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

Critical walltime

- CPU Cycle
- Network Latency

- ns
- μs
- ms
- s
- min
- h

Requirements for MD
- Strong scalability
- Performance portability
HPC ≠ HPC

- CPU Cycle
- Network Latency
- Deep Learning
- Astrophysics

Critical walltime:
- ns
- μs
- ms
- s
- min
- h

Requirements for MD:
- Strong scalability
- Performance portability
HPC ≠ HPC

- CPU Cycle
- Network Latency
- Game Dev
- High Frequency Trading
- Deep Learning
- Astrophysics

Critical walltime:
- ns
- μs
- ms
- s
- min
- h
HPC ≠ HPC

- Critical walltime
- CPU Cycle
- Network Latency
- High Frequency Trading
- MD
- Game Dev
- Deep Learning
- Astrophysics

Requirements for MD:
- Strong scalability
- Performance portability
HPC ≠ HPC

Critical walltime

- High Frequency Trading
- MD
- Game Dev
- Deep Learning
- Astrophysics

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

Algorithmic Complexity

high

low

easy

hard

Parallelization

Classical $O(N^2)$
Parallelization Potential

Fast Multipole Method (FMM)
- Many dependent phases
- Varying amount of parallelism

Algorithmic Complexity
- Classical O(N²)
- FMM O(N)

Parallelization
- Easy
  - O(N²)
- Hard
  - O(N)
Coarse-Grained Parallelization
Coarse-Grained Parallelization

Input P2M M2M M2L L2L L2P P2P Output

synchronization points
Coarse-Grained Parallelization

Input: P2M M2M M2L L2L L2P P2P
Output: synchronization points
few milliseconds
1 low high
Independent Parallelism
Execution Time
few milliseconds

Member of the Helmholtz Association
April 15, 2019
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

\[ d = 0 \]

\[ 1 \]

\[ 2 \]

\[ 3 \]

\[ 4 \]
FMM Algorithmic Flow

Multipole to multipole (M2M), shifting multipoles upwards

\[
d = \omega_0 + \omega_1 + \omega_2 + \omega_3 + \omega_4
\]

Dataflow – Fine-grained Dependencies

p2m \(\rightarrow\) m2m \(\rightarrow\) m2l \(\rightarrow\) l2l \(\rightarrow\) l2p
FMM Algorithmic Flow

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

\[ d = 0 \]

\[ \mu_0 + \mu_1 + \mu_2 + \mu_3 + \mu_4 \]

Dataflow – Fine-grained Dependencies
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

\[ d = \mu \]

Dataflow – Fine-grained Dependencies
FMM Algorithmic Flow
Local to local (L2L), shifting Taylor moments downwards

\[ d = 0 \]

Dataflow – Fine-grained Dependencies

Member of the Helmholtz Association
April 15, 2019
Slide 11
CPU Tasking Framework

Queue
Scheduler
ThreadingWrapper
Thread
Core

Dispatcher
TaskFactory
LoadBalancer
CPU Tasking Framework

Queue
Scheduler
ThreadingWrapper
Thread
Core
CPU Tasking Framework

Queue
Scheduler
ThreadingWrapper
Thread
Core
Dispatcher
TaskFactory
LoadBalancer
CPU Tasking Framework

Queue
Scheduler
ThreadingWrapper
Thread
Core
Dispatcher
TaskFactory
LoadBalancer
CPU Tasking Framework

Task life-cycle per thread

- Dispatcher
- TaskFactory
- LoadBalancer
- Queues
- Task execution
- Tasks can be prioritized by task type
- Only ready-to-execute tasks are stored in queue
- Workstealing from other threads is possible
CPU Tasking Framework

Task life-cycle per thread

- Dispatcher
- TaskFactory
- LoadBalancer
- Queues
- Task execution
- Queues

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

Task life-cycle per thread

- Dispatcher
- Queues
- Task execution

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

Task life-cycle per thread

- TaskFactory
- Dispatcher
- Queues
- Task execution

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

Rationale: writing to data structure should not be concurrent ⟶ avoid critical sections
Adding Inter-node Communication via MPI

Rationale: writing to data structure should not be concurrent ⟶ avoid critical sections
Adding Inter-node Communication via MPI

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

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

#### Code

```c
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

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>2 Bits</th>
<th>Depth</th>
<th>3 Bits</th>
<th>ID</th>
<th>rest (usually 27 Bits)</th>
</tr>
</thead>
</table>

It's in the Tag

This essentially mimics a matching probe / receive operation.
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

$$\text{MPI}_{-}\text{I}[send|recv]\text{(buf, count, datatype, [dest|source], tag, comm, request)}$$

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

Type of sent data (multipole, local moment, particle)
Level of the corresponding box (\(2d \cdot \text{level}\) boxes)
ID of box on this level
This essentially mimics a matching probe / receive operation
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)

0 1 0 ... 0 1 1 0 1 1

2 Bits

type
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

rest (usually 27 Bits) | 3 Bits | 2 Bits
---|---|---
0 1 0 ... 0 1 1 | 0 1 1 | 0 1

ID | depth | type
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

rest (usually 27 Bits)

0 1 0 ... 0 1 1 0 1 1 0 1

ID depth type
Part III: Results and Outlook
Results from JURECA

- **N = 1000, p = 3, d = 3**
  - Runtime [s]
  - Threads
  - MPI ranks

- **N = 103680, p = 10, d = 4**
  - Runtime [s]
  - Ideal Scaling
  - Threads
  - MPI ranks
Results from JURECA

$N = 1000, p = 3, d = 3$

$N = 103680, p = 10, d = 4$
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
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