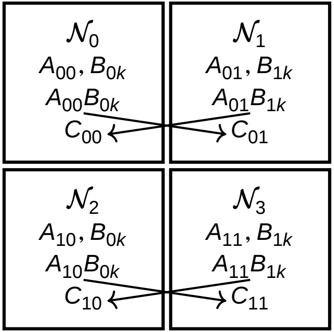
Distributed Memory

Let $M = N = 2$, number of nodes $\mathcal{N} = 4$:

MPI V1 & HPX dist V1



MPI V2 & HPX dist V2



QR Decomposition

Applications of Numerical Linear Algebra

- Matrix decomposition of square or rectangular matrices (we only consider square matrices).
- Let the field F be $\mathbb{R}$ or $\mathbb{C}$, $A = \left(a_{ij}\right) \in F^{m \times m}$. Then,

$$A = QR, \tag{6}$$

  where Q is a orthogonal($F = \mathbb{R}$) / unitary($F = \mathbb{C}$) matrix, and R is an upper triangular matrix.
- Motivation: A large amount (>20%) of runtime in ChASE ((E.D. Napoli, 2019)) routine is used for QR.
- Computing the QR decomposition:
  - Gram-Schmidt process
  - **Householder reflection**
  - Givens rotations

Applications of QR Decomposition

Examples

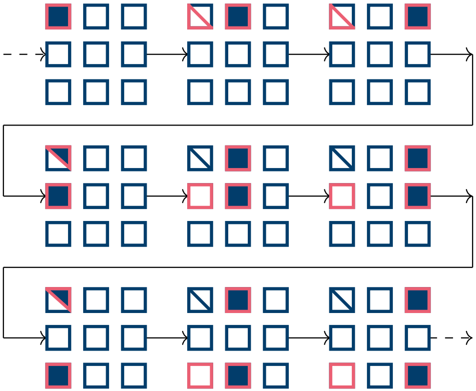
- Computing Eigenvalues (QR algorithm)
- Computing orthogonal base
- Solving least-squares problem
- Solution to linear inverse problems

A selection of papers within the scope of my research interest which use QR decomposition:

- An evaluation of noise reduction algorithms for particle-based fluid simulations in multi-scale applications (M.J. Zimon et al., 2016)
- Dynamic mode decomposition of numerical and experimental data (P.J. Schmid, 2010)
- Krylov Methods for the Incompressible Navier-Stokes Equations (W.S. Edwards, 1994)
- Computing Lyapunov exponents of continuous dynamical systems: method of Lyapunov vectors (J. Lu, 2005)

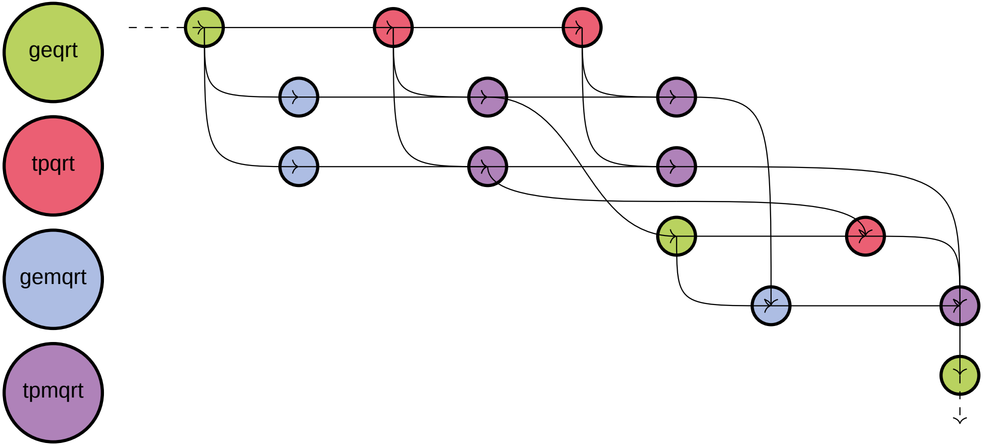
Block QR Decomposition

Example: 3×3 **block matrix**



Block QR Decomposition

Example: 3×3 **block matrix**

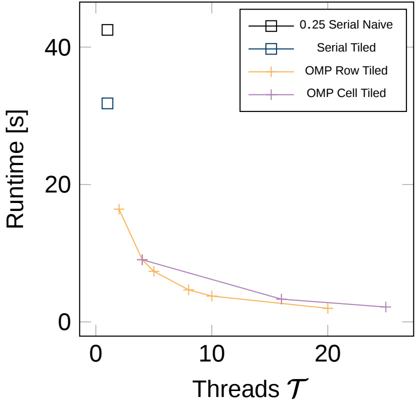


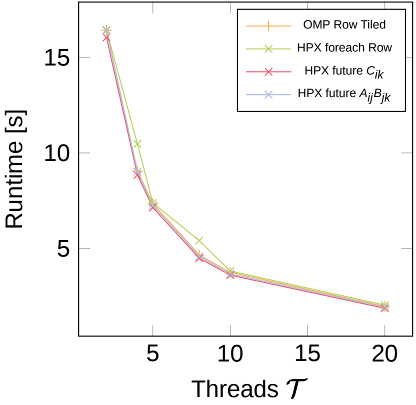
Part IV: Benchmark: Results for GEMM

JÜLICH Forschungszentrum

GEMM Shared Memory

Let $m = n = p = 4000$, $M = N = P = 40$, vary number of threads $\mathcal{T}$:

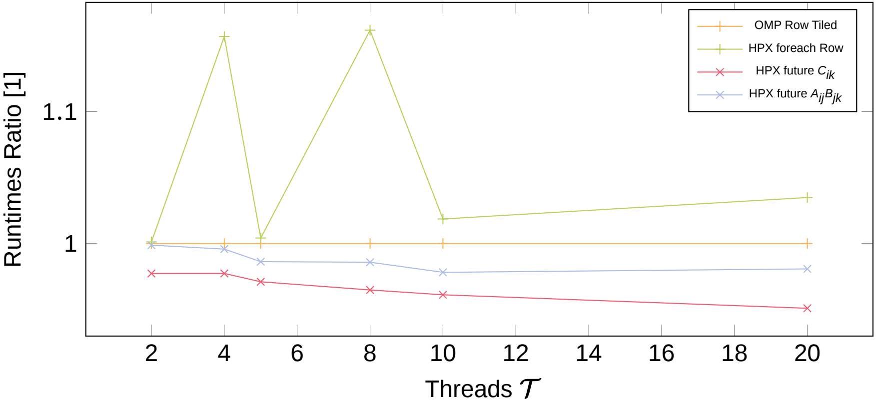




JÜLICH Forschungszentrum

GEMM Shared Memory

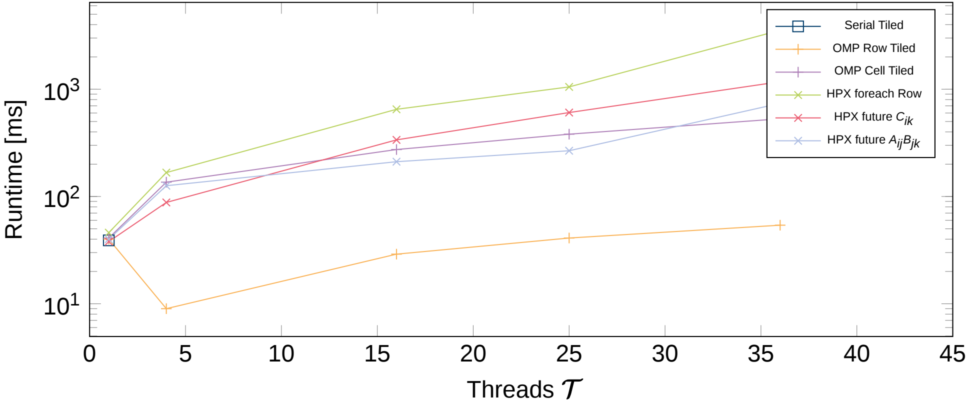
Let $m = n = p = 4000$, $M = N = P = 40$, vary number of threads $\mathcal{T}$:



$\Longrightarrow$ HPX can be faster than OpenMP, even for embarrassingly parallel problems, in particular `hpx::futures`.

GEMM Shared Memory

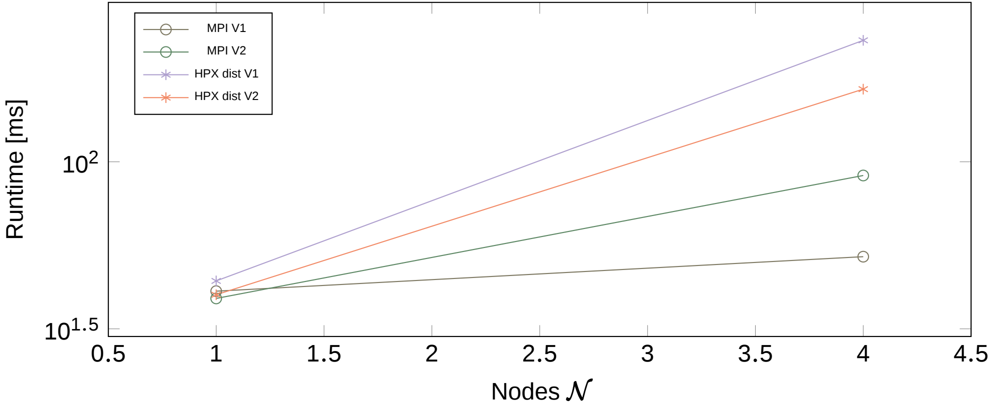
Vary number of threads $\mathcal{T}$, and let $m = n = p = 400\sqrt{\mathcal{T}}, M = N = P = \sqrt{\mathcal{T}}$:



$\Longrightarrow$ Task-size crucial for HPX. Here the block size was too big.

GEMM Distributed Memory

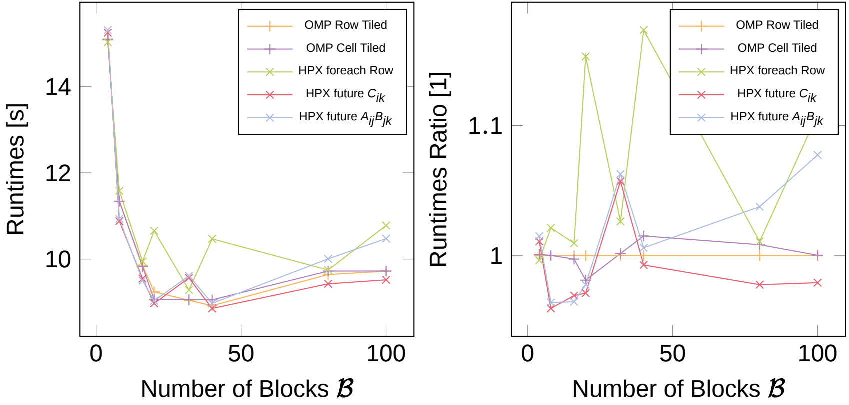
Vary number of Nodes $\mathcal{N}$, and let $m = n = p = 400\sqrt{\mathcal{N}}$, $M = N = P = \sqrt{\mathcal{N}}$:



HPX distributed is slower than MPI, their implementations are presumably analogous.

GEMM Shared Memory

Let $m = n = p = 4000$, $\mathcal{T} = 4$, vary number of blocks $\mathcal{B} := M = N = P$:

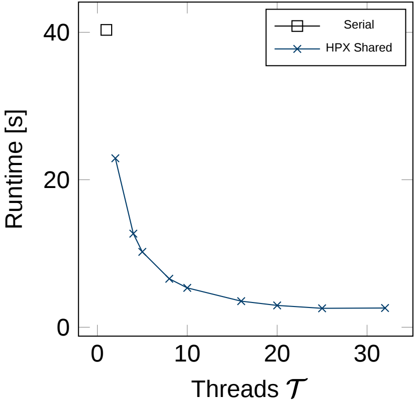


$\Longrightarrow$ Again, HPX is around the same speed as OpenMP.

Part V: Benchmark: Results for Tiled QR

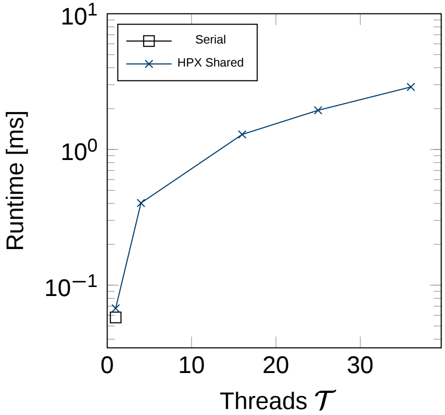
QR Shared Memory

Let $m = 4000$, $M = 40$, vary number of threads $\mathcal{T}$:



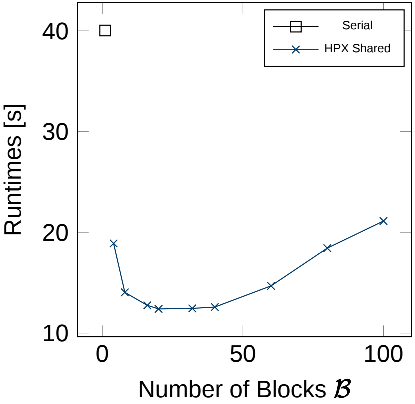
QR Shared Memory

Vary number of threads $\mathcal{T}$, and let $m = n = p = 400\sqrt{\mathcal{T}}$, $M = N = P = \sqrt{\mathcal{T}}$:



QR Shared Memory

Let $m = n = p = 4000$, $\mathcal{T} = 4$, vary number of blocks $\mathcal{B} := M = N = P$:



$\implies$ Discretizing problem into blocks has a high impact on runtime!

Part VI: Conclusion and Resume

Conclusion

On HPX

- HPX has the potential to speed-up and simplify parallelization for certain types of problems
- On par with OpenMP
- Comparison with MPI unfortunately not possible at this moment (supplemented for report)
- Learning HPX takes some time, in particular HPX for distributed programs
- Debugging, profiling, runtime errors, documentation, ... could be improved
- Execution policies: might give additional speed-up
- Altogether: Very strong library and concept with some negative points

Application of HPX to Tiled GEMM and QR: A Benchmark

October 1, 2019 | Thomas Miethlinger | Johannes Kepler University Linz

HPX: Example Program

```
double calc_area(hpx::future<double> future_r, hpx::future<double> future_pi)
{
    double r = future_r.get(); // r is returned immediately (make_ready_future)
    double pi = future_pi.get(); // pi is returned once the async computation finishes
    return r * r * pi; // return area = r^2 * pi
}

int hpx_main(variables_map& vm) // In hpx_main the HPX environment is loaded
{   // boost::program_options::variables_map: retrieve commandline arguments
    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=...
{   // boost::program_options::options_description: handle commandline arguments
    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
}
```