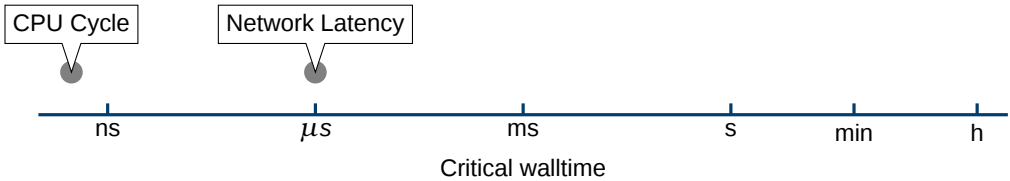
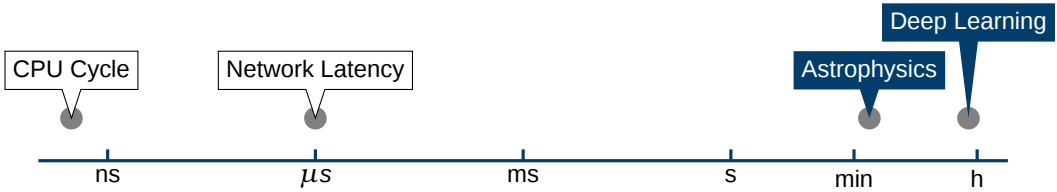
Adding Inter-node Communication to a C++ Tasking Framework

April 15, 2019 | M. Innerberger, L. Morgenstern, A. Beckmann, I. Kabadshow | Juelich Supercomputing Centre

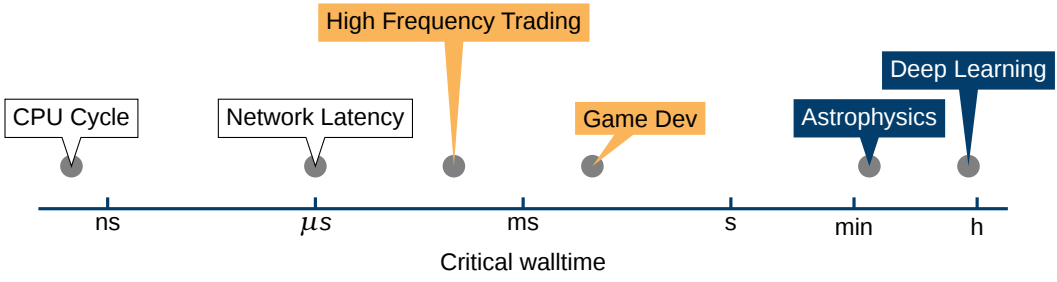
HPC ≠ HPC


CPU Cycle
Network Latency
ns
μs
ms
s
min
h
Critical walltime

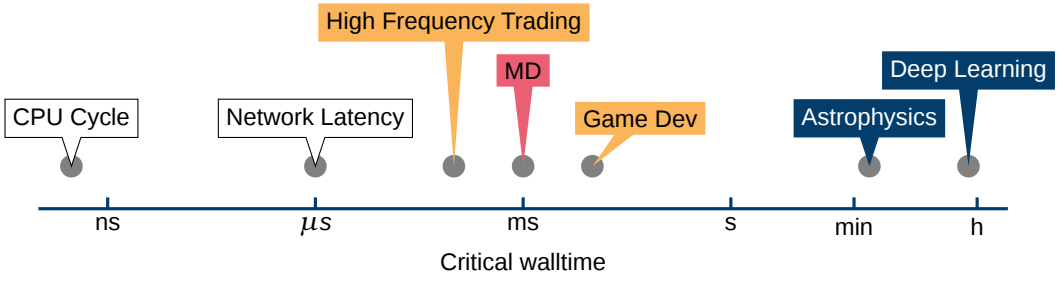
HPC ≠ HPC


Deep Learning
CPU Cycle
Network Latency
Astrophysics
ns
μs
ms
s
min
h
Critical walltime

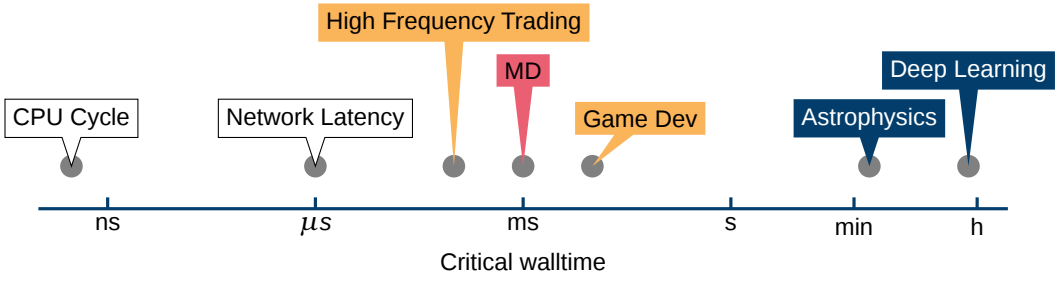
HPC ≠ HPC


High Frequency Trading
Deep Learning
CPU Cycle
Network Latency
Game Dev
Astrophysics
ns
μs
ms
s
min
h
Critical walltime

HPC ≠ HPC


High Frequency Trading
MD
Deep Learning
CPU Cycle
Network Latency
Game Dev
Astrophysics
ns
μs
ms
s
min
h
Critical walltime

HPC ≠ HPC



Requirements for MD

- Strong scalability
- Performance portability

Our Motivation

Solving Coulomb problem for Molecular Dynamics

Task: Compute all pairwise interactions of N particles

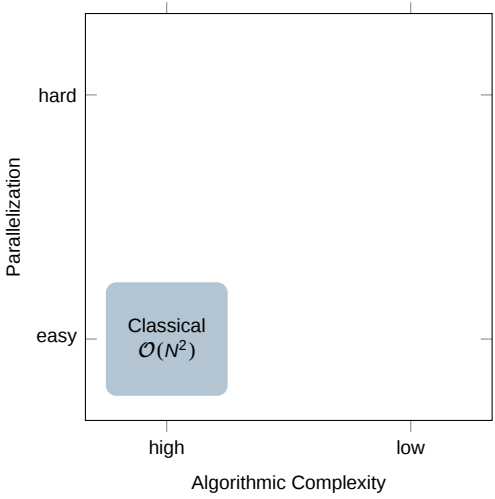
N-body problem: $\mathcal{O}(N^2) \rightarrow \mathcal{O}(N)$ with FMM

Why is that an issue?

- MD targets $< 1ms$ runtime per time step
- MD runs millions or billions of time steps
- not compute-bound, but synchronization bound
- no libraries (like BLAS) to do the heavy lifting

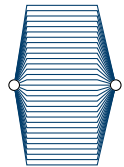
We might have to look under the hood ... and get our hands dirty.

Parallelization Potential

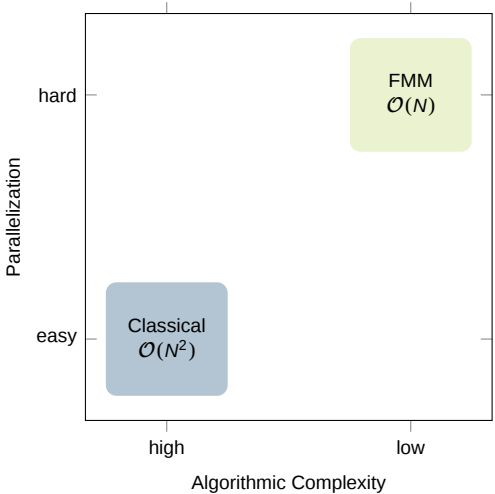


Classical Approach

- Lots of independent parallelism

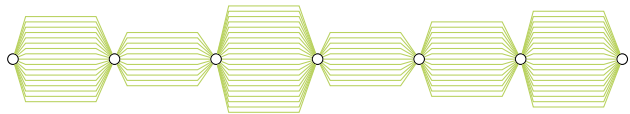


Parallelization Potential



Fast Multipole Method (FMM)

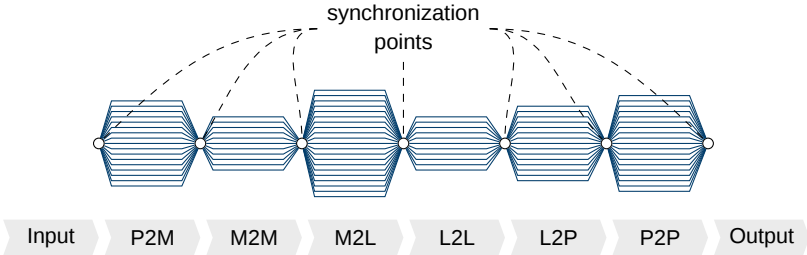
- Many dependent phases
- Varying amount of parallelism



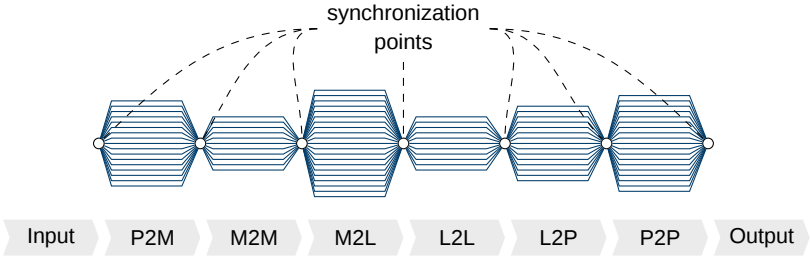
JÜLICH Forschungszentrum

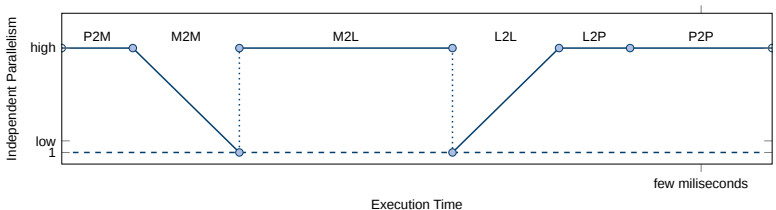
Coarse-Grained Parallelization

Coarse-Grained Parallelization

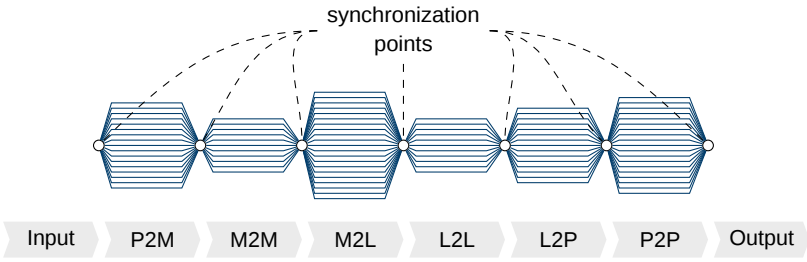


Coarse-Grained Parallelization





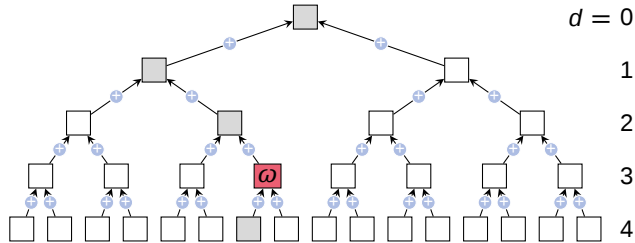
Coarse-Grained Parallelization



- Different amount of available loop-level parallelism within each phase
- Some phases contain sub-dependencies
- Synchronizations might be problematic

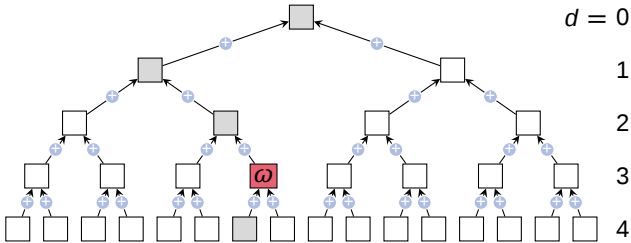
FMM Algorithmic Flow

Multipole to multipole (M2M), shifting multipoles upwards

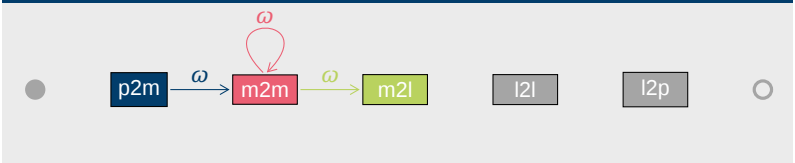


FMM Algorithmic Flow

Multipole to multipole (M2M), shifting multipoles upwards

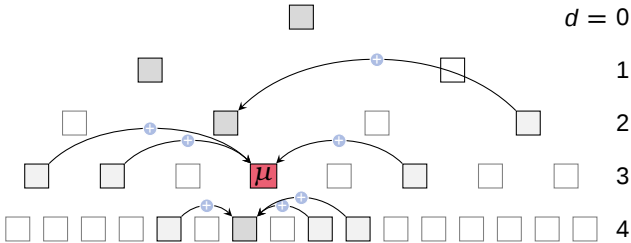


Dataflow – Fine-grained Dependencies



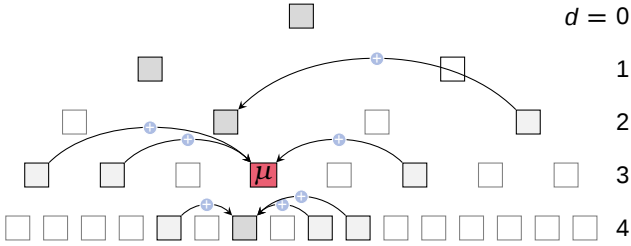
FMM Algorithmic Flow

Multipole to local (M2L), translate remote multipoles into local taylor moments

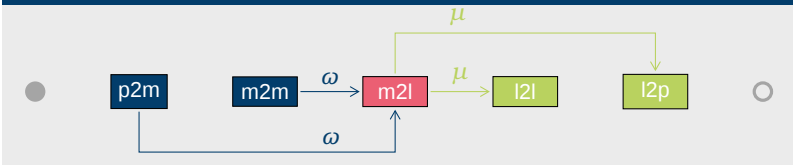


FMM Algorithmic Flow

Multipole to local (M2L), translate remote multipoles into local taylor moments

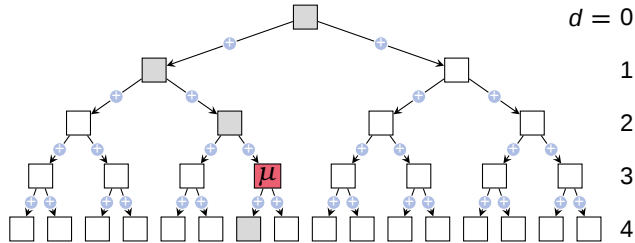


Dataflow – Fine-grained Dependencies



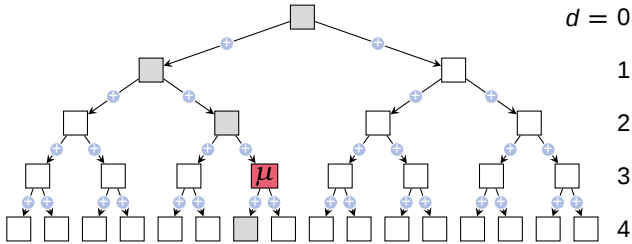
FMM Algorithmic Flow

Local to local (L2L), shifting Taylor moments downwards

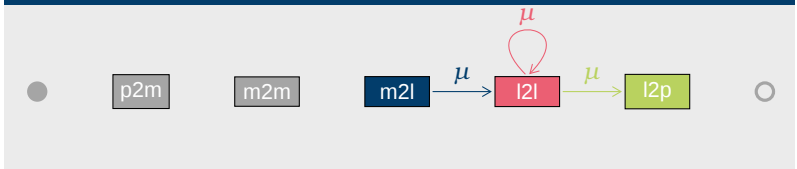


FMM Algorithmic Flow

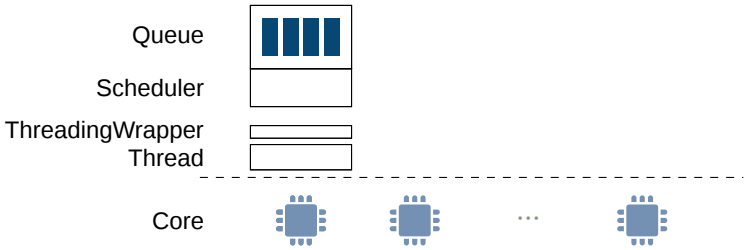
Local to local (L2L), shifting Taylor moments downwards



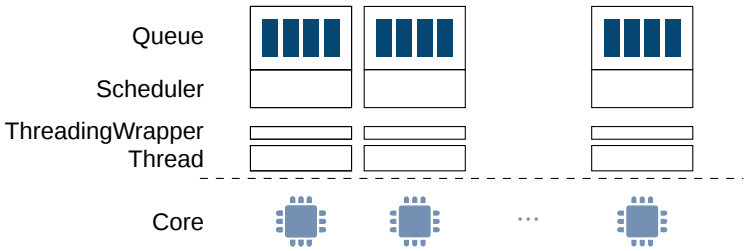
Dataflow – Fine-grained Dependencies



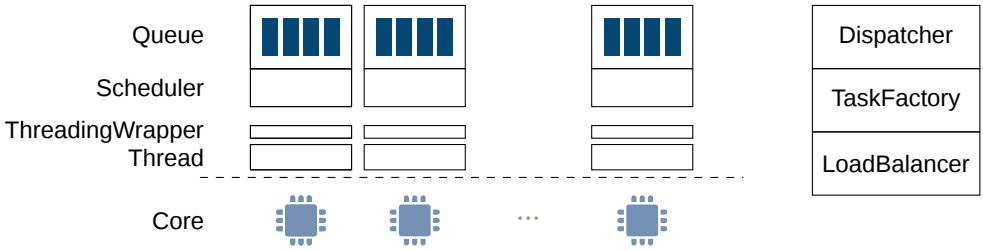
CPU Tasking Framework



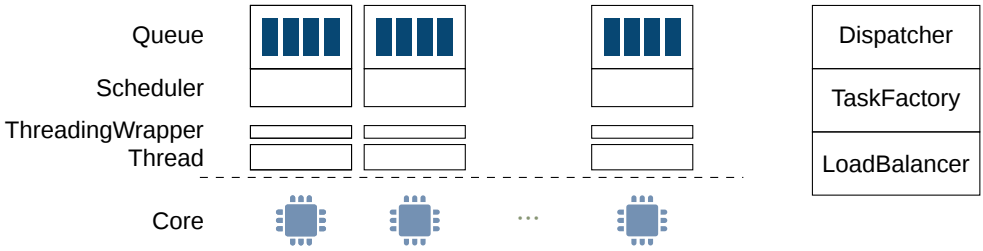
CPU Tasking Framework



CPU Tasking Framework

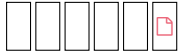


CPU Tasking Framework



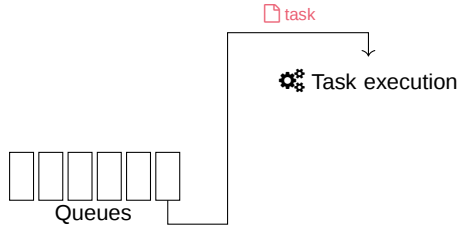
CPU Tasking Framework

Task life-cycle per thread



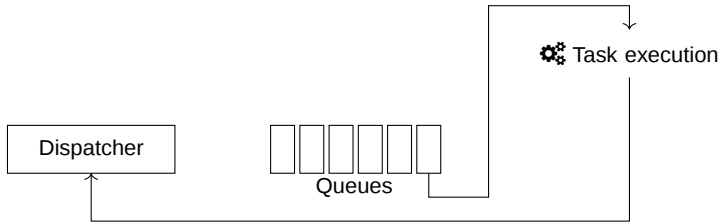
CPU Tasking Framework

Task life-cycle per thread



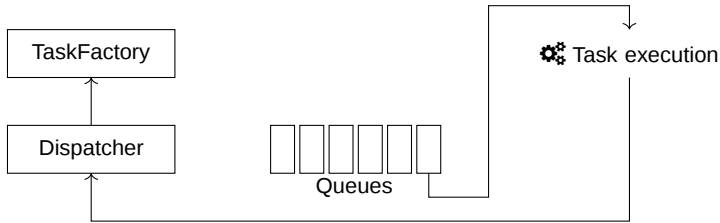
CPU Tasking Framework

Task life-cycle per thread



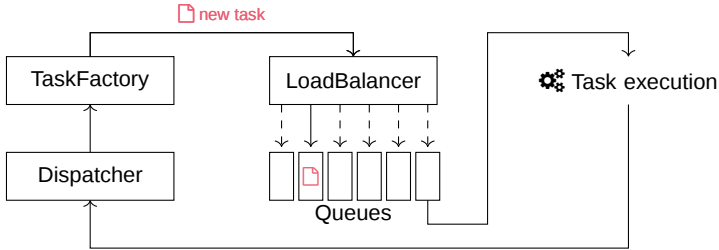
CPU Tasking Framework

Task life-cycle per thread



CPU Tasking Framework

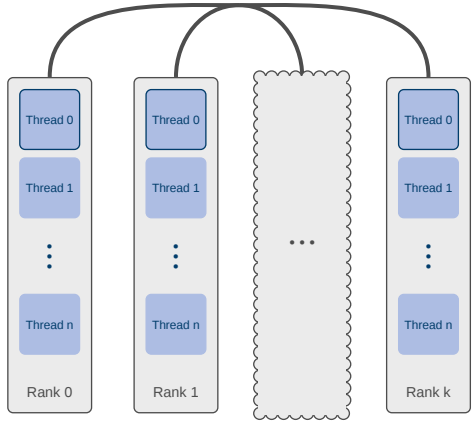
Task life-cycle per thread



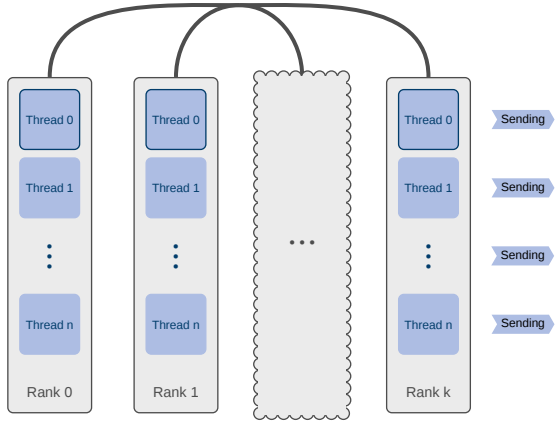
- Tasks can be prioritized by task type
- Only ready-to-execute tasks are stored in queue
- Workstealing from other threads is possible

Part I: Inter-node Parallelization

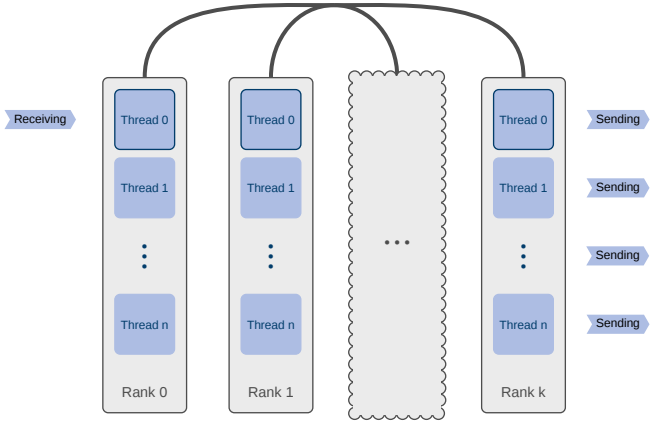
Adding Inter-node Communication via MPI



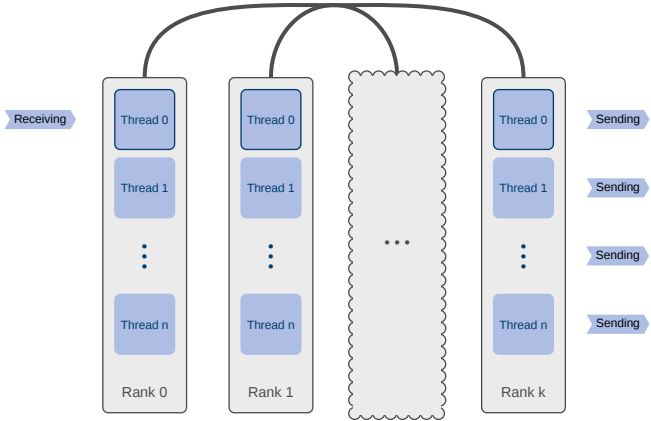
Adding Inter-node Communication via MPI



Adding Inter-node Communication via MPI



Adding Inter-node Communication via MPI



Rationale: writing to data structure should not be concurrent ⟶ avoid critical sections

Recapitulation of Method

- Computation of multipoles on lowest level (P2M) divided equally among ranks

Recapitulation of Method

- Computation of multipoles on lowest level (P2M) divided equally among ranks
- Multipoles are sent to all ranks so that every rank has all data

Recapitulation of Method

- Computation of multipoles on lowest level (P2M) divided equally among ranks
- Multipoles are sent to all ranks so that every rank has all data
- All threads can send, only main thread can receive

Recapitulation of Method

- Computation of multipoles on lowest level (P2M) divided equally among ranks
- Multipoles are sent to all ranks so that every rank has all data
- All threads can send, only main thread can receive
- Only necessary local moments are computed during M2L

Recapitulation of Method

- Computation of multipoles on lowest level (P2M) divided equally among ranks
- Multipoles are sent to all ranks so that every rank has all data
- All threads can send, only main thread can receive
- Only necessary local moments are computed during M2L
- Computation of forces also restricted (L2P and P2P)

Part II: Implementation

General Remarks

- Tasking engine for shared memory already existing

General Remarks

- Tasking engine for shared memory already existing
- Highly configurable library (e.g. change threading wrapper, locks, ...)

General Remarks

- Tasking engine for shared memory already existing
- Highly configurable library (e.g. change threading wrapper, locks, ...)
- C++11 code with heavy use of Template Meta Programming (TMP)

General Remarks

- Tasking engine for shared memory already existing
- Highly configurable library (e.g. change threading wrapper, locks, ...)
- C++11 code with heavy use of Template Meta Programming (TMP)
  - TMP allows to resolve function calls at compile time

General Remarks

- Tasking engine for shared memory already existing
- Highly configurable library (e.g. change threading wrapper, locks, ...)
- C++11 code with heavy use of Template Meta Programming (TMP)
  - TMP allows to resolve function calls at compile time
  - Data Flow Dispatcher works in a static way (little overhead)

General Remarks

- Tasking engine for shared memory already existing
- Highly configurable library (e.g. change threading wrapper, locks, ...)
- C++11 code with heavy use of Template Meta Programming (TMP)
  - TMP allows to resolve function calls at compile time
  - Data Flow Dispatcher works in a static way (little overhead)
- Currently, `std::threads` is used for intra node parallelization

MPI Details

Code

```
while (notFinished()){
  executeTask();
  Status = Communicator.Iprobe();
  if (Status.MessageNeeded()){
    Communicator.Irecv(); }
  else {
    Communicator.Discard(); }
  /*do something else */
  Communicator.Wait();
  /*use received data */
}
```

MPI Calls

MPI Details

Code

```
while (notFinished()){
  executeTask();
  Status = Communicator.Iprobe();
  if (Status.MessageNeeded()){
    Communicator.Irecv(); }
  else {
    Communicator.Discard(); }
  /*do something else */
  Communicator.Wait();
  /*use received data */
}
```

MPI Calls

- Isend to all ranks

MPI Details

Code

```
while (notFinished()){
  executeTask();
  Status = Communicator.Iprobe();
  if (Status.MessageNeeded()){
    Communicator.Irecv(); }
  else {
    Communicator.Discard(); }
  /*do something else */
  Communicator.Wait();
  /*use received data */
}
```

MPI Calls

- Isend to all ranks
- Iprobe busy waiting for messages

MPI Details

Code

```
while (notFinished()){
  executeTask();
  Status = Communicator.Iprobe();
  if (Status.MessageNeeded()){
    Communicator.Irecv(); }
  else {
    Communicator.Discard(); }
  /*do something else */
  Communicator.Wait();
  /*use received data */
}
```

MPI Calls

- Isend to all ranks
- Iprobe busy waiting for messages
- Call Irecv for messages in any case

MPI Details

Code

```
while (notFinished()){
  executeTask();
  Status = Communicator.Iprobe();
  if (Status.MessageNeeded()){
    Communicator.Irecv(); }
  else {
    Communicator.Discard(); }
  /*do something else */
  Communicator.Wait();
  /*use received data */
}
```

MPI Calls

- Isend to all ranks
- Iprobe busy waiting for messages
- Call Irecv for messages in any case
- If message not needed, write data to dummy buffer

MPI Details

Code

```
while (notFinished()){
  executeTask();
  Status = Communicator.Iprobe();
  if (Status.MessageNeeded()){
    Communicator.Irecv(); }
  else {
    Communicator.Discard(); }
  /*do something else */
  Communicator.Wait();
  /*use received data */
}
```

MPI Calls

- Isend to all ranks
- Iprobe busy waiting for messages
- Call Irecv for messages in any case
- If message not needed, write data to dummy buffer
- Call Wait before using data

Distinguishing Incoming Messages

```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

It’s in the Tag

Distinguishing Incoming Messages

```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

It's in the Tag

- MPI send / receive operations need a tag (integer)

Distinguishing Incoming Messages

```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

0 1 0 ⋯ 0 1 1 0 1 1 0 1

It's in the Tag

- MPI send / receive operations need a tag (integer)
- Information can be encoded in this tag

Distinguishing Incoming Messages

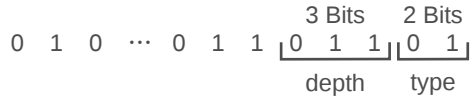
```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

It's in the Tag

- MPI send / receive operations need a tag (integer)
- Information can be encoded in this tag
- Type of sent data (multipole, local moment, particle)

Distinguishing Incoming Messages

```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

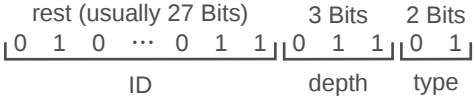


It's in the Tag

- MPI send / receive operations need a tag (integer)
- Information can be encoded in this tag
- Type of sent data (multipole, local moment, particle)
- Level of the corresponding box ($2^{d \cdot \text{level}}$ boxes)

Distinguishing Incoming Messages

```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

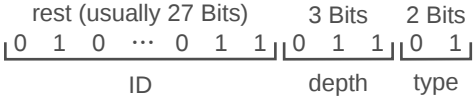


It's in the Tag

- MPI send / receive operations need a tag (integer)
- Information can be encoded in this tag
- Type of sent data (multipole, local moment, particle)
- Level of the corresponding box ($2^{d\cdot\text{level}}$ boxes)
- ID of box on this level

Distinguishing Incoming Messages

```
MPI_I[send|recv](buf,
                 count,
                 datatype,
                 [dest|source],
                 tag,
                 comm,
                 request)
```

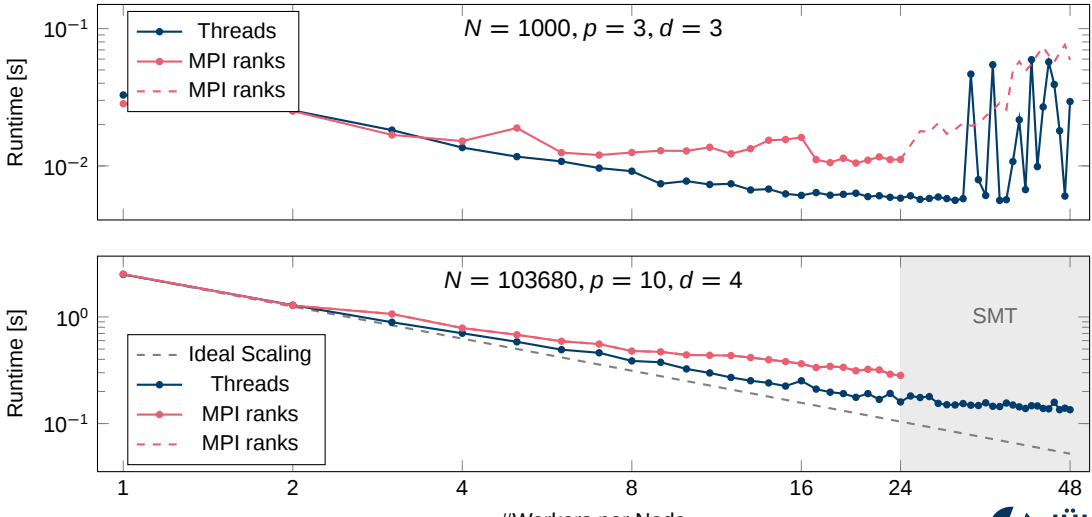


It's in the Tag

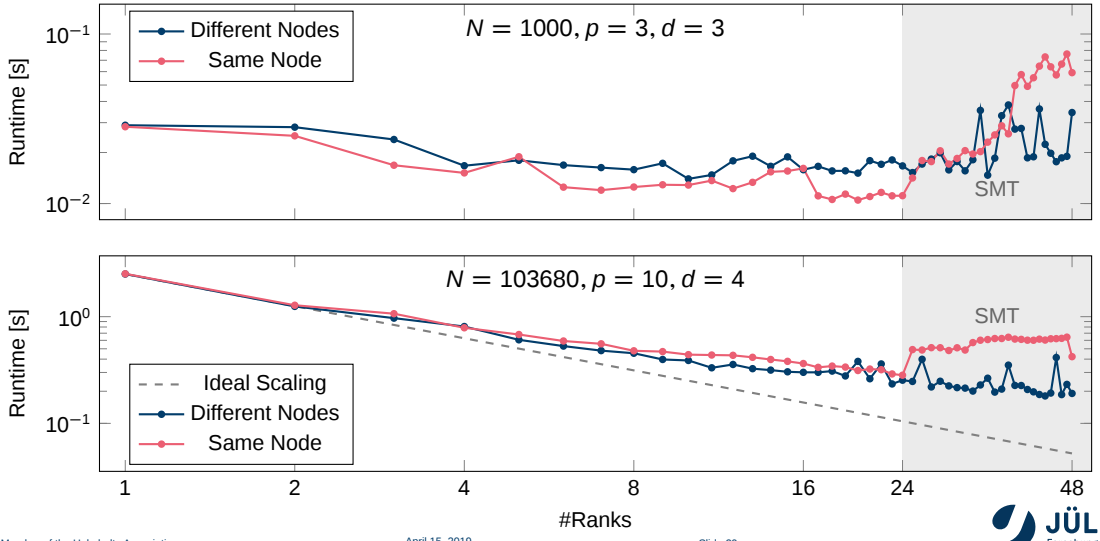
- MPI send / receive operations need a tag (integer)
- Information can be encoded in this tag
- Type of sent data (multipole, local moment, particle)
- Level of the corresponding box ($2^{d \cdot \text{level}}$ boxes)
- ID of box on this level
- This essentially mimics a matching probe / receive operation

Part III: Results and Outlook

Results from JURECA



Results from JURECA



Outlook

- Fix race condition in multithreaded MPI

Outlook

- Fix race condition in multithreaded MPI
- Handle message information more generally

Outlook

- Fix race condition in multithreaded MPI
- Handle message information more generally
- Maybe more restrictions in computed data per rank possible

Outlook

- Fix race condition in multithreaded MPI
- Handle message information more generally
- Maybe more restrictions in computed data per rank possible
- Maybe restrict send operations to just some ranks

Outlook

- Fix race condition in multithreaded MPI
- Handle message information more generally
- Maybe more restrictions in computed data per rank possible
- Maybe restrict send operations to just some ranks
- Communicate data in chunks

Adding Inter-node Communication to a C++ Tasking Framework

April 15, 2019 | M. Innerberger, L. Morgenstern, A. Beckmann, I. Kabadshow | Juelich Supercomputing Centre