

High-Performance Computing for the simulation of particles with the Discrete Element Method

13th International SuperComputing Camp
<https://sc-camp.org>

SC-Camp-Hybrid 2022

Bucaramanga, Colombia

November 24-26, 2022

+

Guadalajara, México

December 8-10, 2022

+

Online



Xavier Besseron

LuXDEM Research Team
University of Luxembourg

<https://luxdem.uni.lu>



Outline

Introduction to High-Performance Computing

- Motivations
- Parallelization Approaches
- Memory Models and Programming Models
- Parallel Programming Caveats
- Performance Modeling and Analysis

HPC for the Simulation of Particles

- Discrete Element Method and XDEM
- Domain Decomposition and Load-Balancing
- Fine Grain Parallelization with OpenMP
- Faster Broad-Phase with Roofline Analysis
- Verlet Buffer approach for Collision Detection

Going further: DEM+CFD

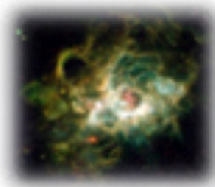
- Parallel Multi-Physics Simulation of a Biomass Furnace

Introduction to High-Performance Computing

Motivations

Computer Simulation is everywhere

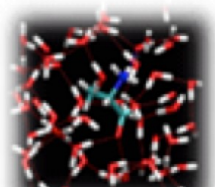
Electro-
Magnetics



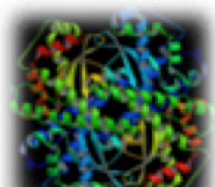
Computational Chemistry
Quantum Mechanics



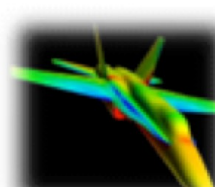
Computational Chemistry
Molecular Dynamics



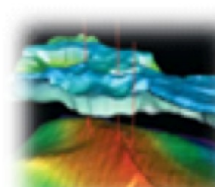
Computational
Biology



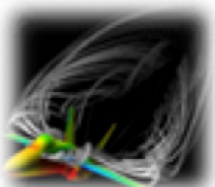
Structural Mechanics
Implicit



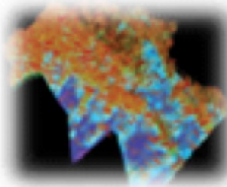
Seismic
Processing



Computational Fluid
Dynamics



Reservoir
Simulation



Rendering
Ray Tracing



Climate / Weather
Ocean Simulation



Data Analytics



Structural Mechanics
Explicit



- Computational Fluid Dynamics ([OpenFOAM](#))
- Finite Element Analysis ([Abaqus](#))
- Climate / Weather / Ocean Simulation ([WRF](#))
- Molecular Dynamics ([Gromacs](#), [Amber](#))
- Quantum Chemistry ([Quantum Espresso](#))
- Visualization ([Paraview](#))
- Data processing ([R](#), [Matlab](#))
- ...

What is High Performance Computing?

High Performance Computing (HPC)

- Use of parallel and distributed computers with fast interconnects
- To execute an application quickly and efficiently

Why parallel computers?

- Performance of single CPU core is getting limited (power, physics)
- Multiple cores are used to increase the computing capacity

HPC is challenging

- Active research domain
- Provides tools for many other researchers

How to get faster with HPC?

Build faster processor

- Moore's law continues but ***The free lunch is over!***
- CPU serial-processing speed is reaching its physical limit
- **Multi-cores processor architectures**
- **Accelerators and specialized processors** (GPU, TPU, FPGA, etc.)

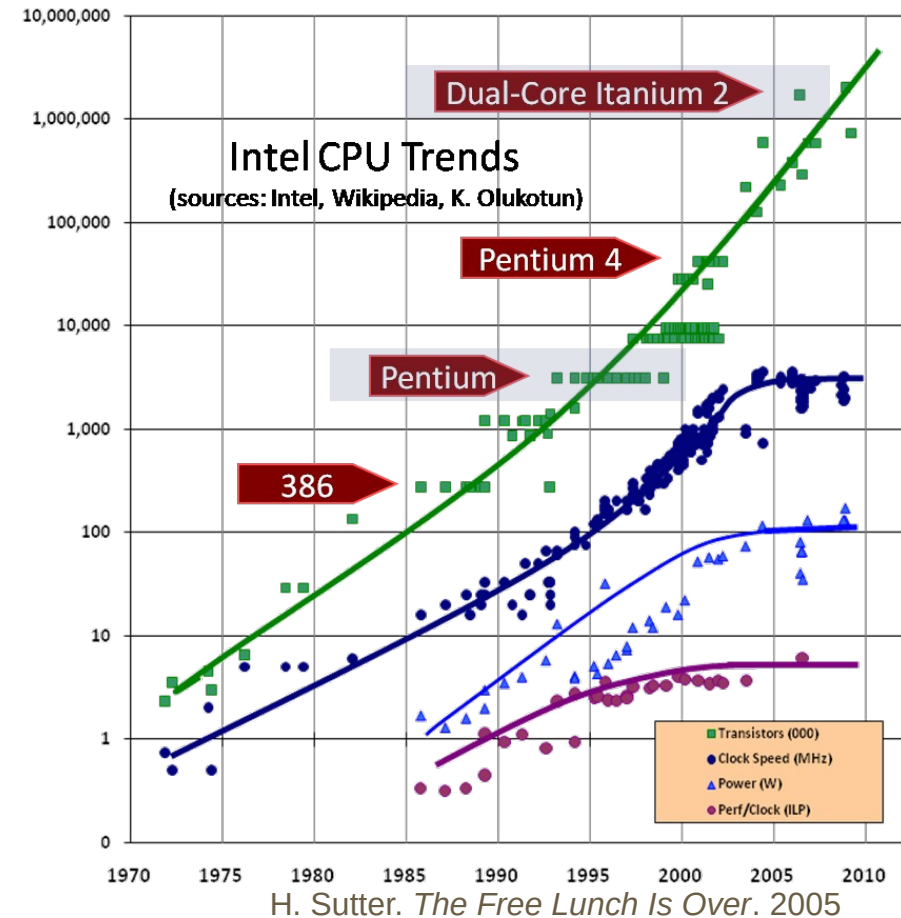
Combine multiple computers

- **HPC Clusters and Supercomputers**

Better use of the hardware

- **Identify the actual bottleneck** (CPU, memory, network, etc.)
- **Vectorization (SIMD)**

Not to forget: **Better algorithms**



How to get faster with HPC?

Build faster processor

- Moore's law continues but ***The free lunch is over!***
- CPU serial-processing speed is reaching its physical limit
- **Multi-cores processor architectures**
- **Accelerators and specialized processors** (GPU, TPU, FPGA, etc.)

Combine multiple computers

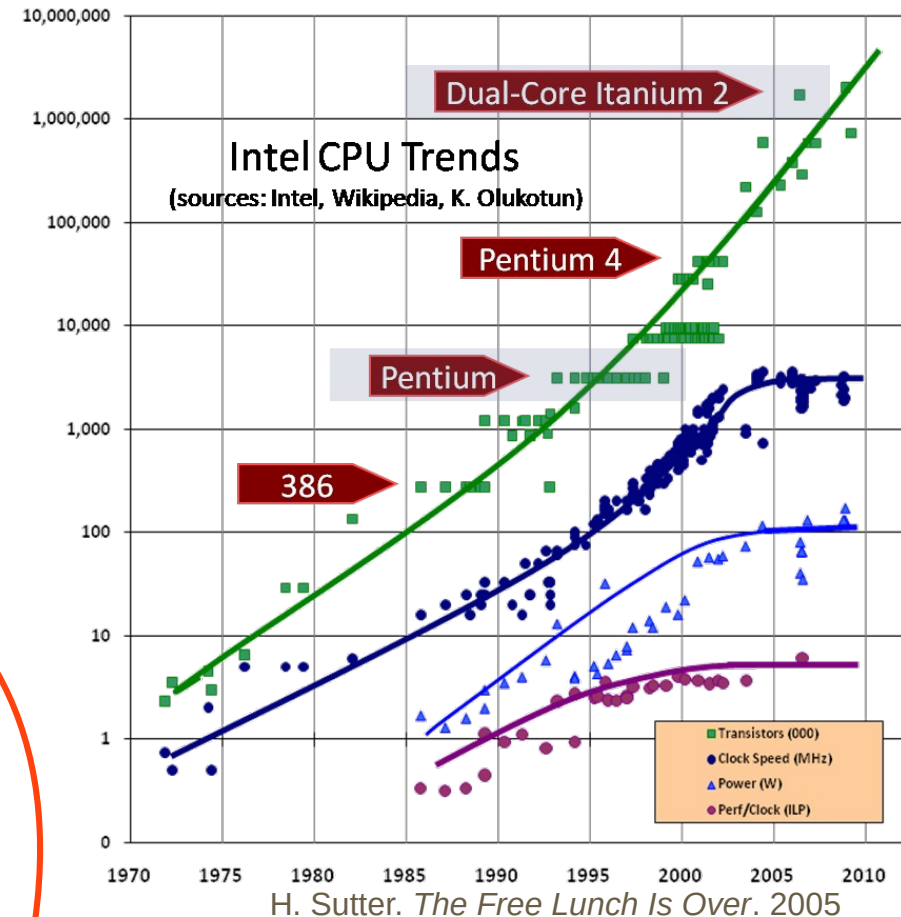
- **HPC Clusters and Supercomputers**

Better use of the hardware

- **Identify the actual bottleneck** (CPU, memory, network, etc.)
- **Vectorization (SIMD)**

Not to forget: **Better algorithms**

Parallel programming



How much faster is HPC?

Your simulation is limited by the performance of you computer



Your laptop



Uni.lu HPC*



* shared with other users

	Your laptop	Uni.lu HPC*	602,112 cores Frontier*
CPU	4 cores	46,528 cores	602,112 cores
Memory	16 GB	130 TB	9.2 PB
Storage	1 TB	3.48 PB	700 PB
Network	Ethernet 10 Gb/s	Infiniband 100 Gb/s	Slingshot 100 GB/s
Accelerators	1 GPU	96 GPUs	37,632 GPUs
R_{peak}	350 Gflops	1,847 Tflops	1,686 Pflops

→ HPC provides the **methodology** and **tools** for your application to run faster



Introduction to High-Performance Computing

Parallelization Approaches

How to parallelize an algorithm?

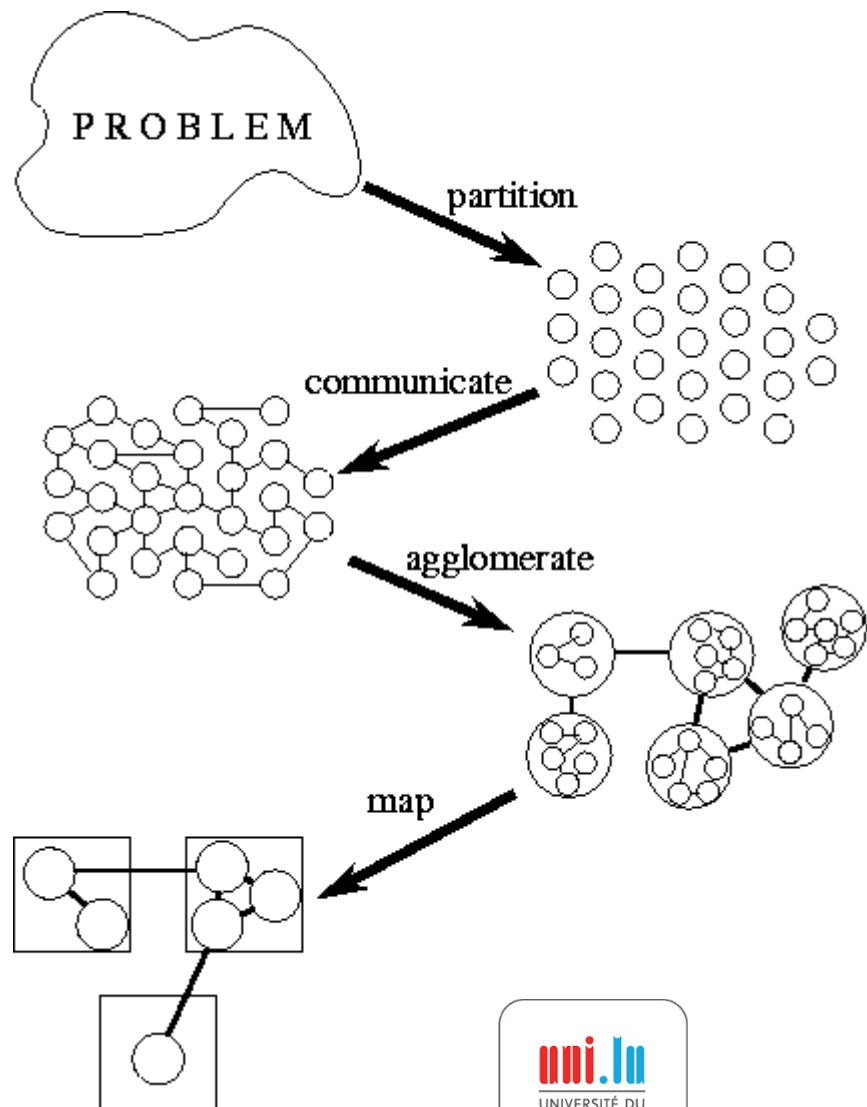
Designing and Building Parallel Programs,
by Ian Foster, 1995.

Partitioning: decompose computation in small tasks, independently of the number of processors

Communication: identify coordination and dependencies between tasks

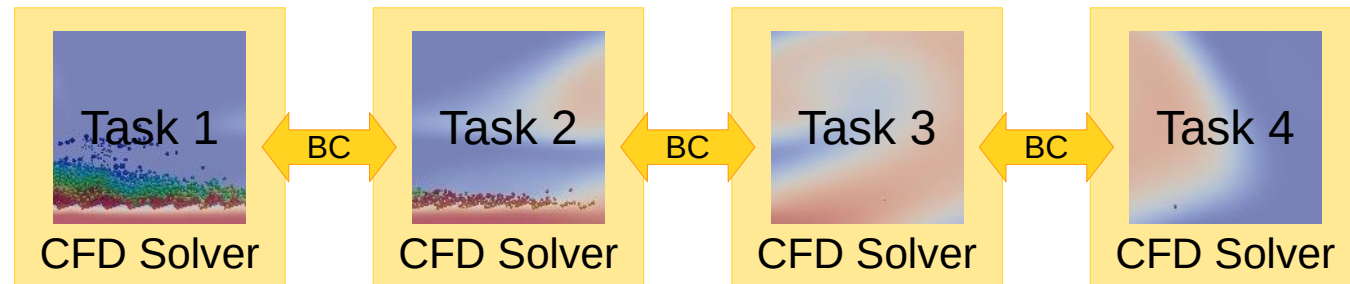
Agglomeration: tasks are combined into larger tasks to improve performance or to reduce development costs

Mapping: Assign tasks to processors in order to maximize processor utilization and minimize communication costs
→ load-balancing algorithms



Problem Partitioning → Domain Decomposition

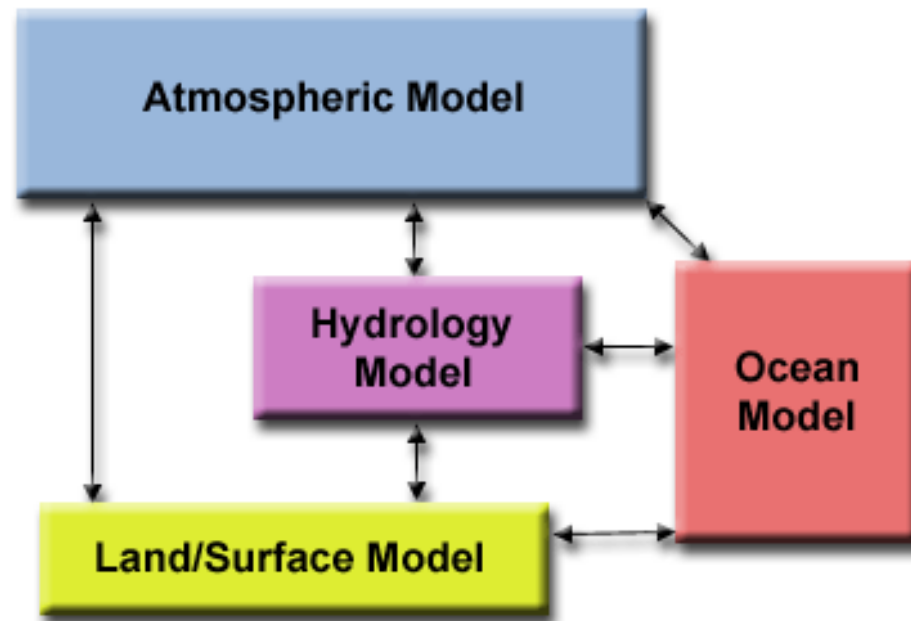
- The data associated with the problem is decomposed
- Each parallel task works on a portion of the data
- The same program is used to process each piece of data
- Communication may be needed between tasks



→ This is called **SPMD** for **Single Program, Multiple Data**

Problem Partitioning → Functional Decomposition

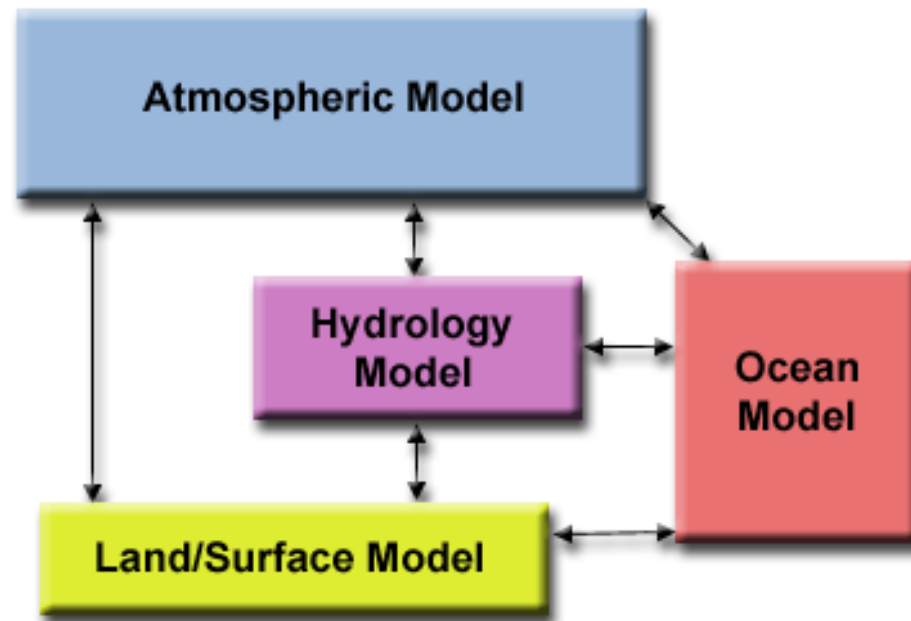
- Focus on the performed computation rather than on the data
- Problem decomposed according to the work to be done
- Each task then performs a portion of the overall work
- Communication may be needed between tasks



→ This is called **MPMD** for **Multiple Program, Multiple Data**

Problem Partitioning → Functional Decomposition

- Focus on the performed computation rather than on the data
- Problem decomposed according to the work to be done
- Each task then performs a portion of the overall work
- Communication may be needed between tasks



→ This is called **MPMD** for **Multiple Program, Multiple Data**

Complex applications might use an hybrid approach between Domain Decomposition and Functional Decomposition!

Introduction to High-Performance Computing

Memory Models and Programming Models

Thread vs Process

At the level of the Operating System

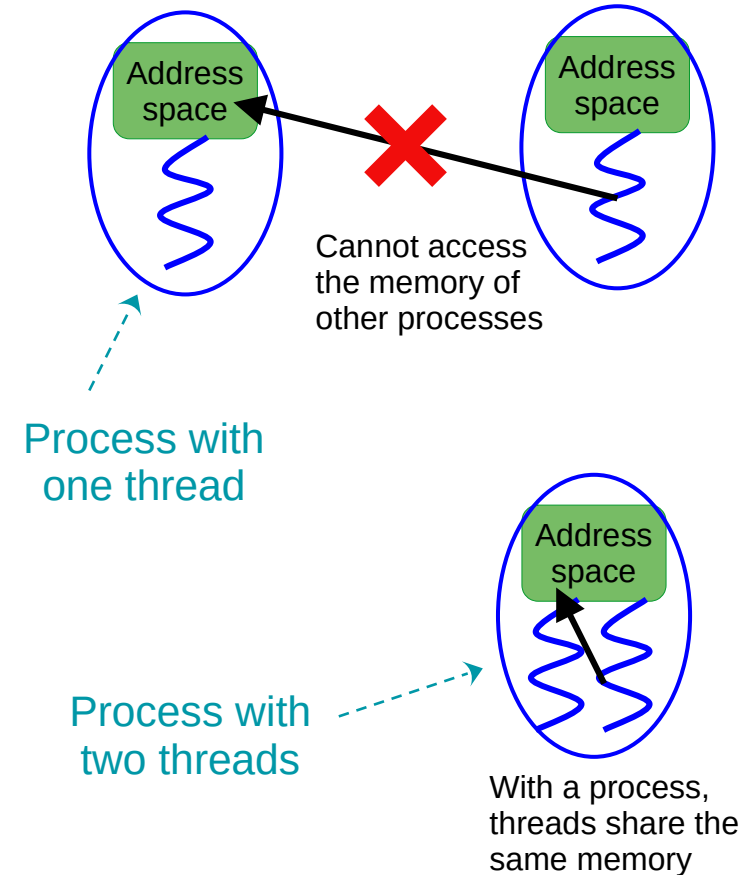
- Processes and Threads are two ways to exploit parallelism i.e. execute code on different cores at the same time
- There can be more processes/threads than CPU cores, but for HPC purpose, we usually use one threads per core

Processes ~ program

- Have their own address space (memory with variables)
- The process address space is not accessible to other processes
- Contain at least one thread

Threads ~ execution flow

- Use the address space of the process
- Threads within one process share the same address space
- Lightweight ~ Faster to create and destroy than processes



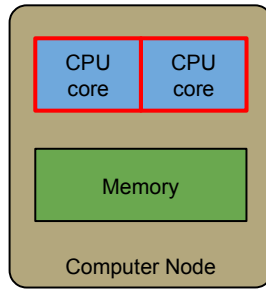
Memory Models for Parallel Programming

Shared Memory
Single Computing Node

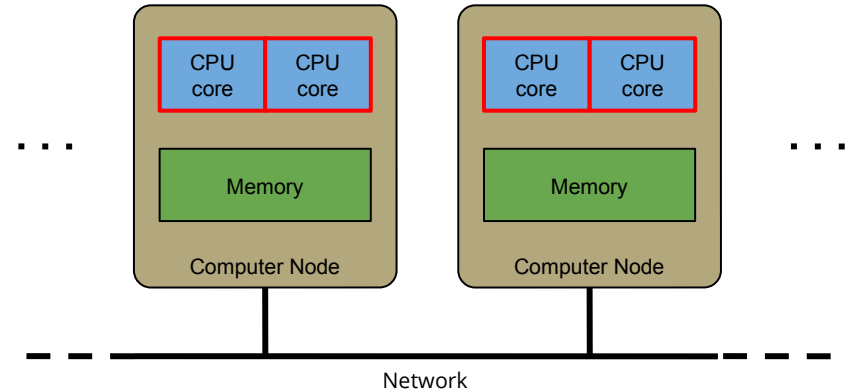
VS

Distributed Memory
Multiple Computing Nodes

*That's your
laptop or
workstation!*



*That's an
HPC cluster!*



Memory Models for Parallel Programming

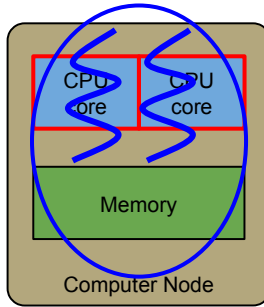
Shared Memory Single Computing Node

VS

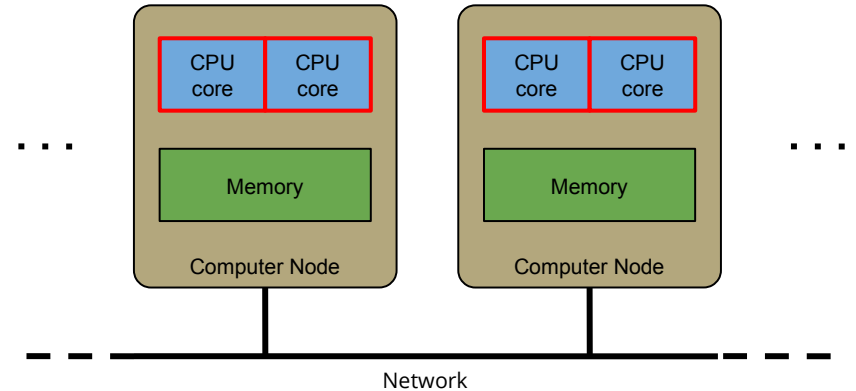
Distributed Memory Multiple Computing Nodes

*That's your
laptop or
workstation!*

One process with
multiple threads



*That's an
HPC cluster!*



To use multiple CPUs on the same computing node

- Distribute the **computation**
- All threads share the same memory space
- Require synchronizations instead of communications

⇒ **OpenMP**: Open Multi-Processing



Memory Models for Parallel Programming

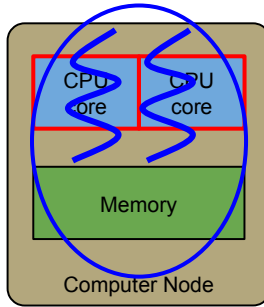
Shared Memory Single Computing Node

VS

Distributed Memory Multiple Computing Nodes

*That's your
laptop or
workstation!*

One process with
multiple threads

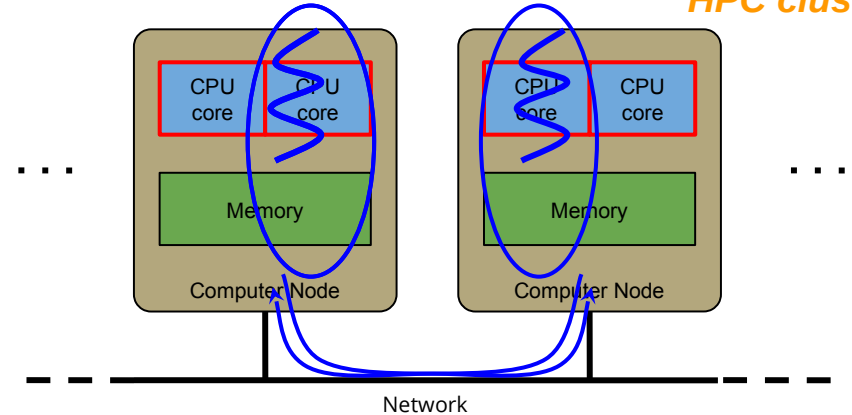


- To use multiple CPUs on the same computing node
- Distribute the **computation**
 - All threads share the same memory space
 - Require synchronizations instead of communications

⇒ **OpenMP**: Open Multi-Processing

*That's an
HPC cluster!*

One process per node with
communication



- To use multiples CPUs on multiple computing nodes
- Distribute the **computation** and the **data**
 - Processes cannot access the memory of others
 - Exchange messages on the network

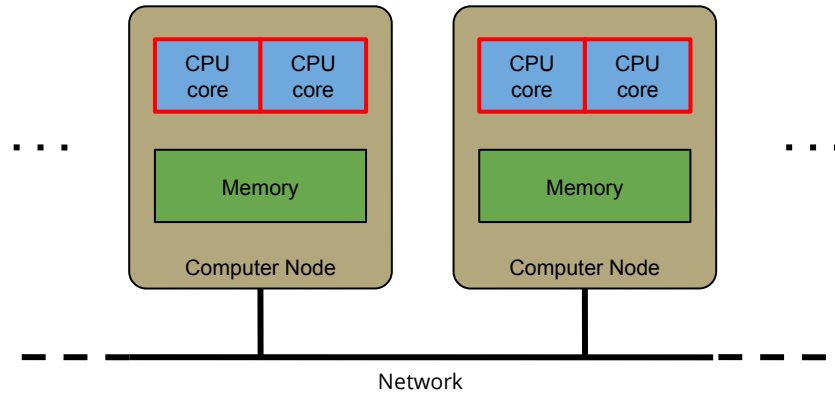
⇒ **MPI**: Message Passing Interface



Memory Models for Parallel Programming

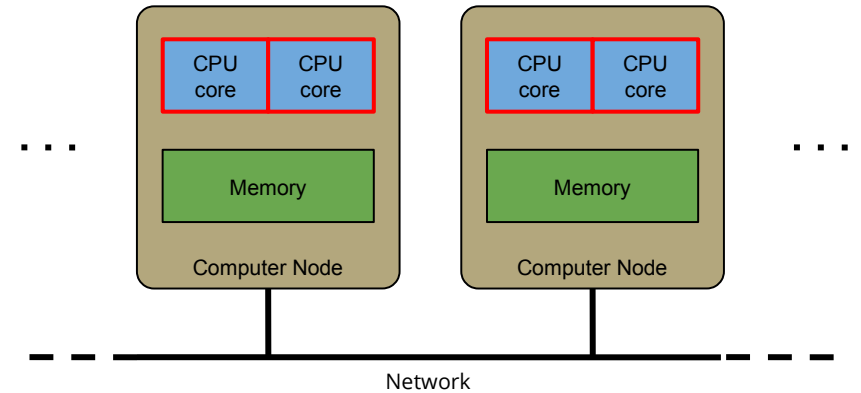
Only Distributed Memory

All cores on Multiple Computing Nodes



Hybrid Shared + Distributed Memory

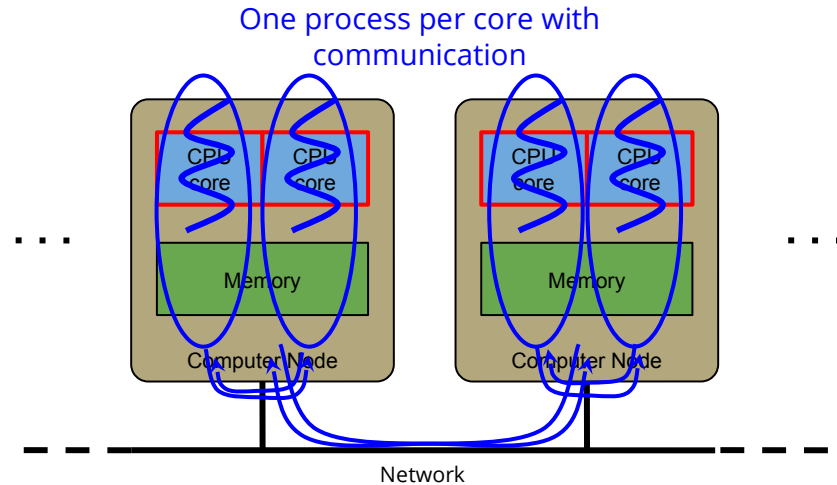
All cores on Multiple Computing Nodes



Memory Models for Parallel Programming

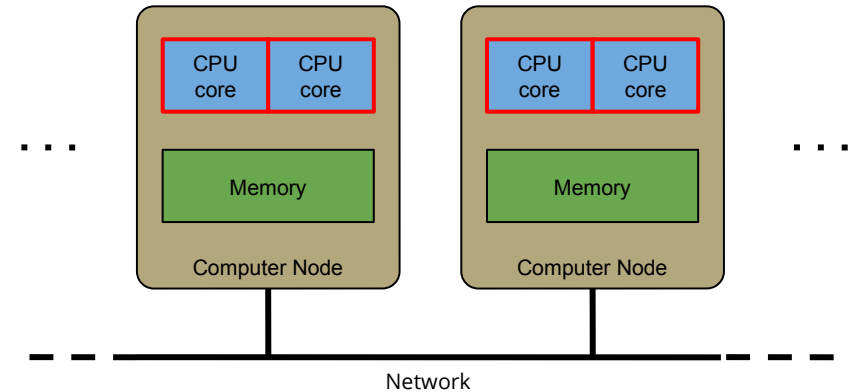
Only Distributed Memory

All cores on Multiple Computing Nodes



Hybrid Shared + Distributed Memory

All cores on Multiple Computing Nodes



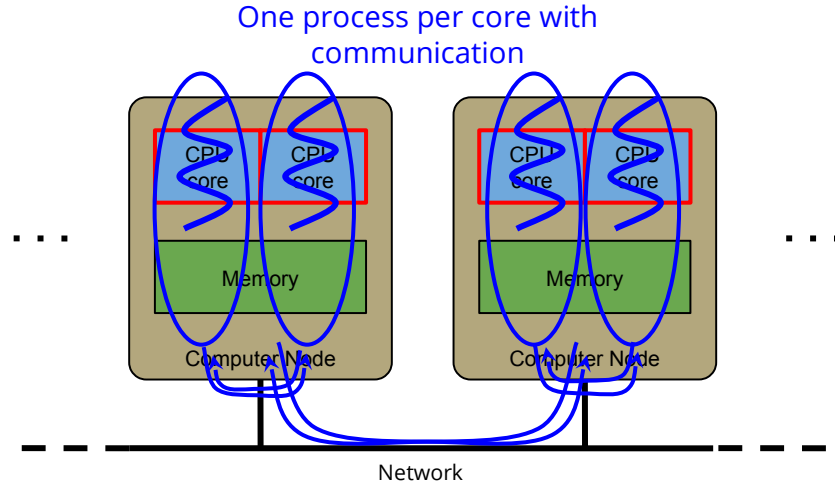
The processes cannot access the memory of others

- Use communication even within a node
- Communication within a node can be optimized by the software layer (e.g. memory copy instead to bypass the network)
- Simplify the programming ⇒ **MPI**

Memory Models for Parallel Programming

Only Distributed Memory

All cores on Multiple Computing Nodes

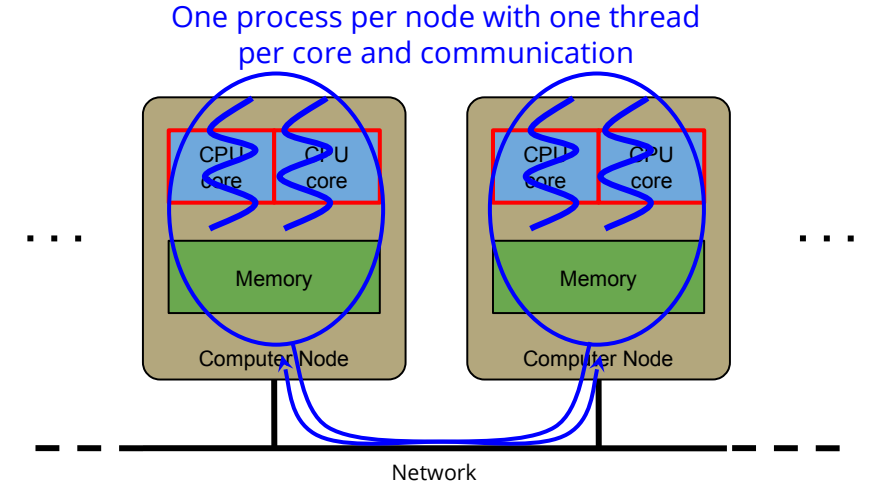


The processes cannot access the memory of others

- Use communication even within a node
- Communication within a node can be optimized by the software layer (e.g. memory copy instead to bypass the network)
- Simplify the programming ⇒ **MPI**

Hybrid Shared + Distributed Memory

All cores on Multiple Computing Nodes



Use shared memory within a computing node and distributed memory across nodes

- To be adapted to the hardware
- Benefit of both models, but more complex

⇒ **Hybrid MPI + OpenMP**

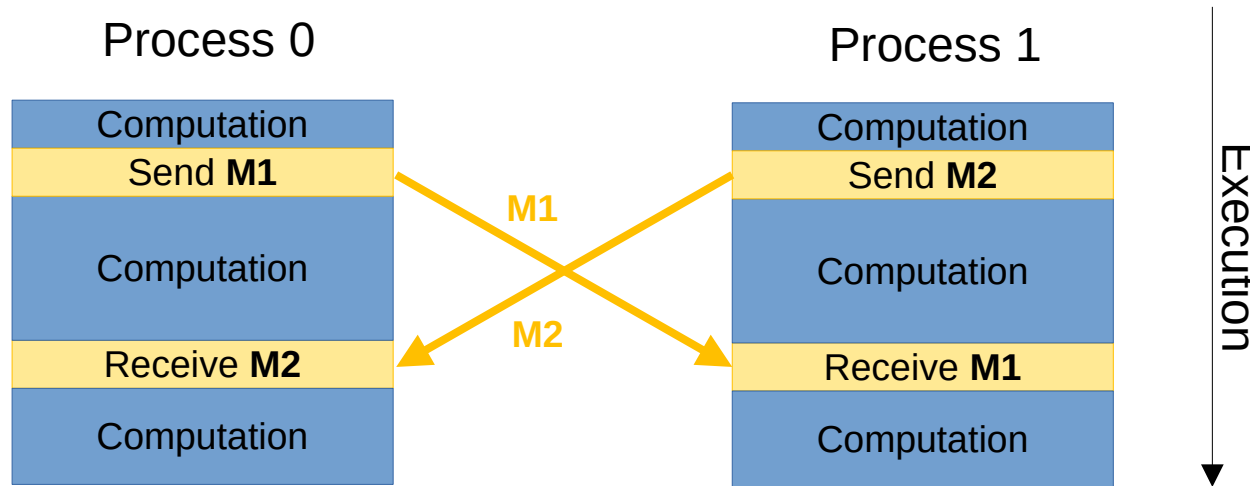


Distributed Memory Programming with MPI



Message Passing Model: Multiple processes run in parallel and exchange messages

→ Analogy: **Paper mails** if your network is slow, **E-mails** if your network is fast



- **MPI is a standard:** MPI-1.0 in 1994, MPI-2.0 in 1997, MPI-3.0 in 2012, MPI-4.0 in 2021
- Different implementations: OpenMPI, MPICH, MVAPICH, Intel MPI, etc.
- Standard API in C and Fortran, non-official API in C++, Python



MPI Concepts

Fixed number of processes

- Specified at application startup, unchanged throughout execution

Communicator

- Abstraction for a group of processes that can communicate
- A process can belong to multiple communicators
- Default and global communicator: `MPI_COMM_WORLD`

Process Rank

- Index of a process within a communicator
- Used to identify other processes in communication operations

MPI Programming Interface

Lifecycle management

- `MPI_Init`, `MPI_Finalize`,
`MPI_Abort`

Communicators

- `MPI_Comm_Size`, `MPI_Comm_Rank`
- `MPI_Comm_create`, `MPI_Comm_dup`,
`MPI_Comm_join`

Datatype and Buffer

- `MPI_Type_*`
- `MPI_Pack`, `MPI_Unpack`

Blocking point-to-point

- `MPI_Send`, `MPI_Recv`

Non-blocking communications

- `MPI_Isend`, `MPI_Irecv`
- `MPI_Wait`, `MPI_Waitall`

Collective communications

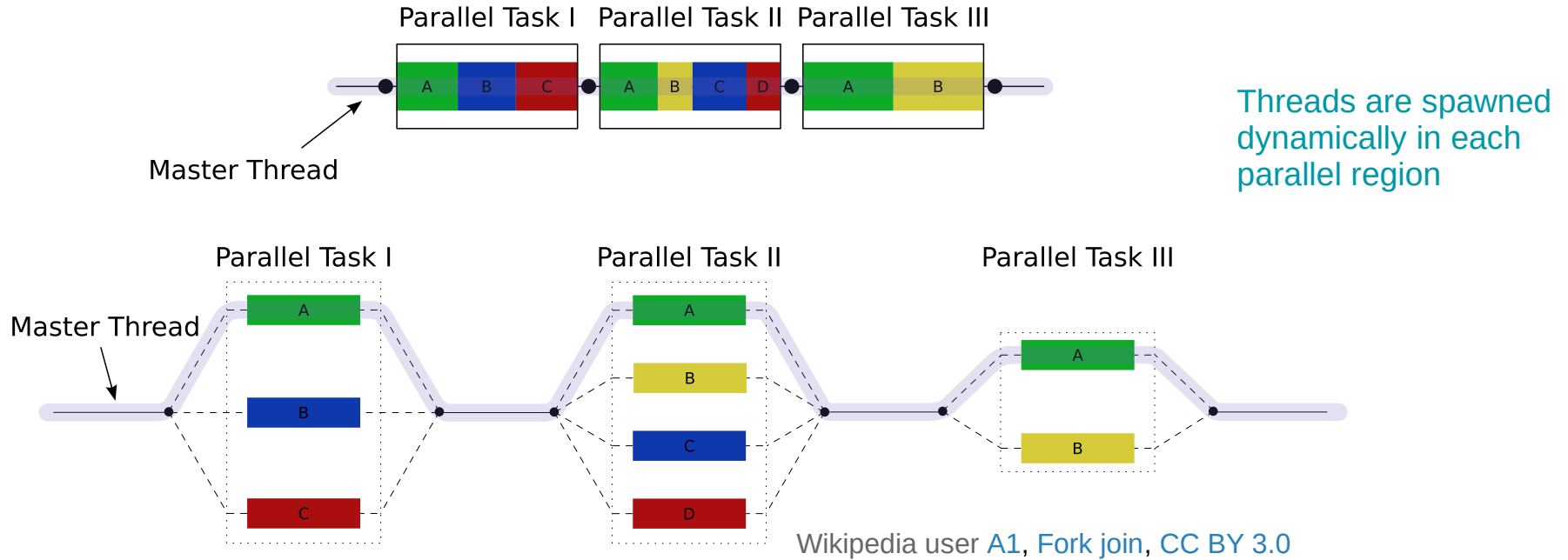
- `MPI_Bcast`, `MPI_Reduce`,
`MPI_Gather`, `MPI_Scatter`
- `MPI_Barrier`

One-sided communications

- `MPI_Win_create`, `MPI_wait`
- `MPI_Put`, `MPI_Get`

Shared Memory Multi-Processing with OpenMP

- OpenMP is based on the **Fork-Join** model
- Analogy: **Restaurant kitchen**, the cooks share the utensils and ingredients to prepare the dishes



- Portable standard API for C, C++ and Fortran
- Support **multi-cores** and **accelerators**

OpenMP Concepts

Based on compiler directives `#pragma omp ...`

Example

```
#pragma omp parallel for
for (int i = 0; i < 100000; i++) {
    a[i] = 2 * i;
}
```

- Can control work distribution with the **schedule** clause (static, dynamic, guided)
- Threads can share variables, cf **private** or **shared** clauses
→ **Caution with concurrent accesses!**

In principle → Simple to use, minor modifications to the code

In practice → Might require changes in loops and data structures

Introduction to High-Performance Computing

Parallel Programming Caveats

Race Condition 1/3

"Debugging programs containing race conditions is no fun at all."
Andrew S. Tanenbaum, *Modern Operating Systems*, 1992.

Race condition

- A **timing-dependent** error involving shared state
- It runs fine most of the time, and from time to time, something weird and unexplained appears

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    account->balance += amount;
}
```

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)

READ balance (0)

ADD 10

WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)

ADD 1000

WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

Race Condition 2/3

Code example

```
void deposit(Account* account, double amount)
{
    READ balance
    ADD amount
    WRITE balance
}
```

Concurrent execution

Thread 1 calls deposit(A, 10)
READ balance (0)

ADD 10
WRITE balance (10)

Thread 2 calls deposit(A, 1000)

READ balance (0)
ADD 1000
WRITE balance (1000)

→ **Result: balance is 10 instead of 1010**

Without protection, any interleave combination is possible!

Race Condition 3/3

Different kind of race conditions

- **Data race:** Concurrent accesses to a shared variable
- **Atomicity bugs:** Code does not enforce the atomicity for a group of memory accesses, e.g. *Time of check to time of use*
- **Order bugs:** Operations are not executed in order
Compilers and processors can actually re-order instructions

What to do?

- Protect critical sections: **Mutexes**, **Semaphores**, etc.
- Use atomic instructions and memory barriers (low level)
- Use compiler builtin for atomic operations (higher level)

Deadlock 1/3



Deadlock, photograph by David Maitland

"I would love to have seen them go their separate ways, but I was exhausted. The frog was all the time trying to pull the snake off, but the snake just wouldn't let go."

Deadlock 2/3

Code Example

```
void deposit(Account* account,
             double amount)
{
    lock(account->mutex);
    account->balance += amount;
    unlock(account->mutex);
}
```

```
void transfer(Account* accA,
              Account* accB,
              double amount)
{
    lock(accA->mutex);
    lock(accB->mutex);
    accA->balance += amount;
    accB->balance -= amount;
    unlock(accA->mutex);
    unlock(accB->mutex);
}
```

→ Use mutexes (lock/unlock) to protect concurrent accesses?

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A, B, 10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

Thread 2 calls `transfer(B, A, 20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A, B, 10)`

```
lock(A->mutex) ;  
  
lock(B->mutex) ; // wait until  
                  // B is unlocked  
  
...
```

Thread 2 calls `transfer(B, A, 20)`

```
lock(B->mutex) ;  
  
lock(A->mutex) ; // wait until  
                  // A is unlocked  
  
...
```

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A, B, 10)`

```
lock(A->mutex) ;  
  
lock(B->mutex) ; // wait until  
                  // B is unlocked  
  
...
```

Thread 2 calls `transfer(B, A, 20)`

```
lock(B->mutex) ;  
  
lock(A->mutex) ; // wait until  
                  // A is unlocked  
  
...
```

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A,B,10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

Thread 2 calls `transfer(B,A,20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A, B, 10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

Thread 2 calls `transfer(B, A, 20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A,B,10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

Thread 2 calls `transfer(B,A,20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A,B,10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

Thread 2 calls `transfer(B,A,20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A,B,10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

Thread 2 calls `transfer(B,A,20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A, B, 10)`

```
lock(A->mutex);
```

```
lock(B->mutex); // wait until  
                // B is unlocked
```

...

→ We have a deadlock!

Thread 2 calls `transfer(B, A, 20)`

```
lock(B->mutex);
```

```
lock(A->mutex); // wait until  
                // A is unlocked
```

...

Deadlock 3/3

Concurrent Execution

Thread 1 calls `transfer(A, B, 10)`

```
lock(A->mutex);  
  
lock(B->mutex); // wait until  
                // B is unlocked
```

...

→ We have a deadlock!

Thread 2 calls `transfer(B, A, 20)`

```
lock(B->mutex);  
  
lock(A->mutex); // wait until  
                // A is unlocked
```

...

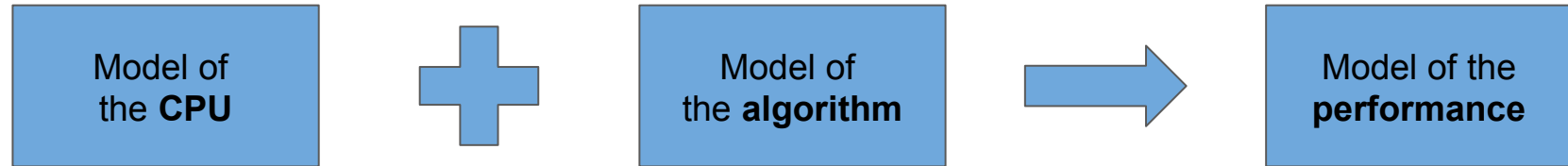
What to do?

- Think before writing multithread code
- Use high level programming model: [OpenMP](#), [Intel TBB](#), [MPI](#), etc.
- Theoretical analysis
- Software for thread safety analysis

Introduction to High-Performance Computing

Performance Modeling and Analysis

Performance Modeling of a CPU → Roofline Model



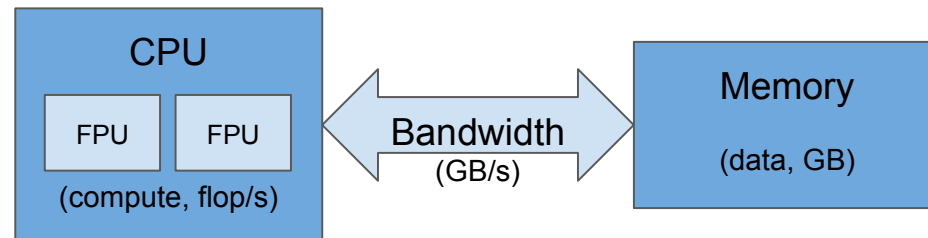
- Estimate the **performance** of an **algorithm** on a given **CPU**
 - Also applies to GPUs, TPUs, etc.
- **Throughput** oriented model
- Identify the bottleneck
- Allow to improve the implementation of an algorithm

Roofline model



Roofline model

Model of a CPU

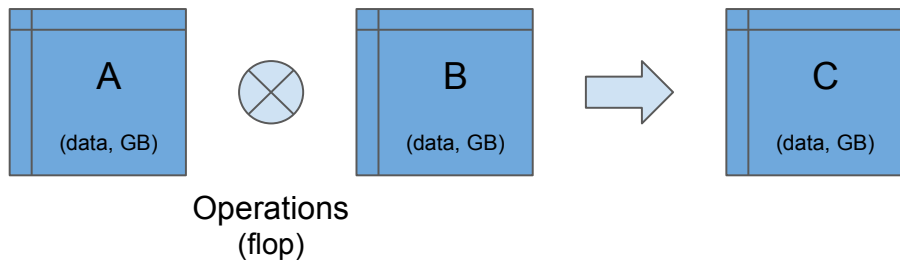


Peak performance limited by

- Compute operations: Gflop/s
- Data bandwidth: GB/s

Roofline model

Model of an algorithm



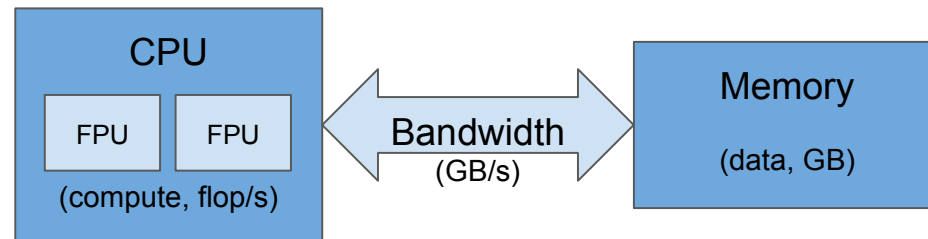
Algorithm characteristics

- Operations: Gflop
- Data: GB

Arithmetic Intensity

AI: flop / Byte

Model of a CPU

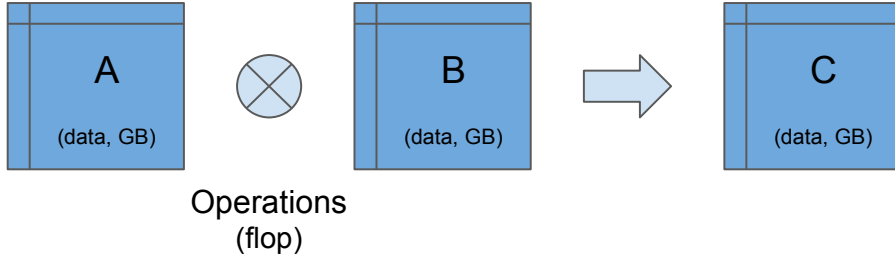


Peak performance limited by

- Compute operations: Gflop/s
- Data bandwidth: GB/s

Roofline model

Model of an algorithm



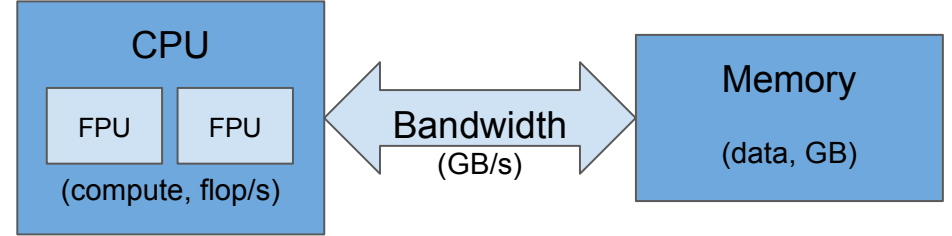
Algorithm characteristics

- Operations: Gflop
- Data: GB

Arithmetic Intensity

AI: flop / Byte

Model of a CPU



Peak performance limited by

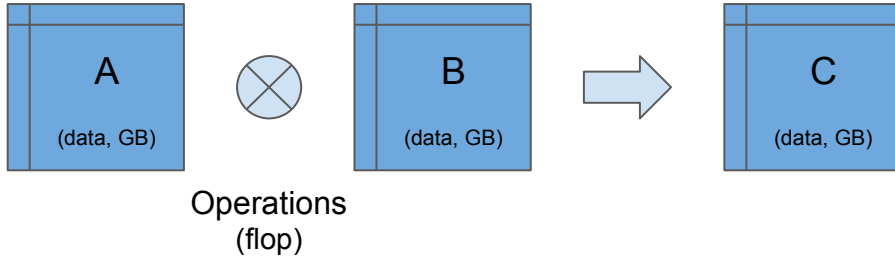
- Compute operations: Gflop/s
- Data bandwidth: GB/s

Attainable performance

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Peak Gflop/s} \\ \text{AI} \times \text{Peak GB/s} \end{array} \right.$$

Roofline model

Model of an algorithm



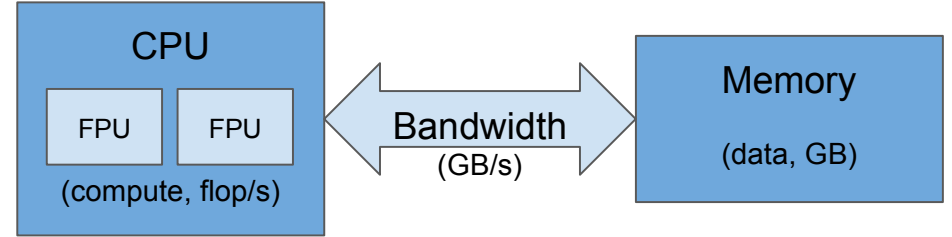
Algorithm characteristics

- Operations: Gflop
- Data: GB

Arithmetic Intensity

AI: flop / Byte

Model of a CPU



Peak performance limited by

- Compute operations: Gflop/s
- Data bandwidth: GB/s

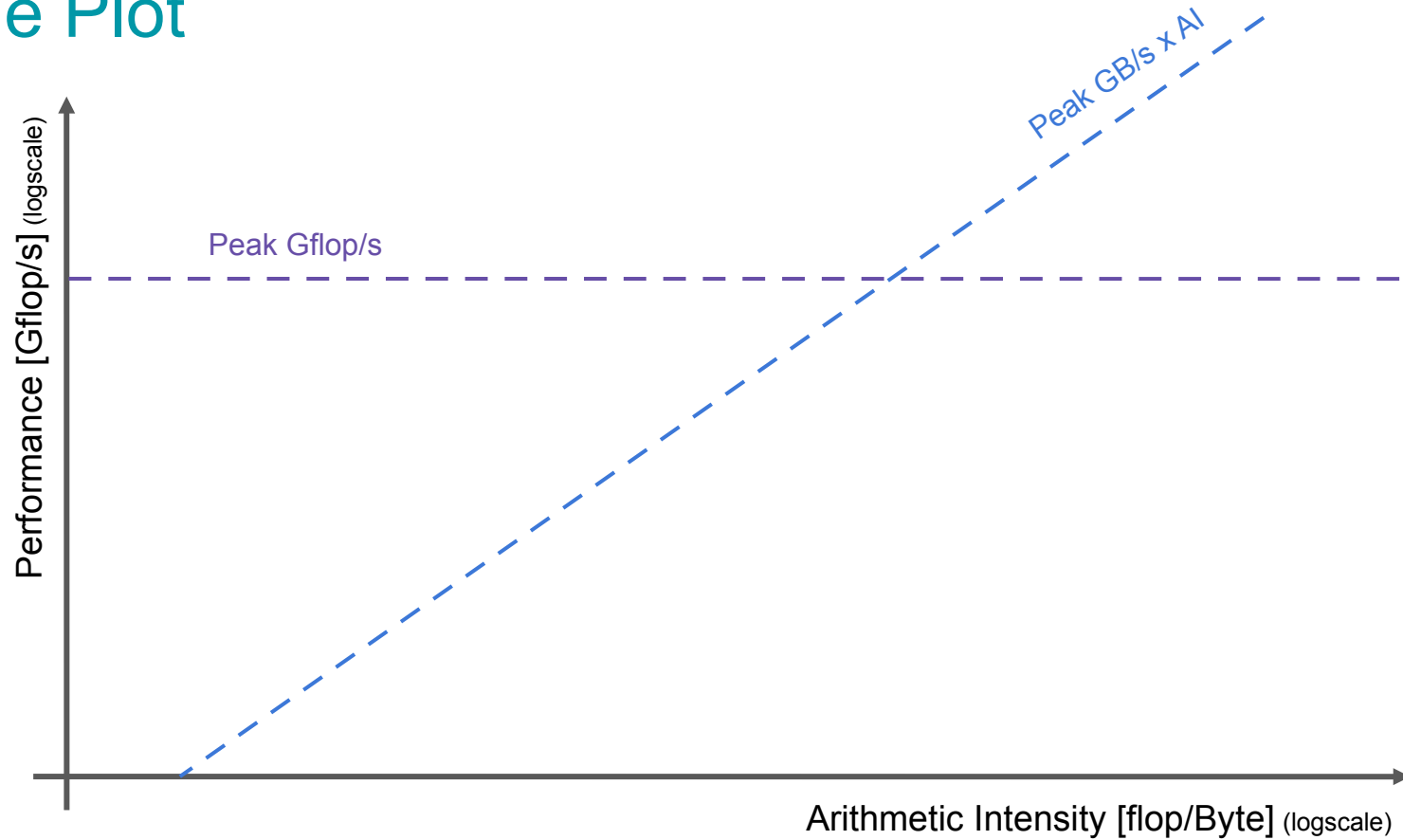
Attainable performance

Gflop/s = min

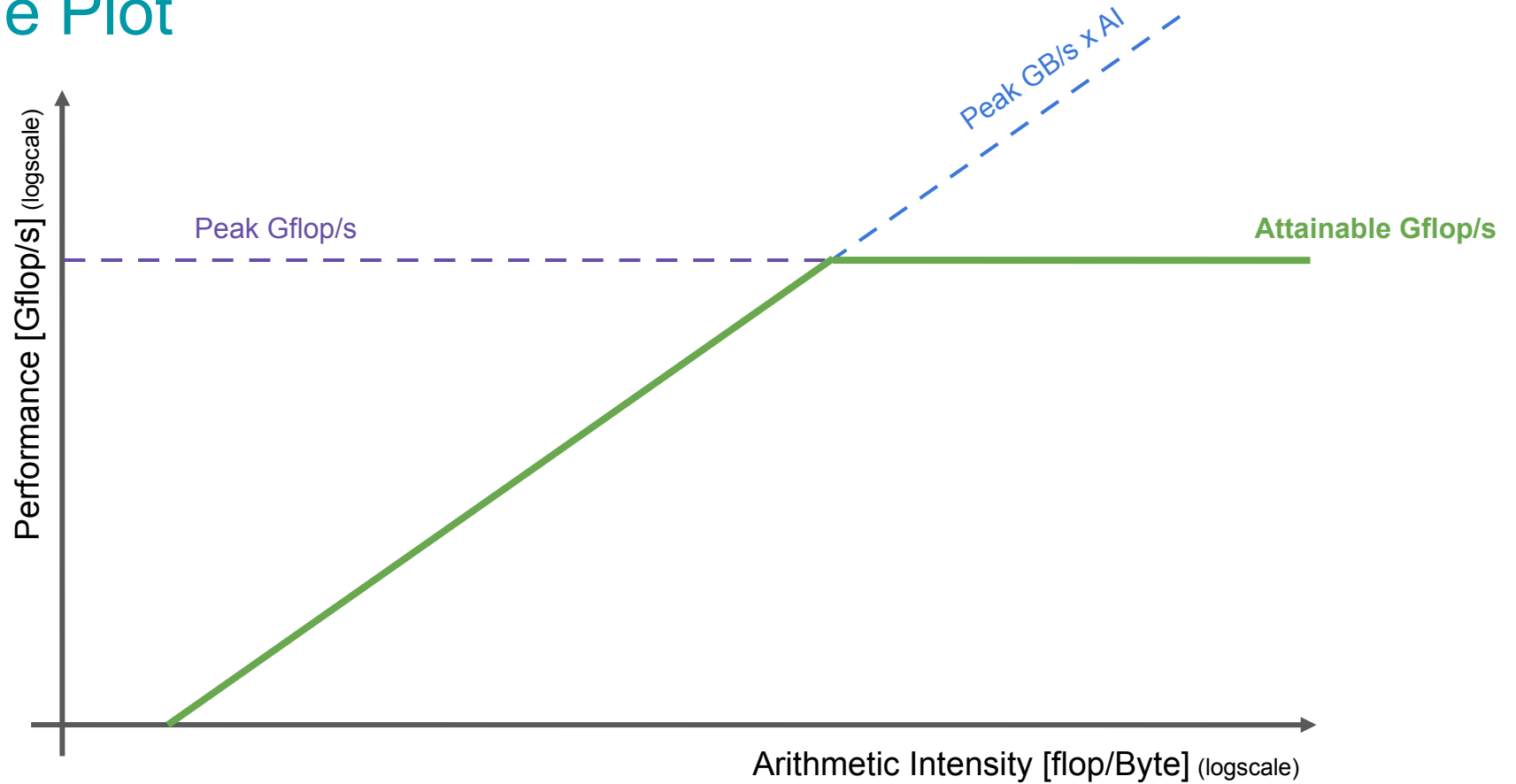
Peak Gflop/s

AI x Peak GB/s

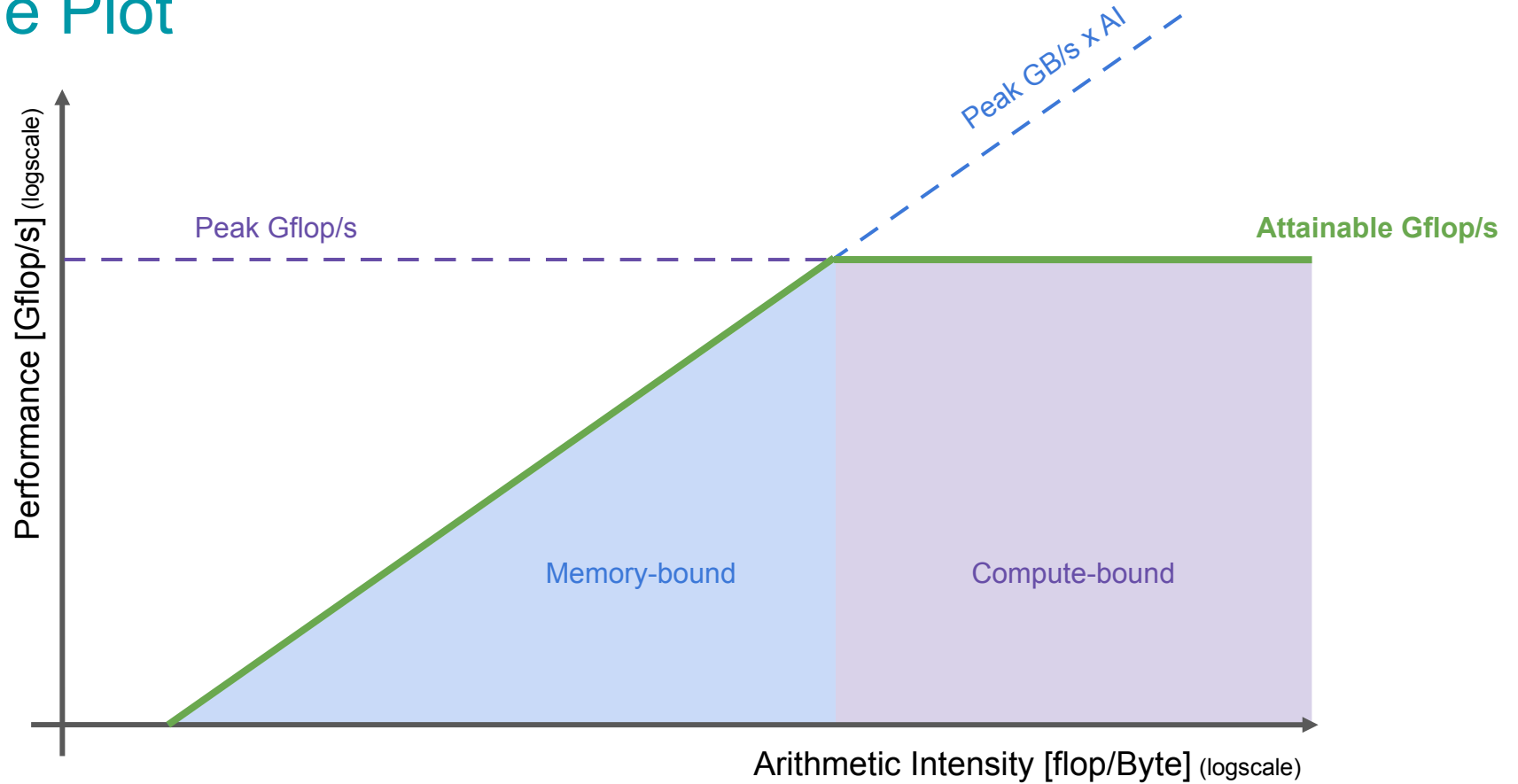
Roofline Plot



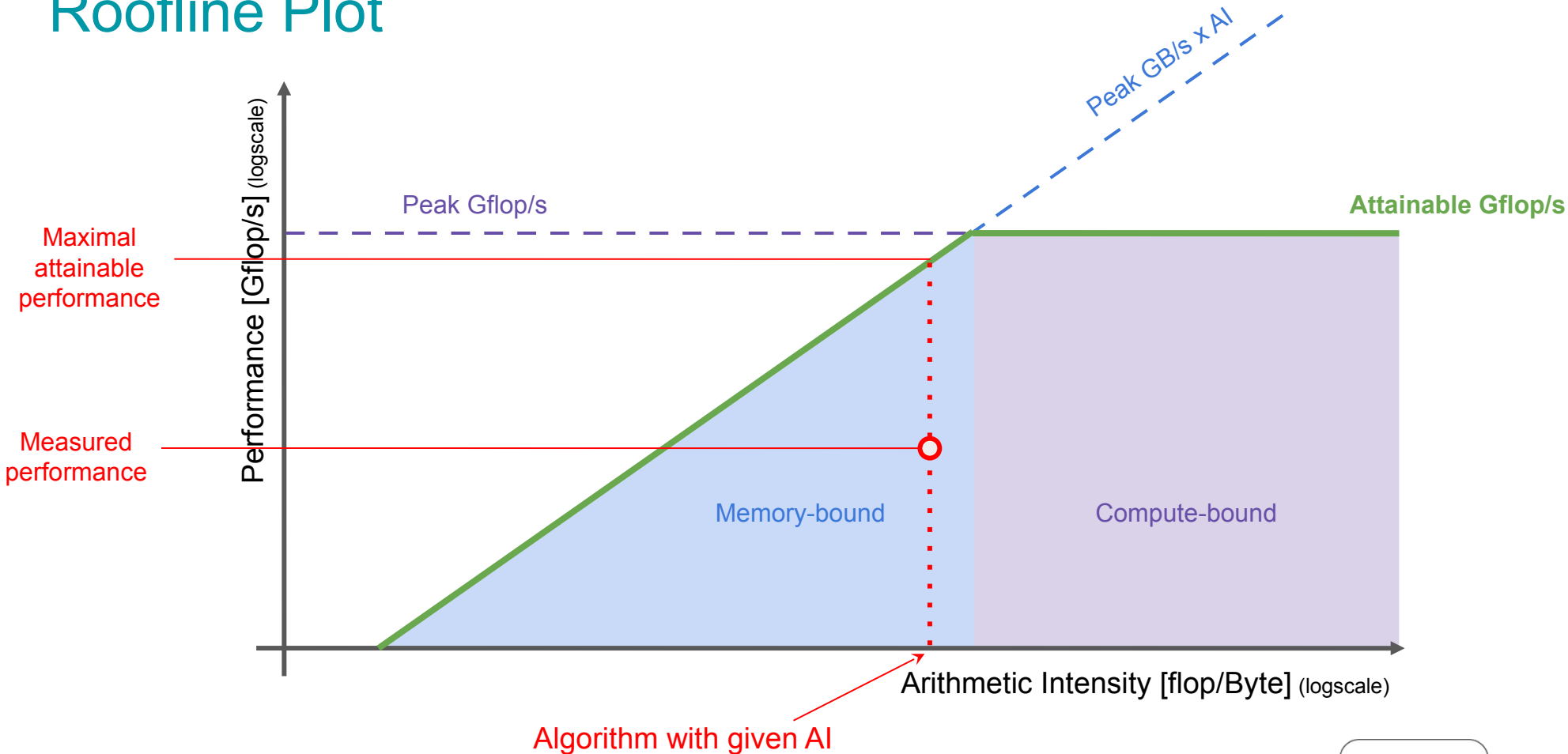
Roofline Plot



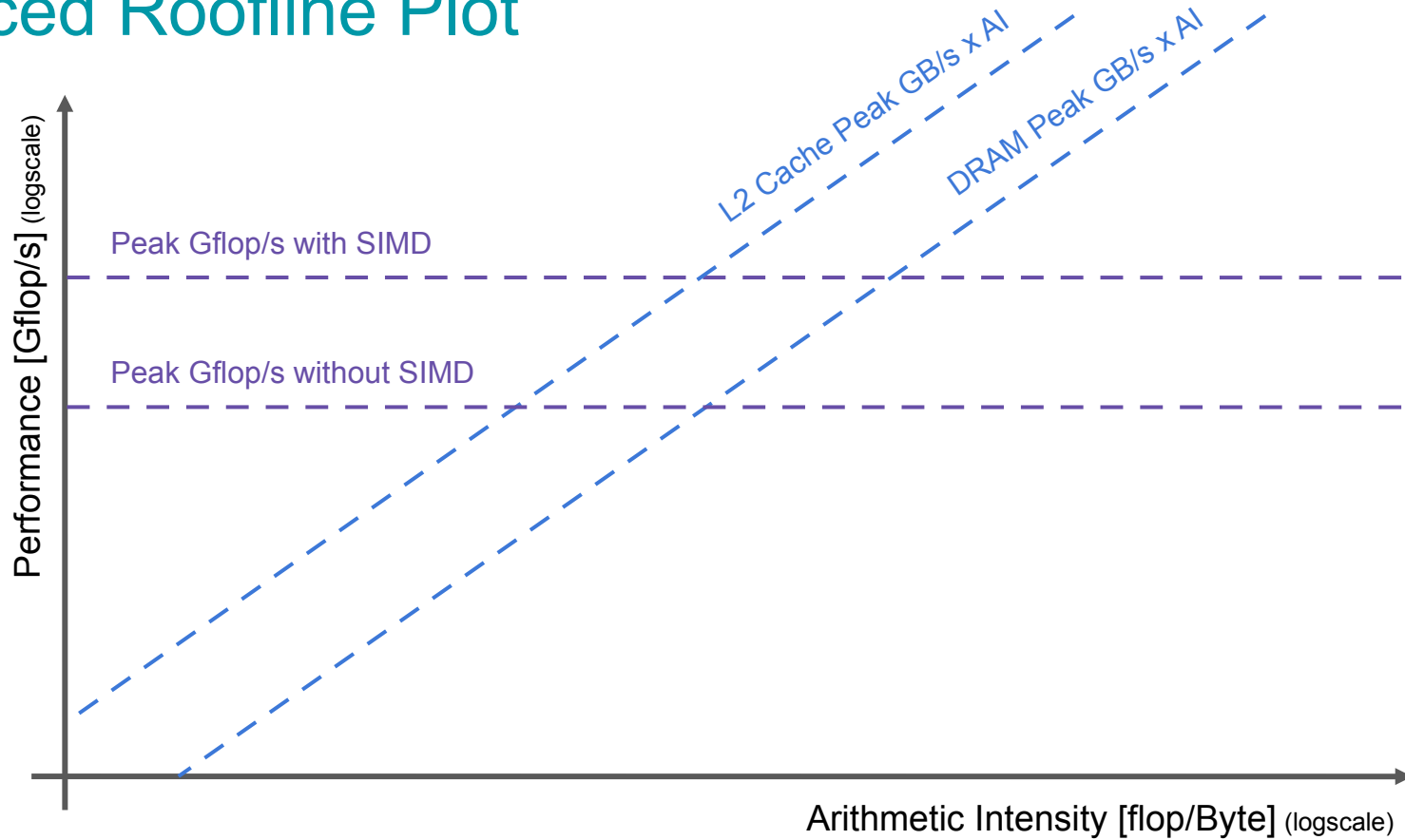
Roofline Plot



Roofline Plot



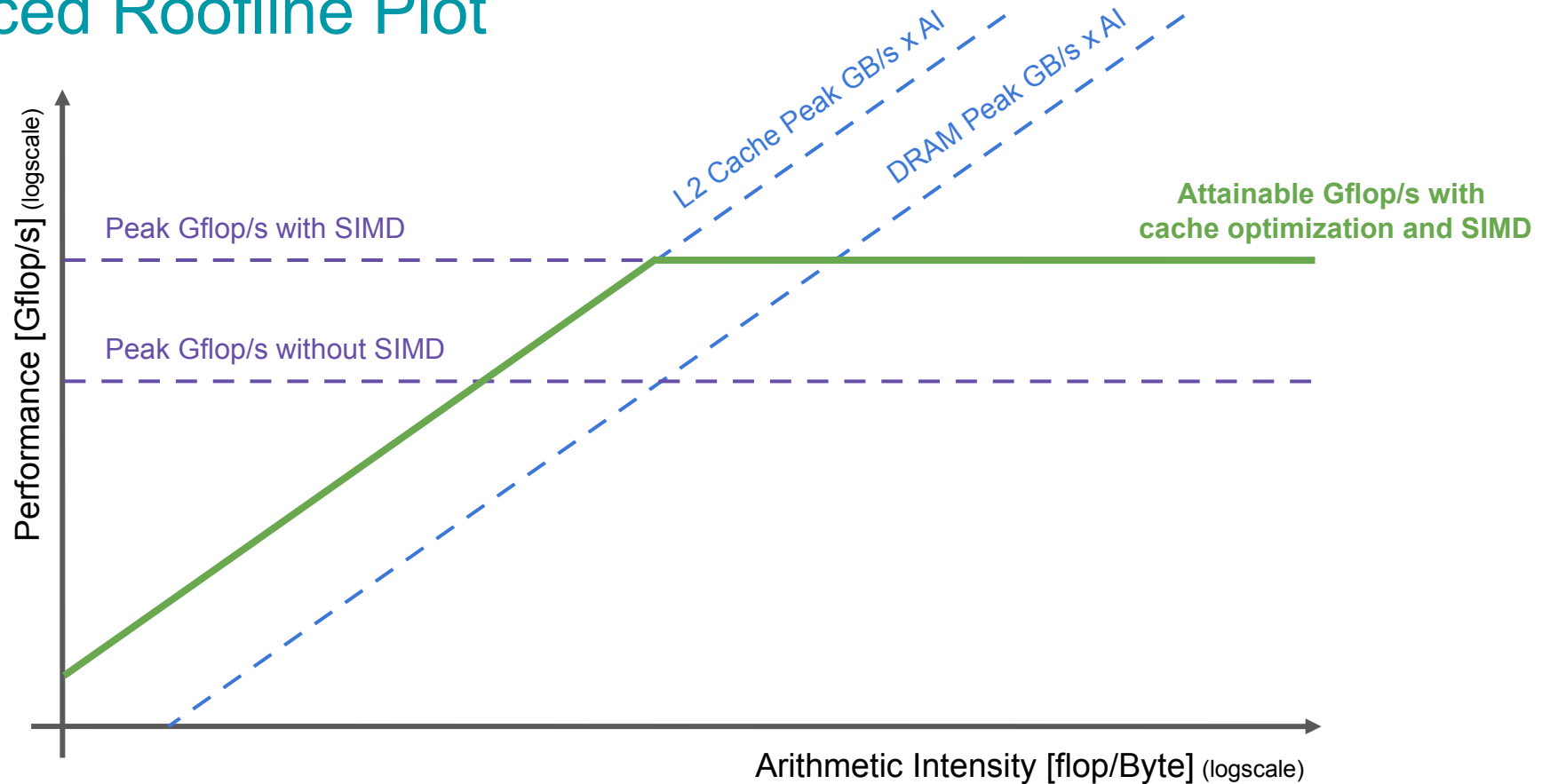
Advanced Roofline Plot



SIMD = Single Instruction, Multiple Data, ie vectorized instructions

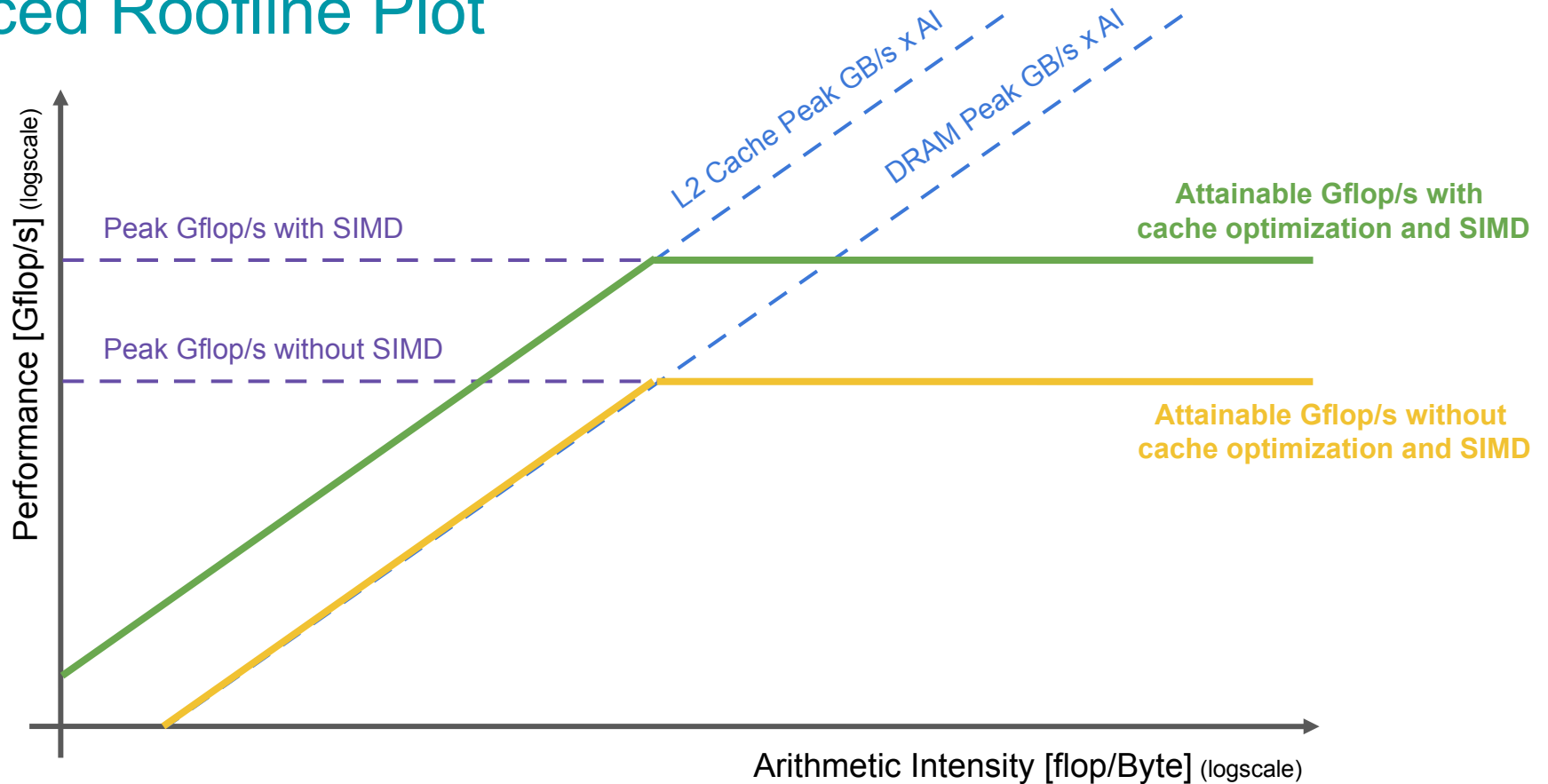


Advanced Roofline Plot



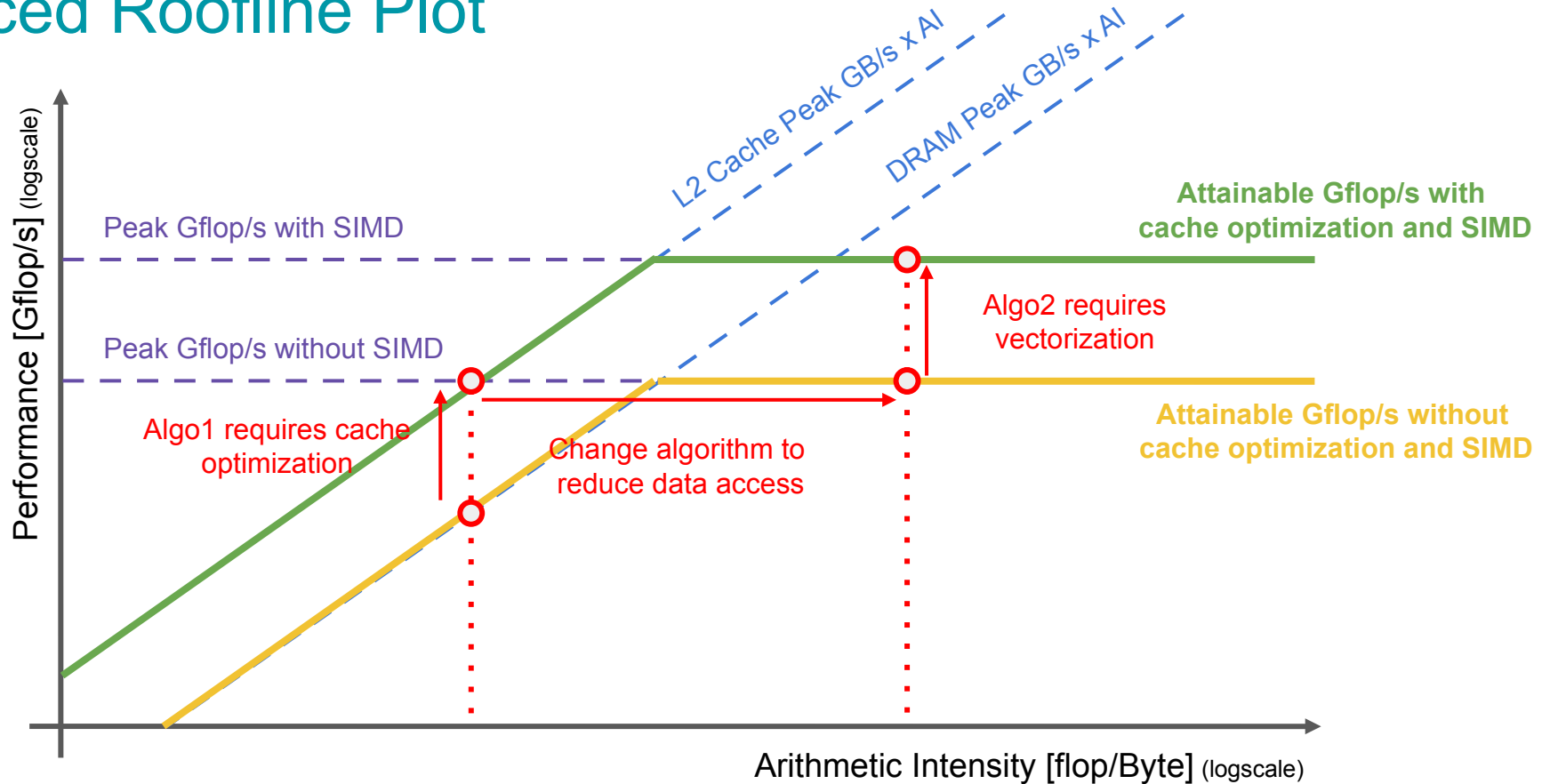
SIMD = Single Instruction, Multiple Data, ie vectorized instructions

Advanced Roofline Plot



SIMD = Single Instruction, Multiple Data, ie vectorized instructions

Advanced Roofline Plot



SIMD = Single Instruction, Multiple Data, ie vectorized instructions



Comments about the Roofline Model

In theory

- Gives good insight of the bottleneck of a given algorithm

In practice, use automatic tools

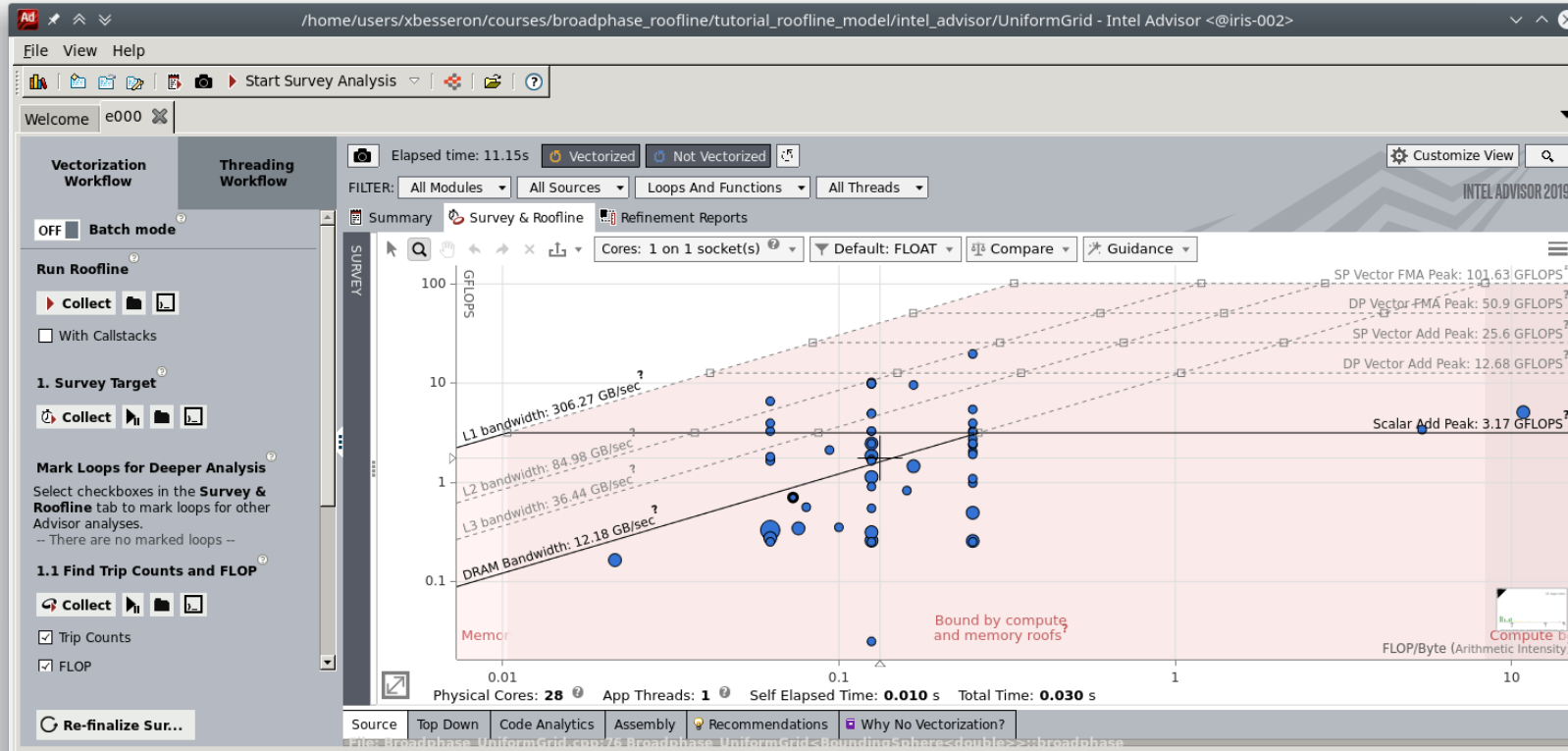
- CPU model can be hard to find
- Algorithm characterization is hard for complex algorithms

Warning

- The Roofline Model tells if an algorithm performs well,
- not if the algorithm is the best for your problem
- e.g. Bubble sort $O(n^2)$ vs Quicksort $O(n \log n)$

Roofline Model in practice

Example with Intel Advisor



Measuring Parallel Performance: Speedup and Scalability

- Number of processors $\rightarrow N$
- Sequential Time $\rightarrow T_1$
- Parallel Time $\rightarrow T_N$

$$\text{Speedup} = \frac{T_1}{T_N}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{N}$$

Strong Scalability:

Problem size is fixed, increase the number of processors
 \rightarrow Constant amount of work in the study

Weak Scalability:

Increase the problem size and the nb of processors with the same ratio
 \rightarrow Constant amount of work per processor

Measuring Parallel Performance: Speedup and Scalability

- Number of processors $\rightarrow N$
- Sequential Time $\rightarrow T_1$
- Parallel Time $\rightarrow T_N$

$$\text{Speedup} = \frac{T_1}{T_N}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{N}$$

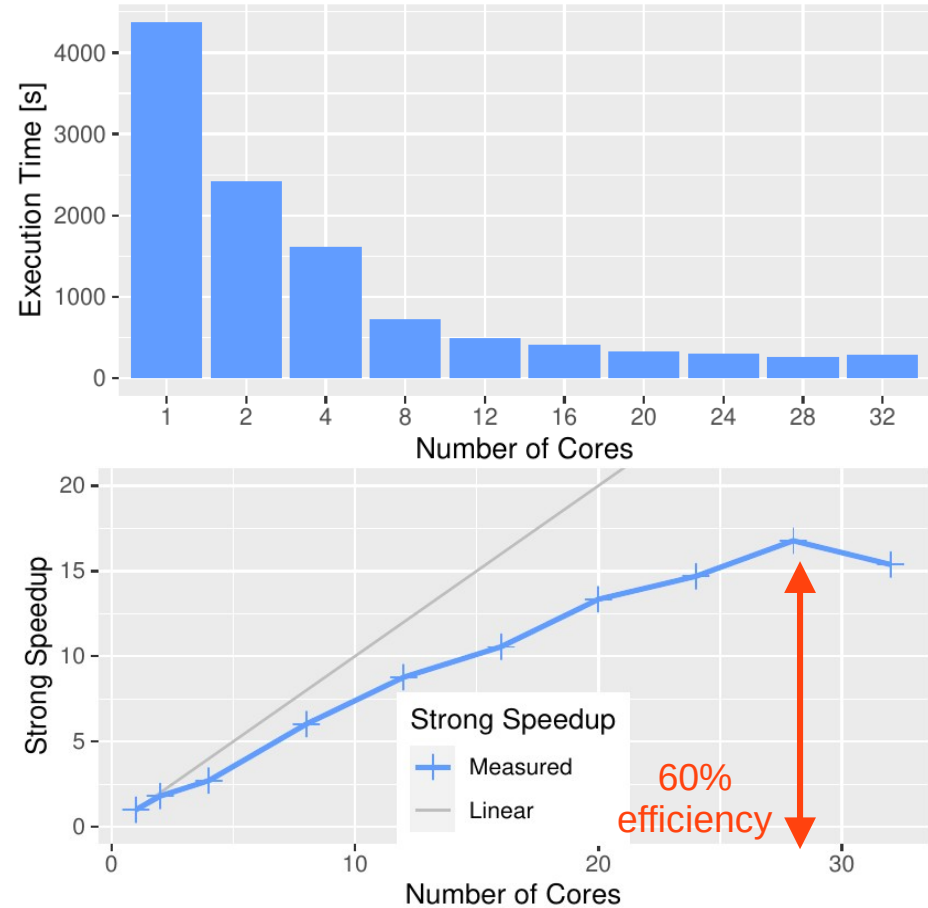
Strong Scalability:

Problem size is fixed, increase the number of processors
 \rightarrow Constant amount of work in the study

Weak Scalability:

Increase the problem size and the nb of processors with the same ratio
 \rightarrow Constant amount of work per processor

Strong scalability



Measuring Parallel Performance: Speedup and Scalability

- Number of processors $\rightarrow N$
- Sequential Time $\rightarrow T_1$
- Parallel Time $\rightarrow T_N$

$$\text{Speedup} = \frac{T_1}{T_N}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{N}$$

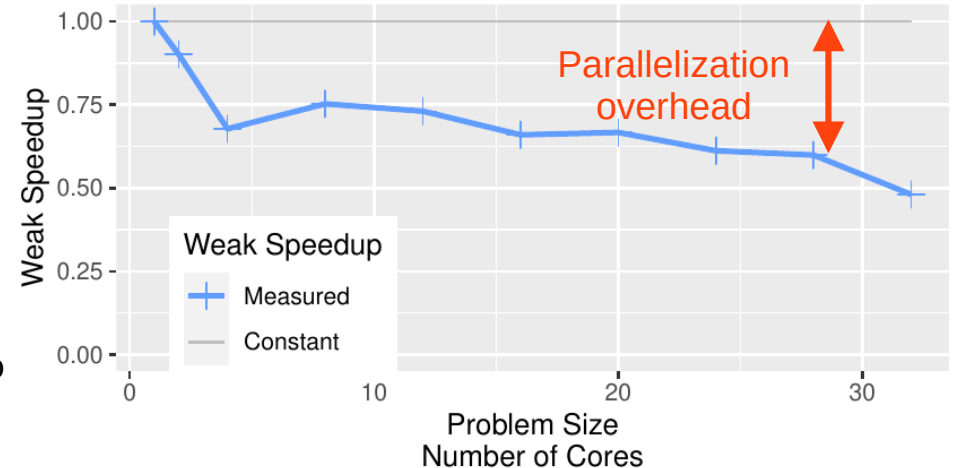
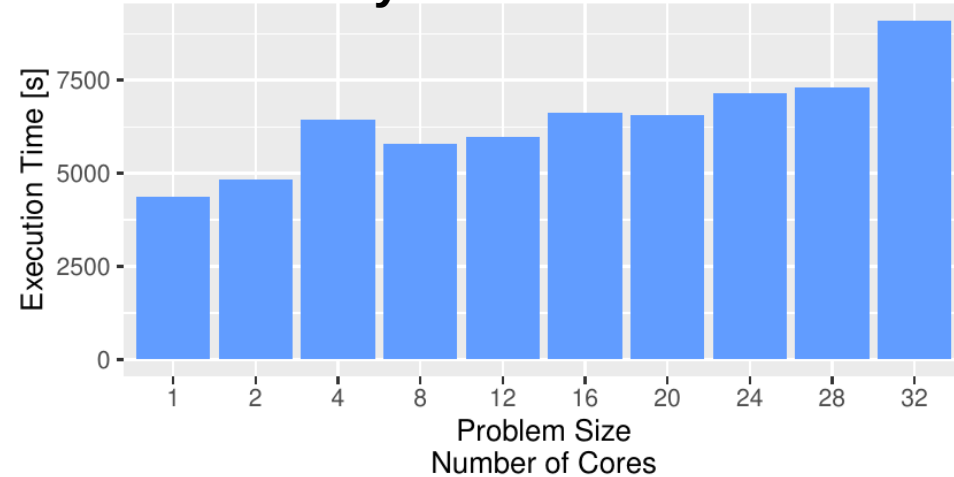
Strong Scalability:

Problem size is fixed, increase the number of processors
 \rightarrow Constant amount of work in the study

Weak Scalability:

Increase the problem size and the nb of processors with the same ratio
 \rightarrow Constant amount of work per processor

Weak scalability



Limit to Scalability: Amdahl's law

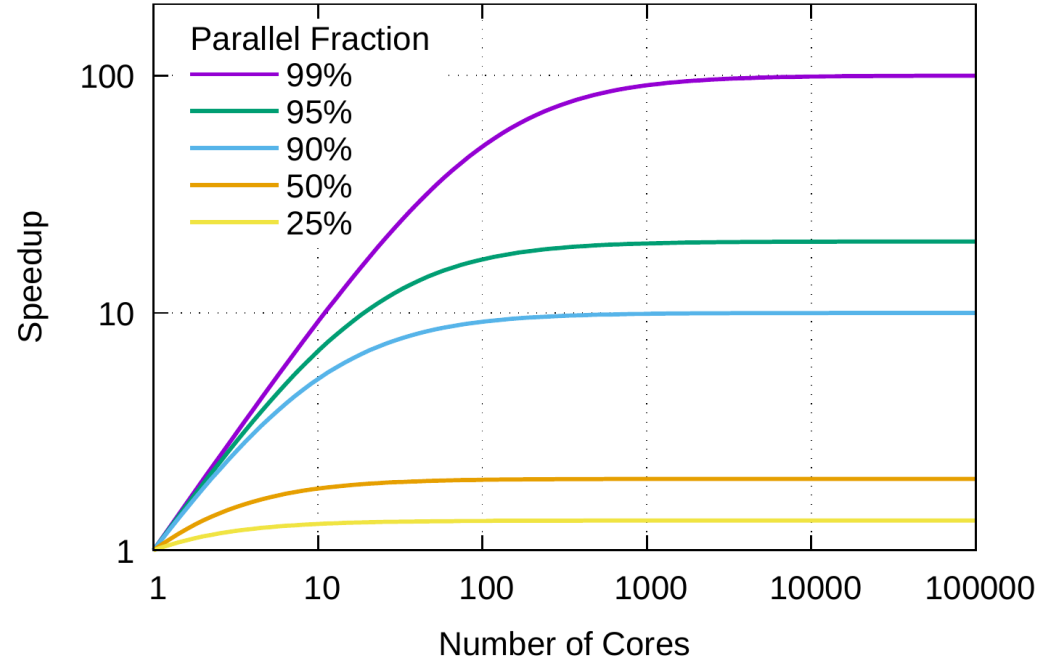
Amdahl's law is a performance model



- Parallel fraction ■ → p
- Serial fraction ■ → $1 - p$
- Number of processors → N

$$\text{Speedup} = \frac{T_1}{T_N} = \frac{1}{1 - p + \frac{p}{N}} \leq \frac{1}{1 - p}$$

Another performance model → Gustafson's law



According to Amdahl's law, scalability is bounded

Limit to Scalability: Load-balancing

Load-balancing

→ Distribution of work between processors

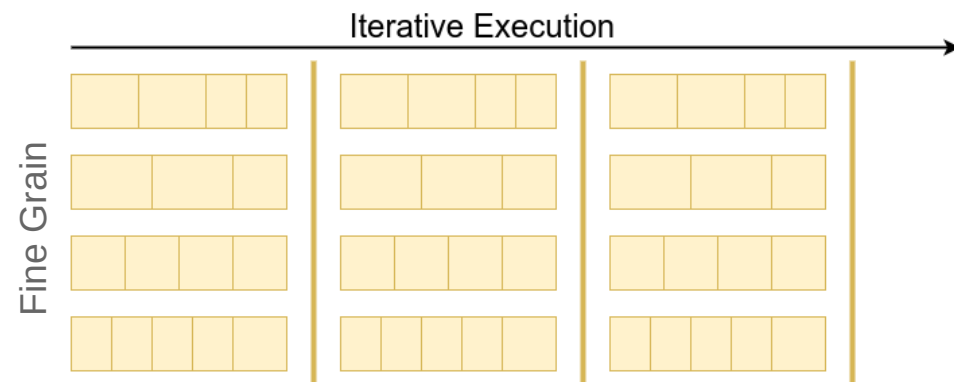
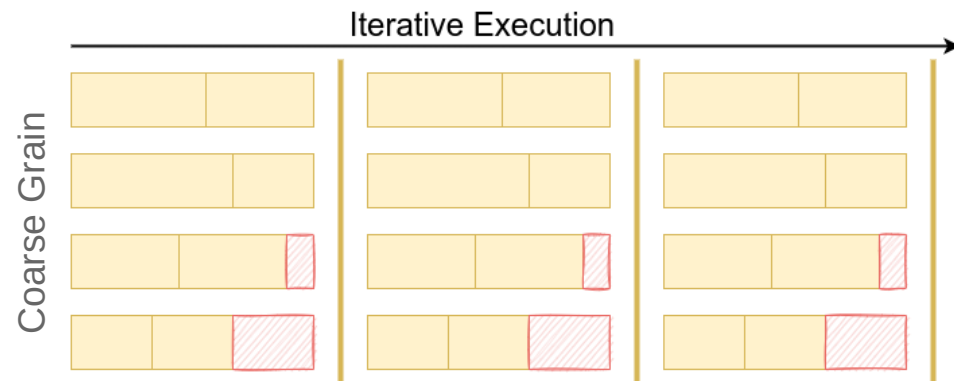
Load unbalance

- Lost computation time
- Accumulates over iterations

→ **Limits the scalability**

- Coarse grain is more difficult to balance than fine grain
- Larger scale requires fine grain

→ A good estimation of the work of each task is critical



High-Performance Computing for the **Simulation of Particles**

Discrete Element Method and XDEM

What is XDEM?

eXtended Discrete Element Method

Simulation software for

Particles Dynamics

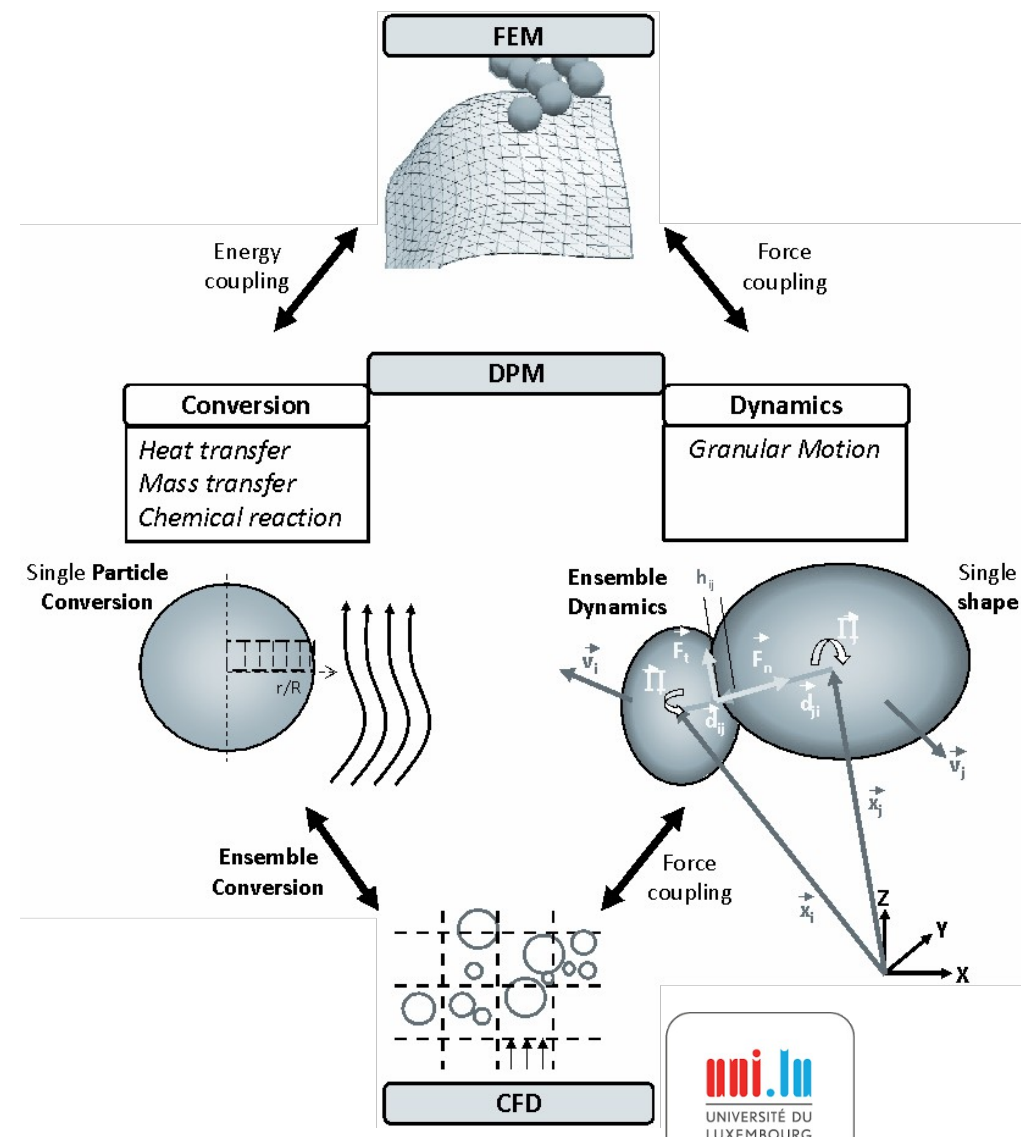
- Force and torques
- Particle motion

Particles Conversion

- Heat and mass transfer
- Chemical reactions

Coupled with

- Computational Fluid Dynamics (CFD)
- Finite Element Method (FEM)



What is XDEM?

eXtended Discrete Element Method

Simulation software for

Particles Dynamics

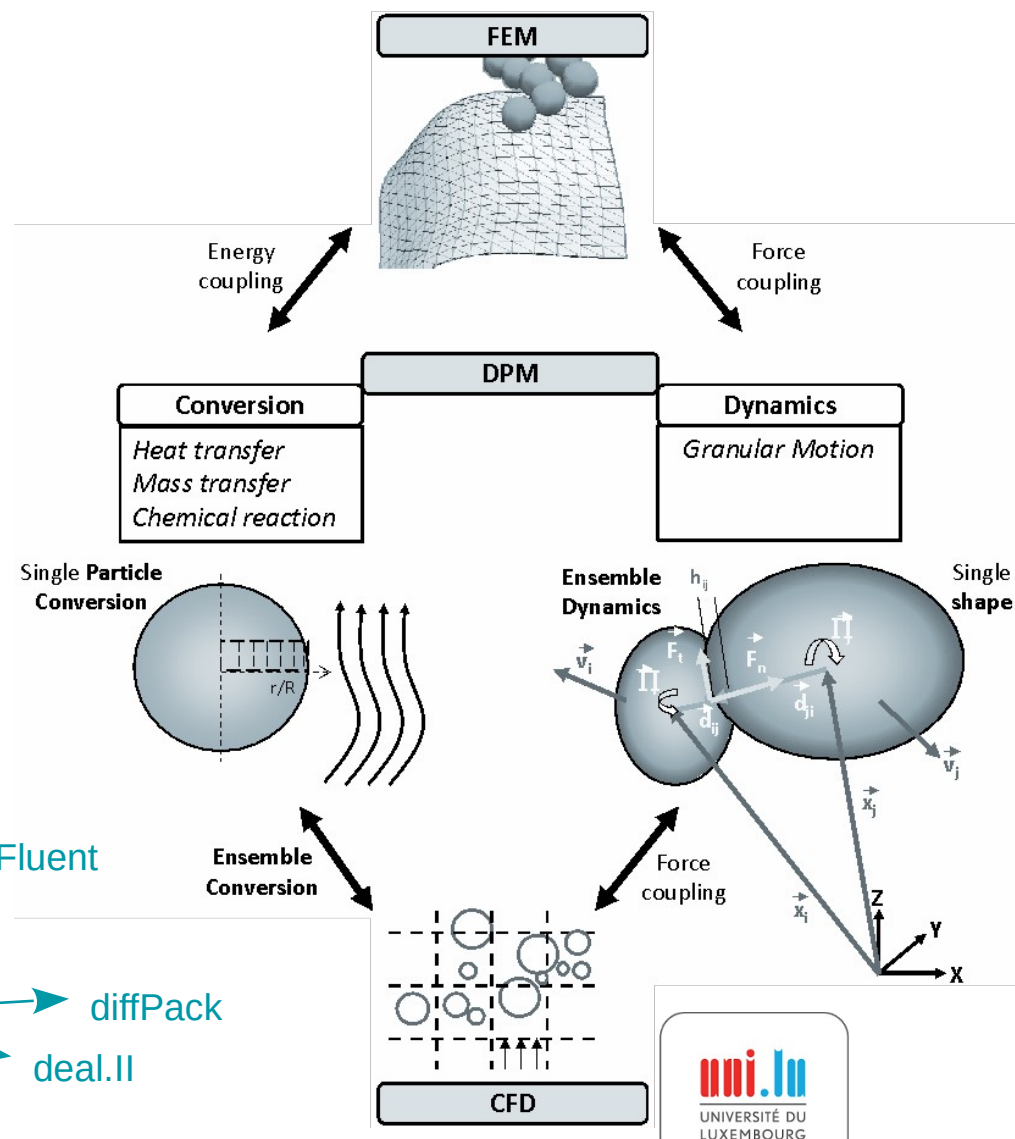
- Force and torques
- Particle motion

Particles Conversion

- Heat and mass transfer
- Chemical reactions

Coupled with

- Computational Fluid Dynamics (CFD)
- Finite Element Method (FEM)



OpenFOAM

ANSYS Fluent

diffPack

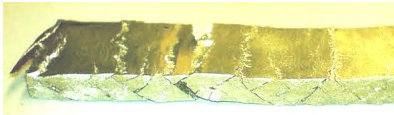
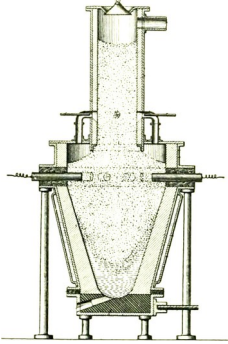
deal.II

CalculiX



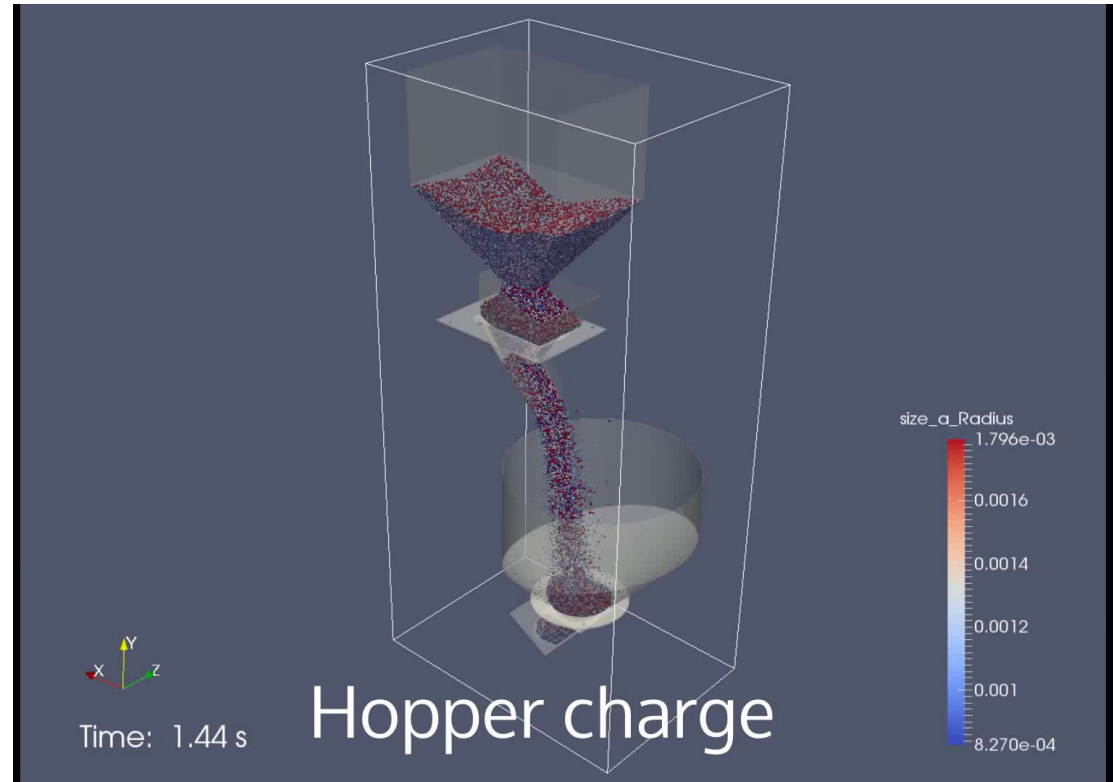
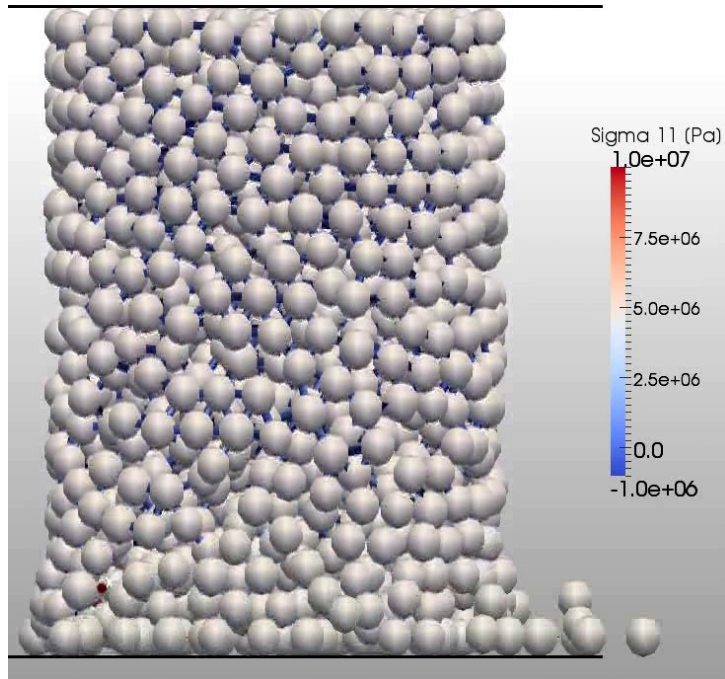
UNIVERSITÉ DU
LUXEMBOURG

Application Examples: XDEM

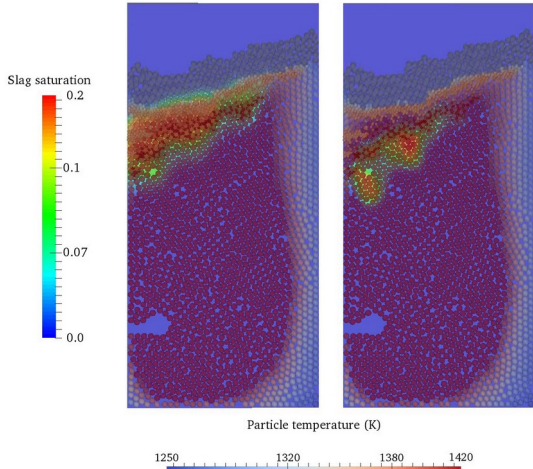


Brittle Failure

Hopper charge and discharge

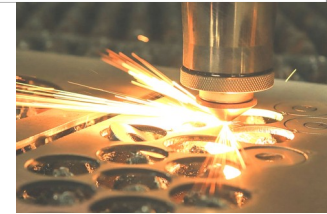
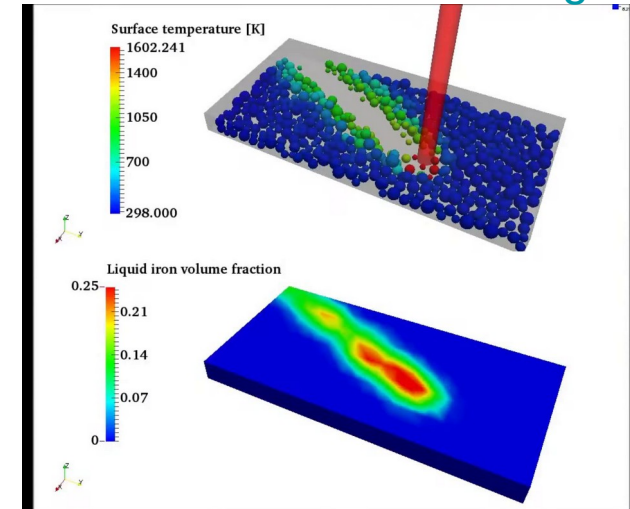


Application Examples: XDEM coupled with CFD

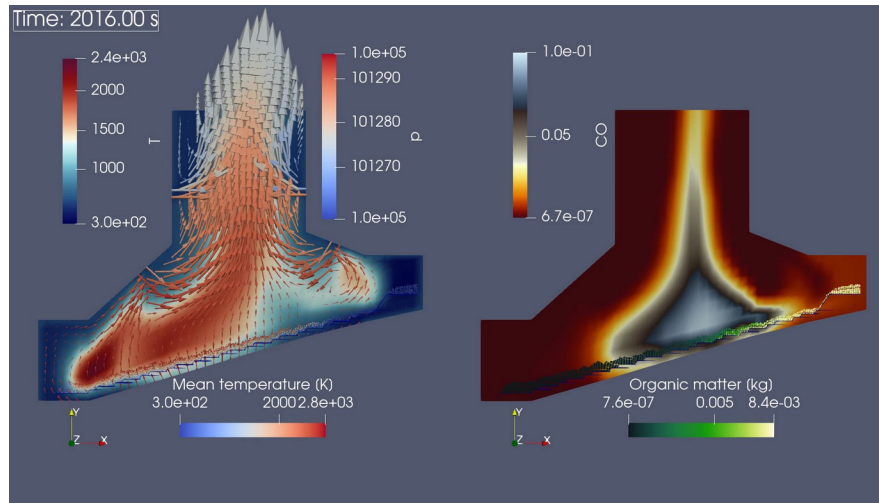


Wood Conversion
in a Biomass Furnace

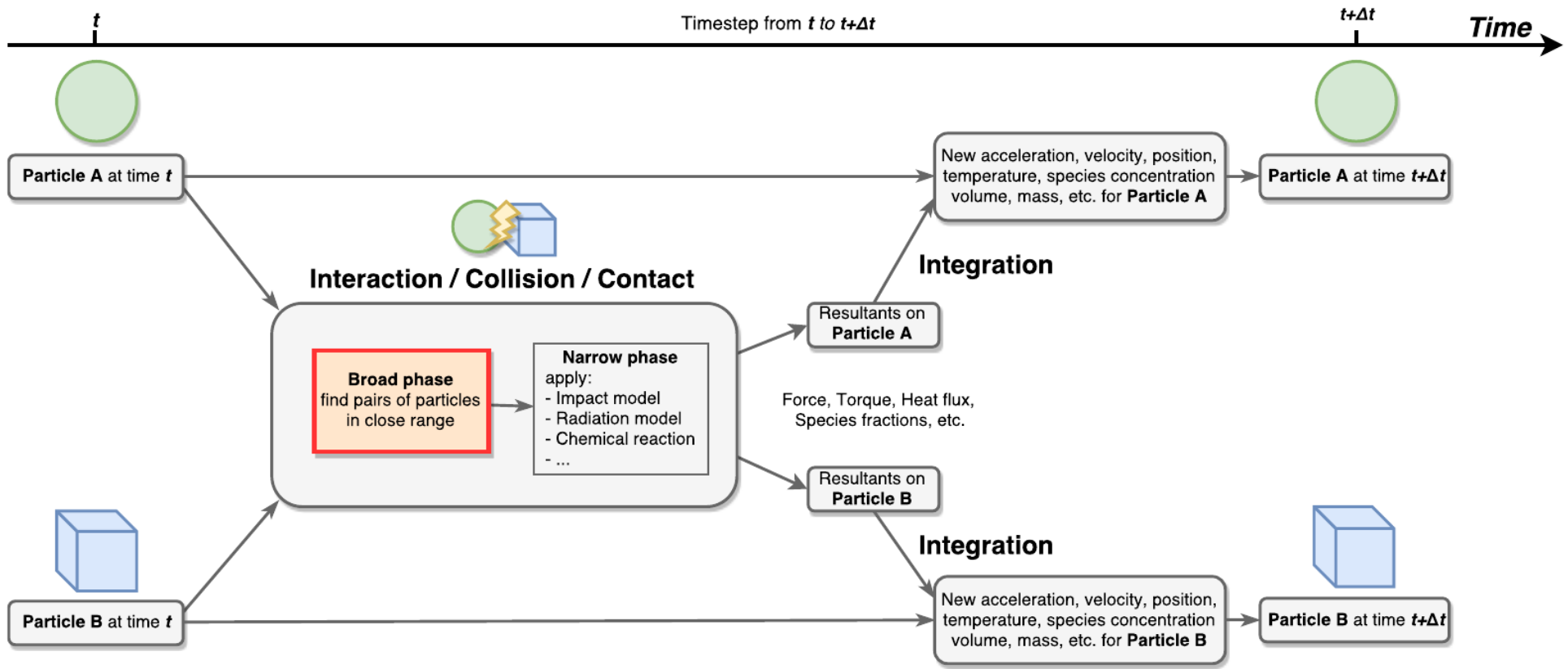
Selective Laser Melting in Additive Manufacturing



Iron & Slag production in a Blast Furnace



Overview of XDEM Execution Flow



Main Computations Phases in XDEM

Broad Phase: Fast but approximate scan to identify the pairs of particles that *could* interact

- uses an approximate shape (bounding volume)

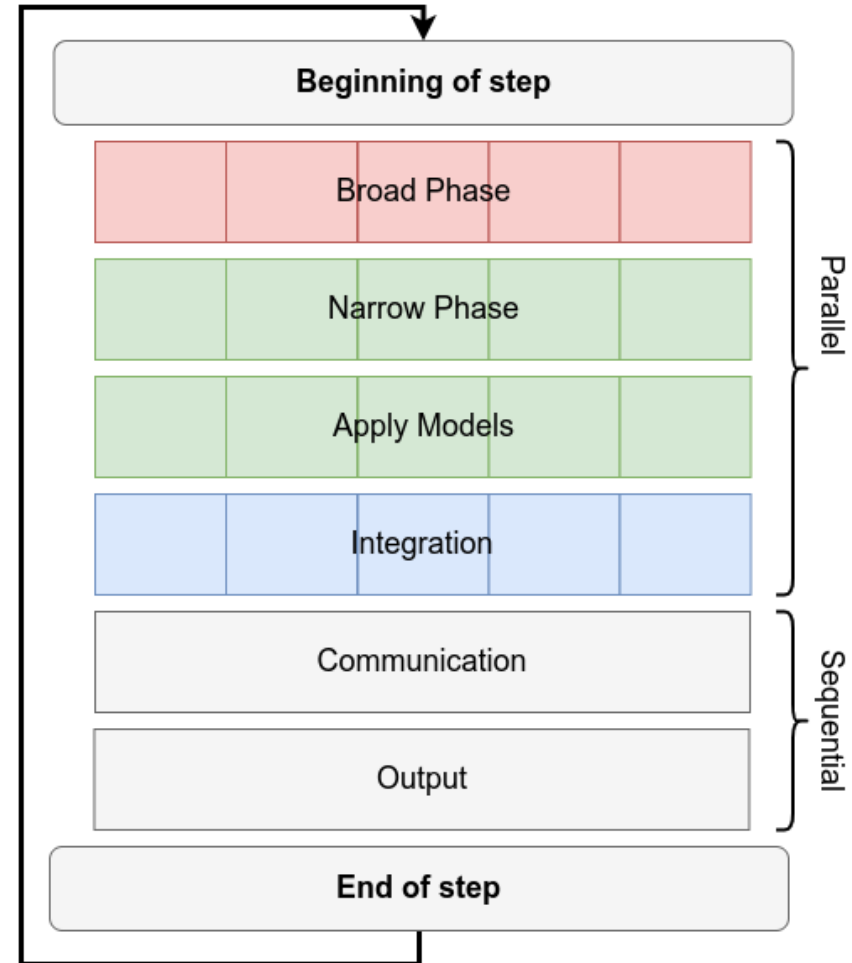
Narrow Phase: Precise collision detection on the particle pairs identified in the broad-phase

- uses the actual shape (sphere, cube, cylinder, etc.)
- calculates the distance/overlap between particles

Apply Models: Apply the physics models to each pair of interacting particles

- accumulate contributions to each particle:
Contact → *force, torque, ...*
Conduction/Radiation → *heat flux, ...*

Integration: Update the particle states by integrating the contributions from all the interacting partners



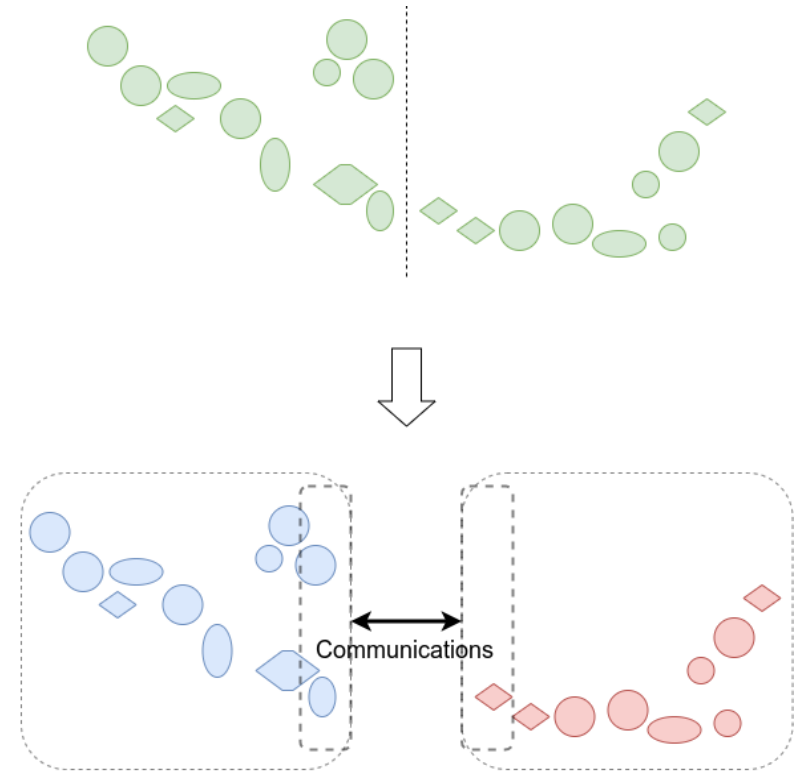
High-Performance Computing for the Simulation of Particles

Domain Decomposition with MPI
and Load-Balancing

Domain Decomposition in XDEM

Decomposing the set of particles?

- Particles move during the simulation
- Neighborhood relations change
- Create undetected dependencies
- Would require frequent re-partitioning



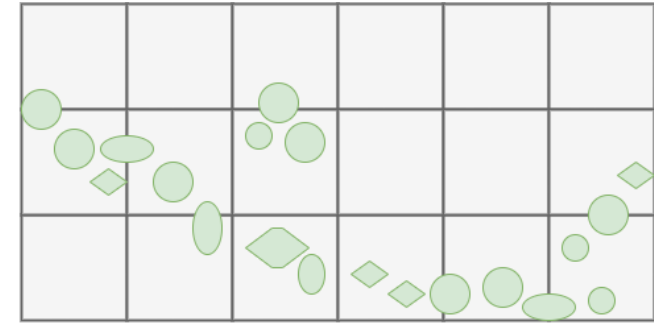
Domain Decomposition in XDEM

Decomposing the set of particles?

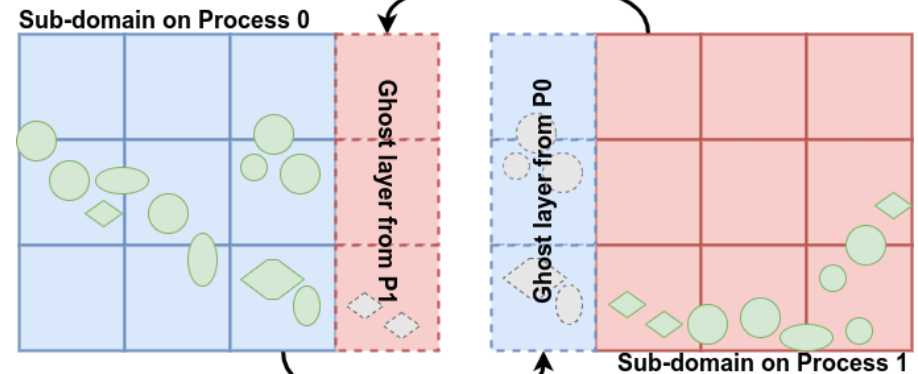
- Particles move during the simulation
- Neighborhood relations change
- Create undetected dependencies
- Would require frequent re-partitioning

Use a static regular grid to 'store' particles

- Find location of a particle in constant time
- Size of grid cells adapted for collision detection
- No missing communication
- Re-partitioning only required in case of imbalance

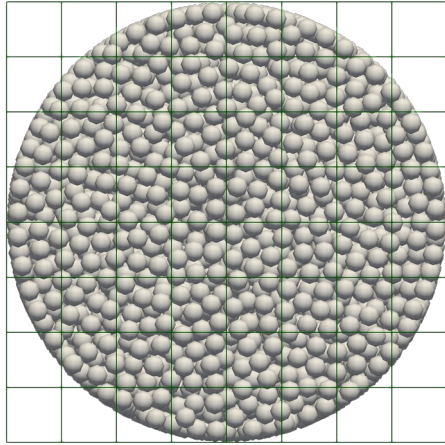


MPI communication from 1 to 0

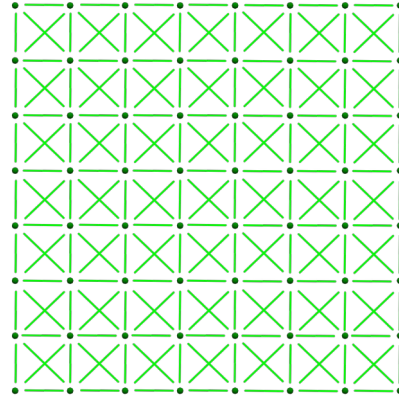


MPI communication from 0 to 1

Partitioning and Load-Balancing for XDEM

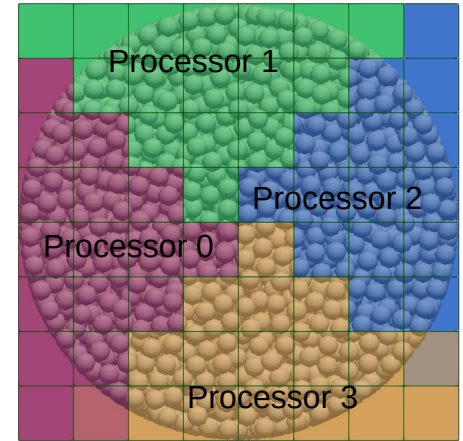


Particles in the cell grid



From grid to graph

- Node \leftarrow Cell
- Node weight $\leftarrow f(\text{nb particles})$
~ Computation cost
- Edge \leftarrow Neighborhood relation
- Edge weight $\leftarrow g(\text{nb particles})$
~ Communication cost
- Node Coordinates (topologic approaches)



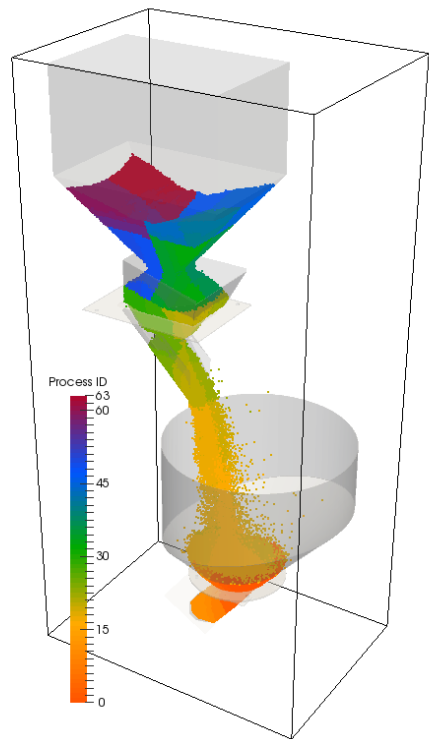
Partitioning algorithm

- Orthogonal Recursive Bisection
- METIS
- SCOTCH
- Zoltan PHG, RCB, RIB, ...
- etc.

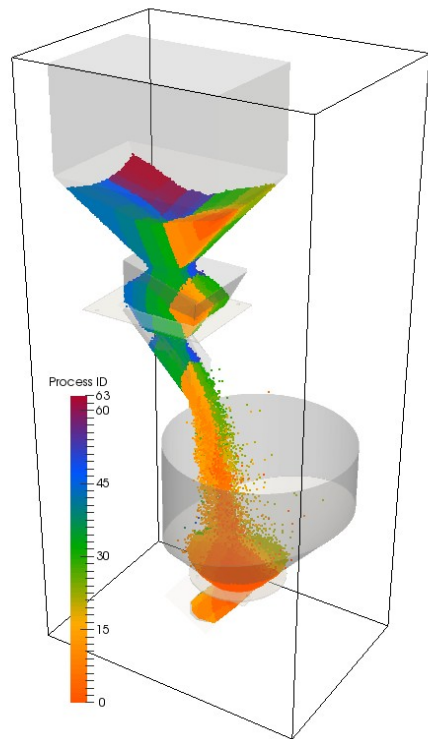
Objectives

- Balance the computation cost
- Minimize the communication cuts

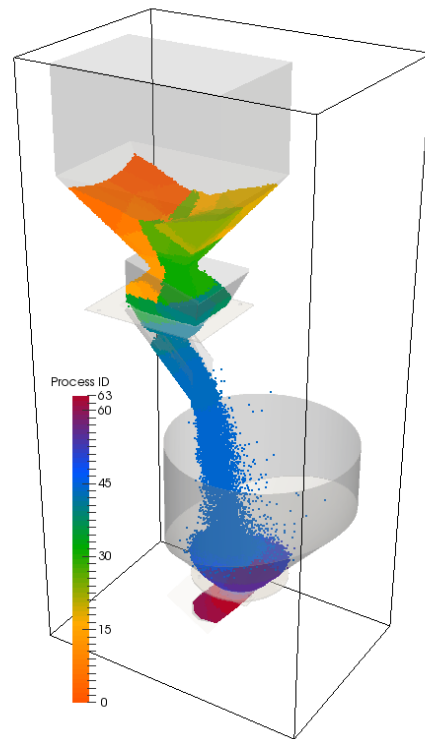
Example of Load-Balancing



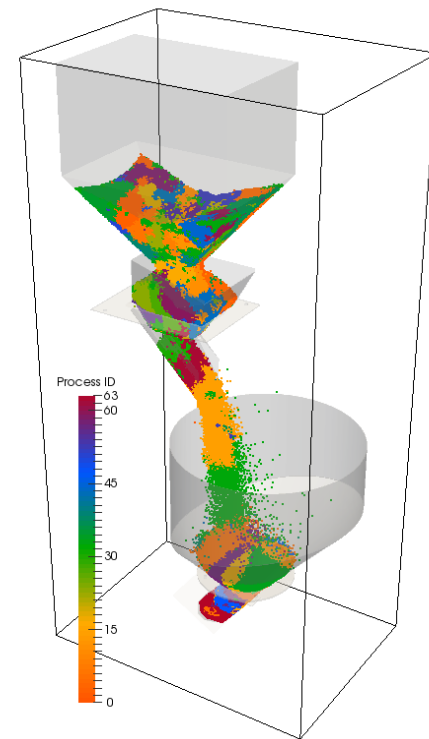
Zoltan RCB
(Recursive Coordinate Bisection)



ORB
(Orthogonal Recursive Bisection)



Zoltan RIB
(Recursive Inertial Bisection)



SCOTCH K-way

Weight estimation for load-balancing

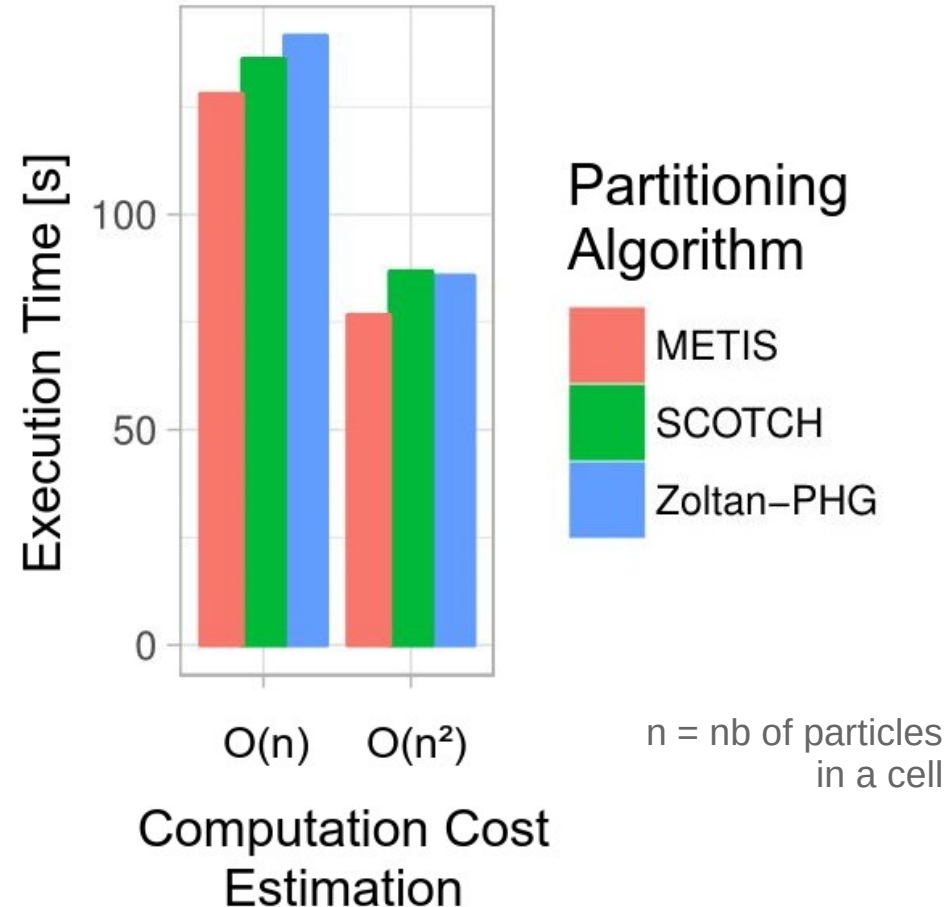
How to estimate the computing cost ?

- Difficult to measure at the level of a single cell
- Multiple phases and different complexities

<i>Computation Phase</i>	<i>Complexity</i>
Broad-phase	$O((nb \text{ particles})^2)$
Narrow-phase	$O(nb \text{ interactions })$
Apply Models	$O(nb \text{ interactions })$
Integration	$O(nb \text{ particles })$

- Nb of interactions is difficult to estimate

→ **Work in progress**

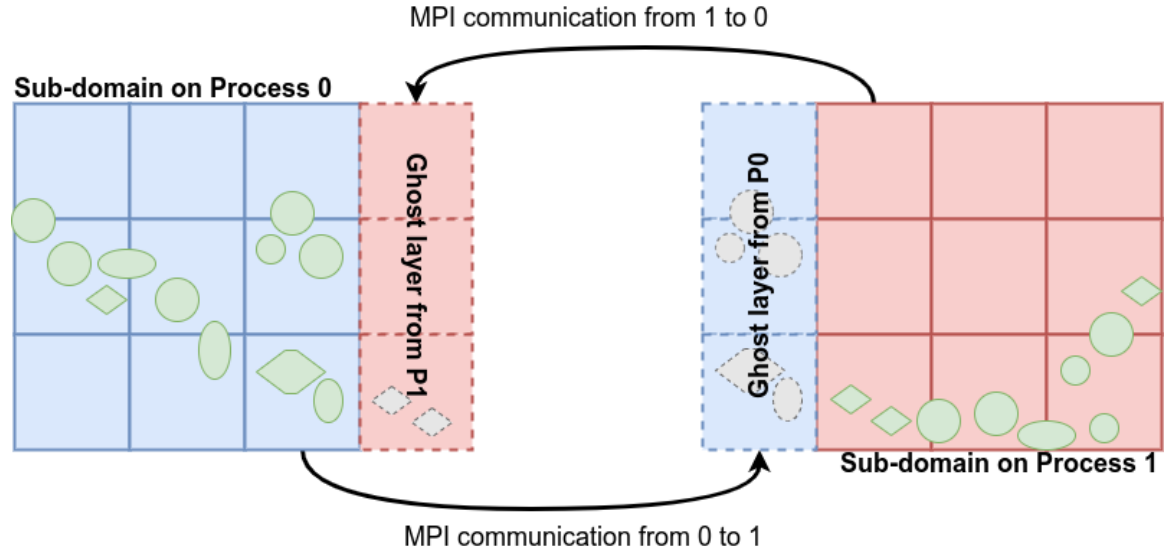


High-Performance Computing for the Simulation of Particles

Fine grain parallelization with OpenMP

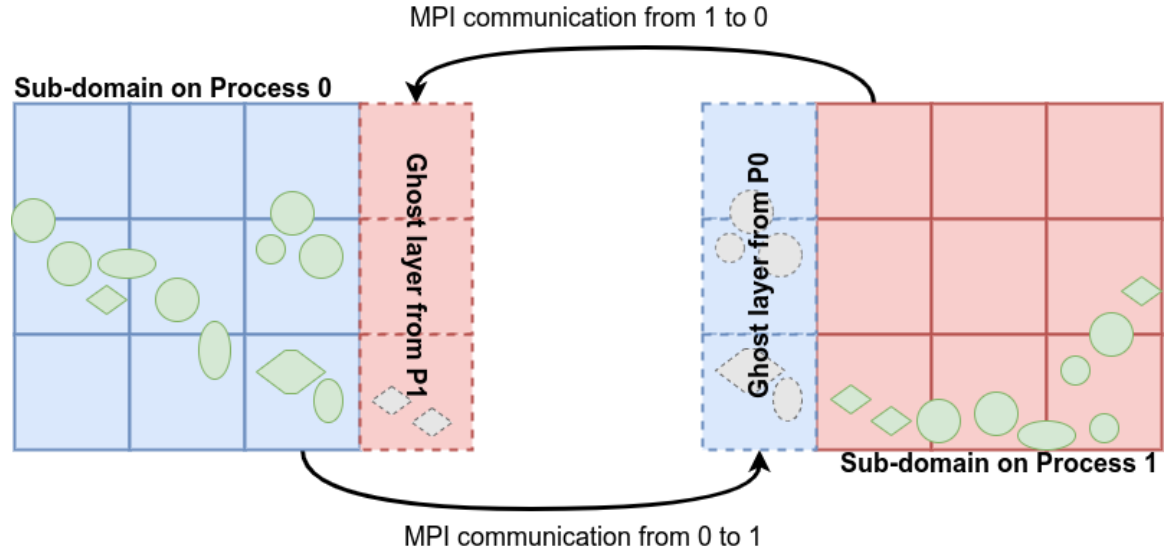
Hybrid Parallelization MPI+OpenMP of XDEM

Decomposed
Particle Domain

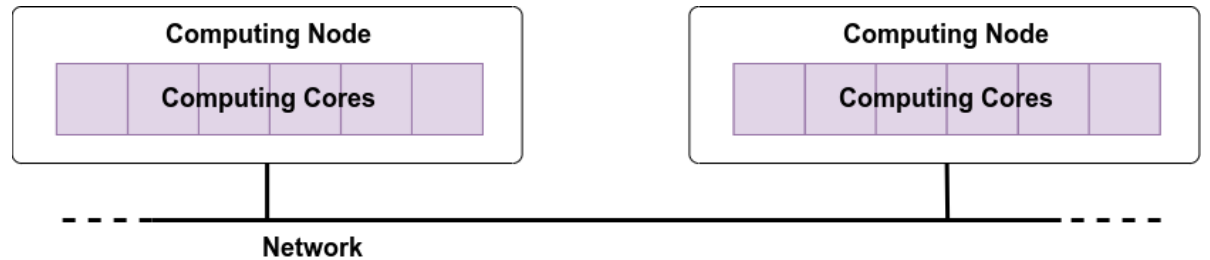


Hybrid Parallelization MPI+OpenMP of XDEM

Decomposed Particle Domain

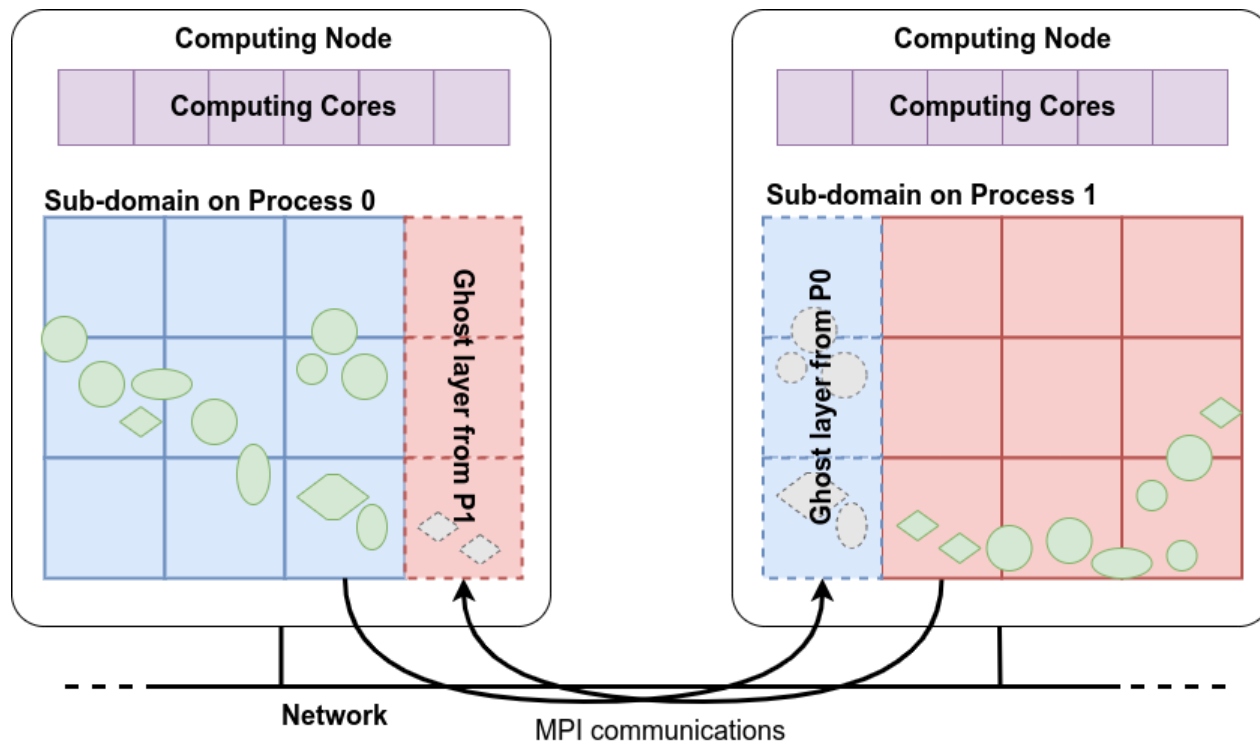


Computing Platform



Hybrid Parallelization MPI+OpenMP of XDEM

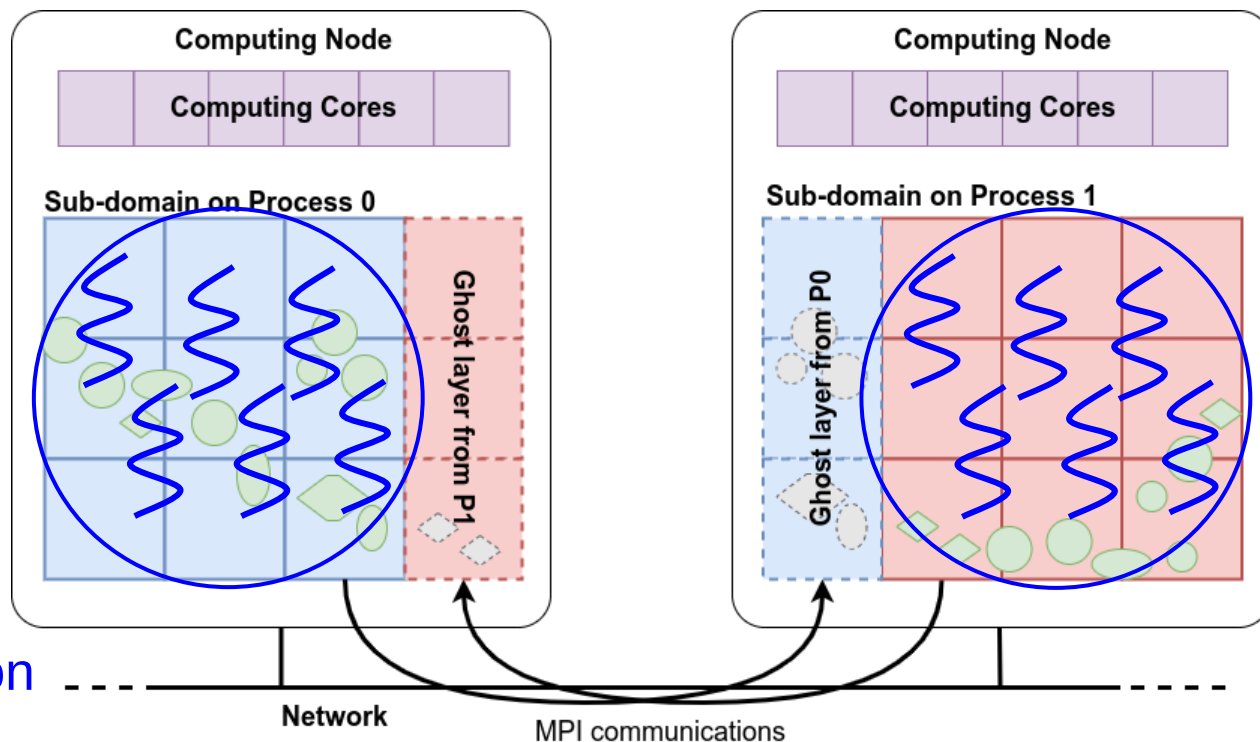
Sub-domains distributed on computing nodes with MPI
→ Coarse grain //



Hybrid Parallelization MPI+OpenMP of XDEM

Sub-domains distributed on computing nodes with MPI
→ Coarse grain //

Intra-subdomain parallelization with OpenMP
→ Fine grain //



XDEM parallelization with OpenMP

Parallelization at a fine grain:

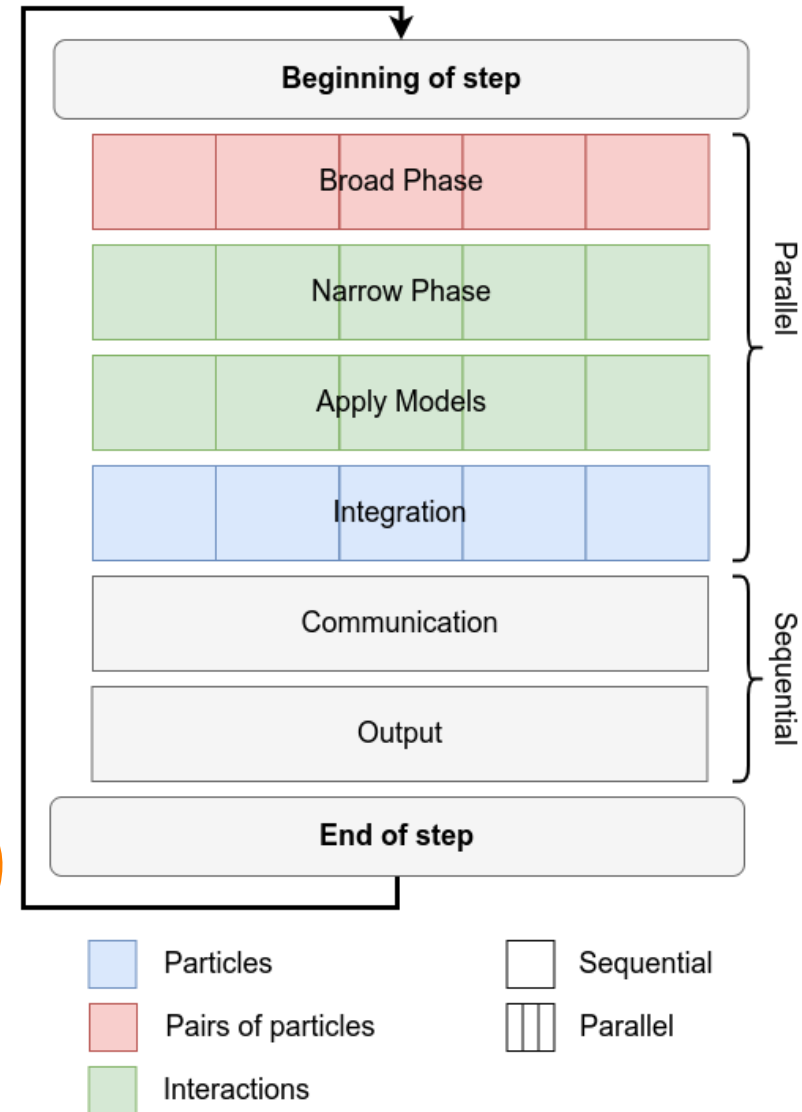
- Particles, Pairs of particles and Interactions

Guided by the type of accesses:

- Iterate on the objects being modified to avoid concurrent accesses (when possible)
- Use containers with random access iterators

	Read Access	Write Access	Iteration on
Broad Phase	Particles	Interactions	Particle pairs
Narrow Phase	Interactions	Interactions	Interactions
Apply Models	Interactions	Particles	Interactions
Integration	Particles	Particles	Particles

Potential concurrent accesses!



Concurrency write

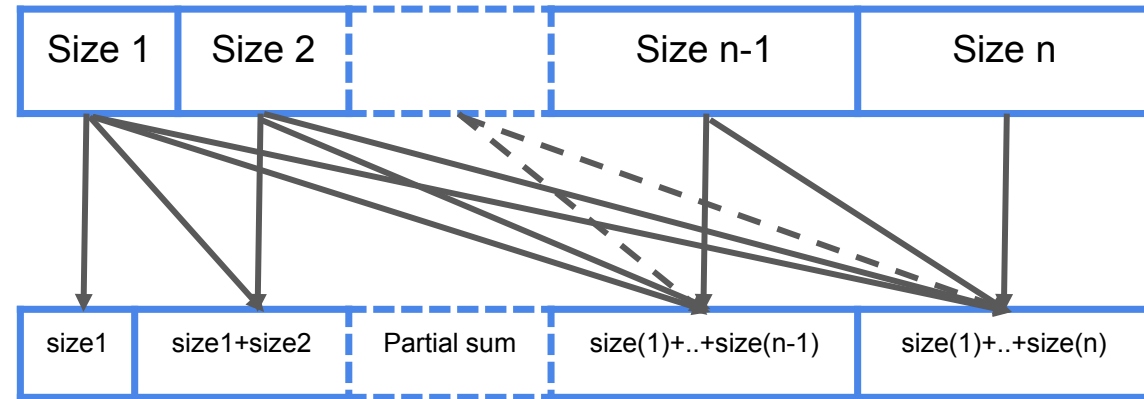
How to fill Interactions vector concurrently?

→ Unknown number of interactions



Solution

- Each thread fills a private deque
- Perform a partial sum of sizes
- Copy in shared vector at the position defined by the partial sum
- Synchronization barrier at the end



→ No critical or atomic regions

Memory allocator

XDEM C++ code is highly dynamic

→ Intensive calls to the memory allocator

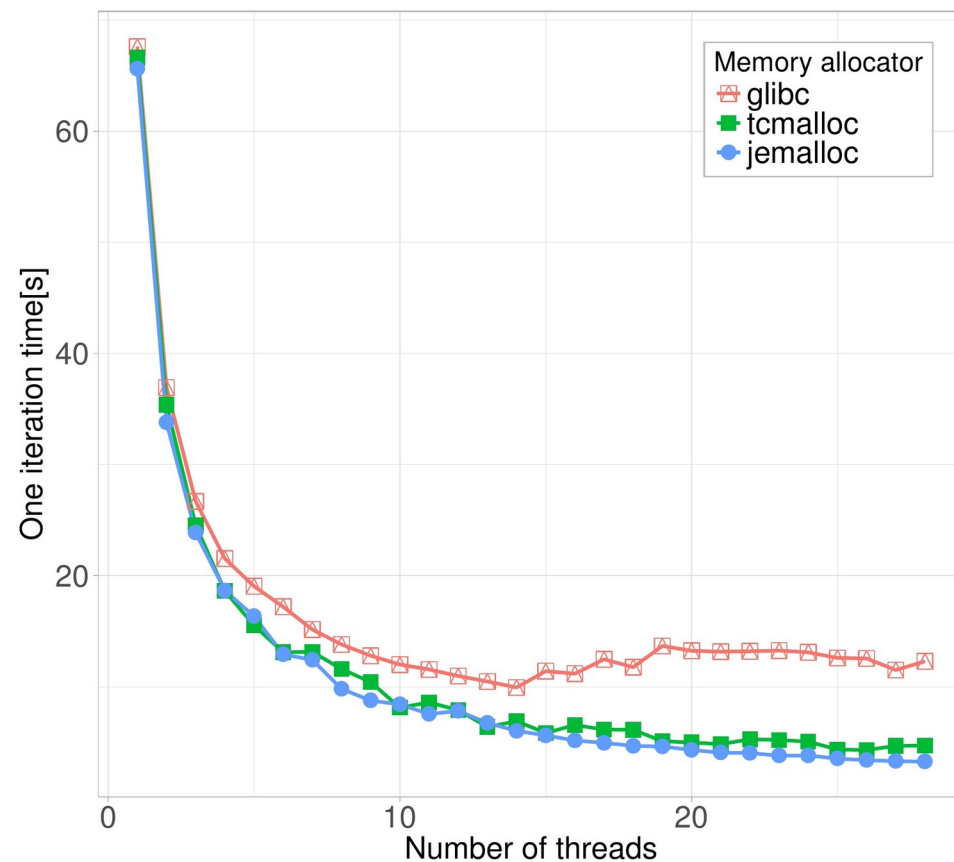
Default **glibc** memory allocator

- uses locks internally
- Limits the scalability of threaded executions

Optimized memory allocators

- **Jemalloc** based on independent arenas
- **TCMalloc** based thread cache

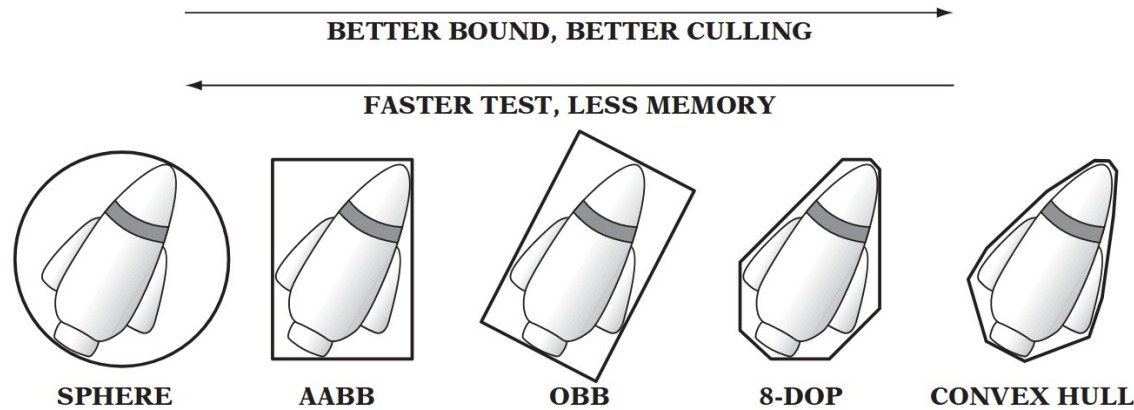
→ **3-4 times faster on 28 cores**



High-Performance Computing for the Simulation of Particles

Faster Broad-Phase with Roofline Analysis

Bounding Volumes in XDEM Broad-phase



Real-Time Collision Detection, by Christer Ericson, 2005.

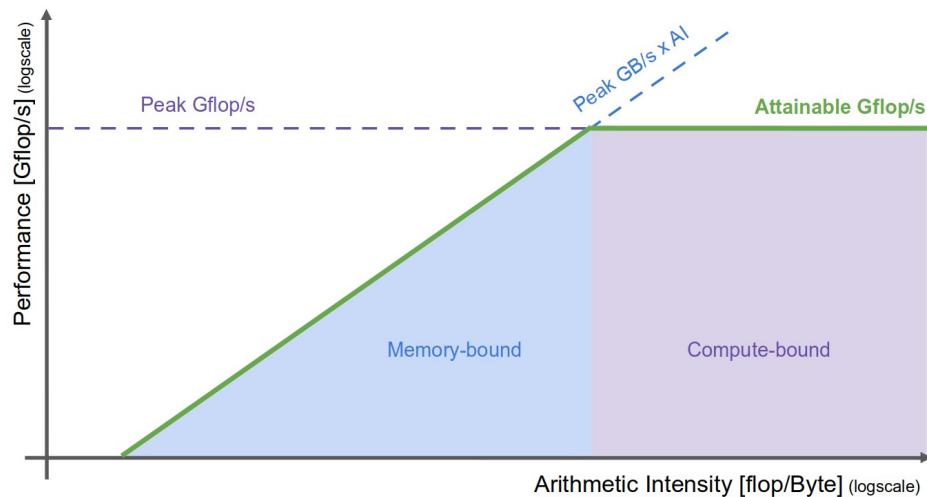
Which bounding volume for the broad-phase?

- Bounding Sphere (BS)?
- Axis Aligned Bounding Box (AABB)?

Roofline Analysis for Bounding Volumes

- Broad-phase is memory-bounded

Intersection of 2 bounding volume?



	Memory	Complexity of n	AI
Bounding Sphere	2 x 4 reals (position + radius)	11 arithmetic ops 1 comparison	1.38 flop/real
Axis Aligned Bounding Box	2 x 6 reals (upper + lower corners)	6 comparisons 5 logical AND	0.5 flop/real

- Bounding Spheres release the pressure on memory bandwidth
- Using `float` type instead `double` also reduces memory accesses

⇒ Use Bounding Spheres of `floats`

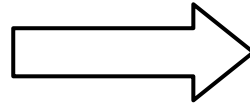
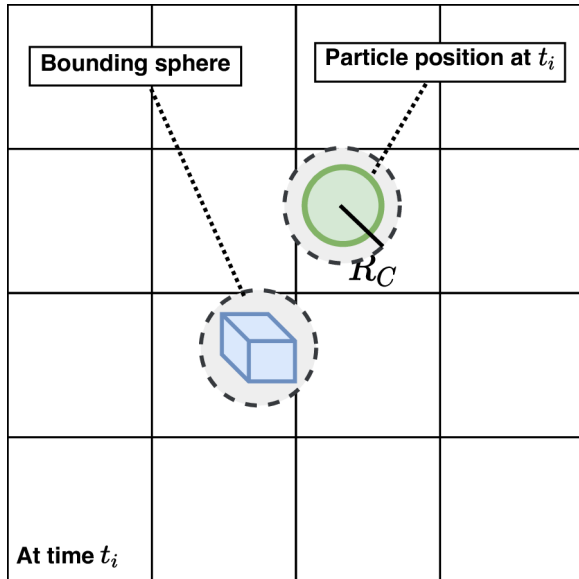
High-Performance Computing for the Simulation of Particles

Verlet Buffer approach for Collision Detection

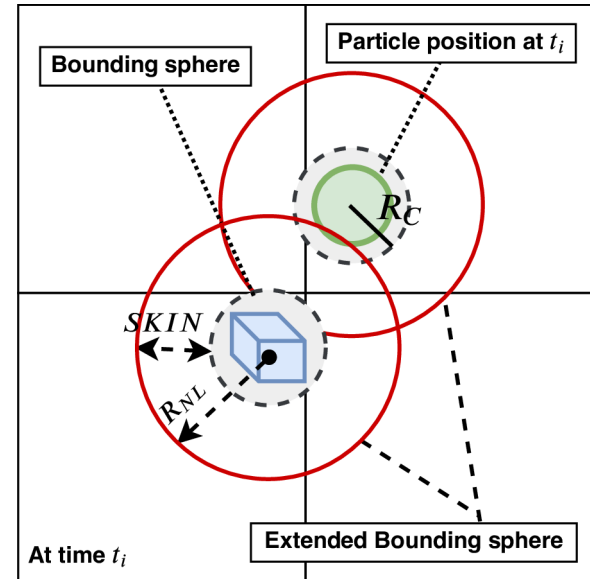
Verlet Buffer for Collision Detection in XDEM 1/3

Idea → Inspired from Computer “Experiments” on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules by L. Verlet, 1967.

- Extend the range of collision detection in the Broad Phase
- Potential collision partners are valid for many iterations



Extend Bounding Spheres with
 $Skin = \mathbf{K} \cdot \mathbf{V}_{particle} \cdot dt$



Verlet Buffer for Collision Detection in XDEM 2/3

Next Timesteps

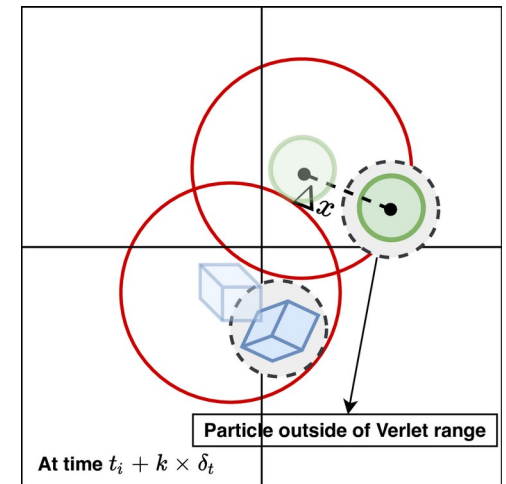
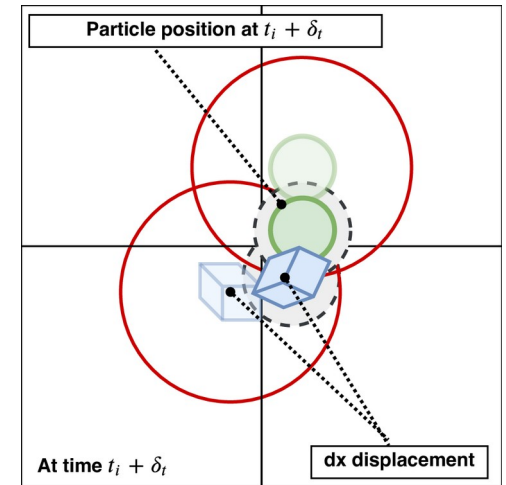
If all particles are still within their extended bounding spheres

- Skip the Broad Phase
- Proceed with the Narrow Phase using previously-calculated list of interaction pairs

If one particle exits its bounding spheres

- Re-calculate the Broad Phase with new extended bounding spheres

- Narrow Phase is always executed
- Identical results are guaranteed



Verlet Buffer for Collision Detection in XDEM 3/3

Increasing K

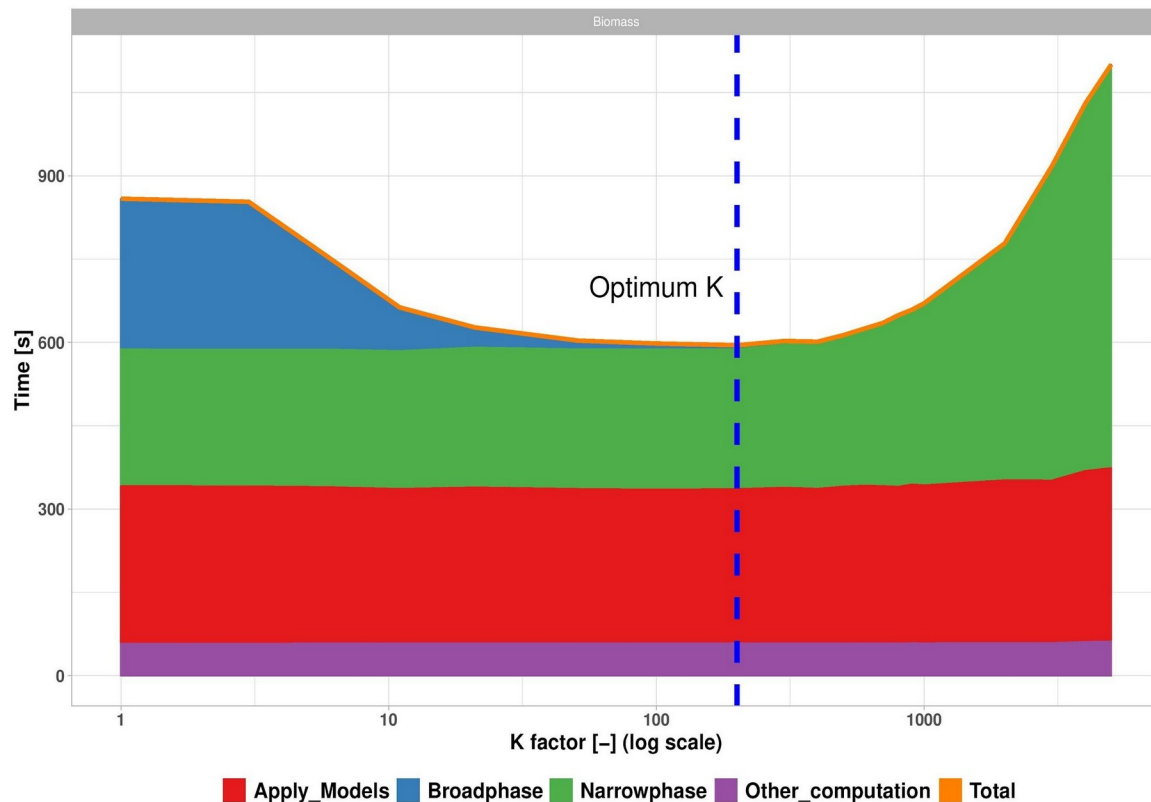
- Increasing cost of **Narrow Phase**
- Increasing cost of each **Broad Phase**
- But less executions of **Broad Phase**
- Less time spent in **Broad Phase** overall

K = 200 is a good default value

- Simulation time reduced by 18% to 81%

Regression model for K trained on examples

- Best K value between 150 and 600
- Additional improvement between 0% to 26%



Going further:

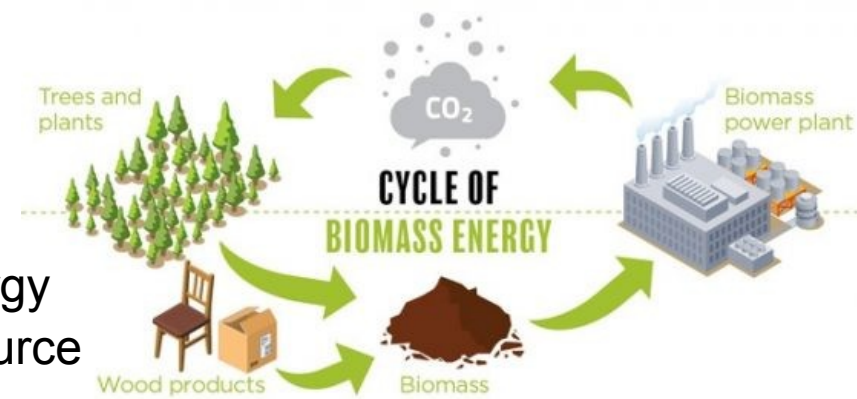
Discrete Element Method (DEM)
+
Computational Fluid Dynamics (CFD)

Parallel Multi-Physics Simulation
of a Biomass Furnace

Biomass Combustion

Biomass combustion (e.g. wood chips)

- widely used for generating electric and thermal energy
- renewable and potentially carbon-neutral energy source

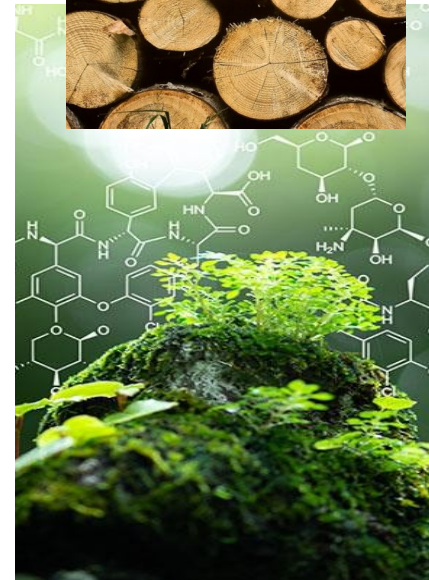


Combustion process

- very complex
- requires advanced techniques to minimize harmful gas emissions

Alternative biomass

- wood waste, straw, bark, olive pits, nut shells, grain husks, bagasse, etc.
- can cause problems due to their chemical composition, ash melting temperature, humidity, ash content, calorific value and others.



Combustion process in a biomass furnace

Combustion chamber of a biomass furnace

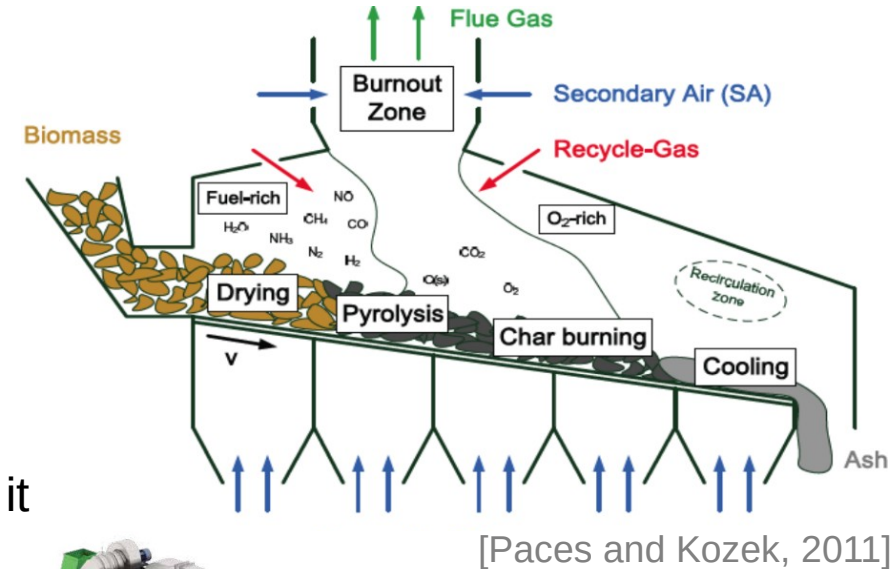
- forward acting grate
- transports the fuel through the furnace

The fuel undergoes a number of steps

- **drying**, **pyrolysis**, **char burning**, **cooling** in which it releases hydrocarbons
- hydrocarbons are **burned** in the gas phase

Use **numerical simulations**

- to study efficiency and performance
- and reduce the costs of experiments



Numerical Approach for Biomass Furnace: Multi-Physics Simulation

Two-way **volume coupling** between
Discrete Element Method (DEM) and
Computational Fluid Dynamics (CFD)

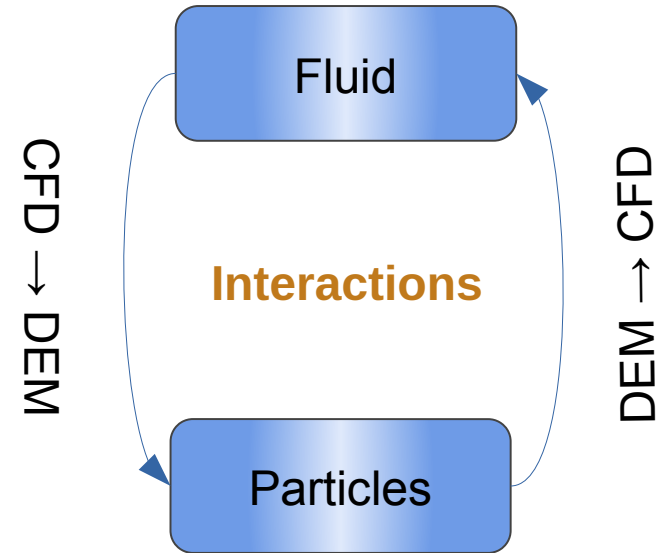
XDEM (Lagrangian) for:

- Motion and collisions of biomass particles
- Thermodynamic Conversion of biomass particles

OpenFOAM (Eulerian) for:

- Flow of gas phase
- Reactions in the gas phase

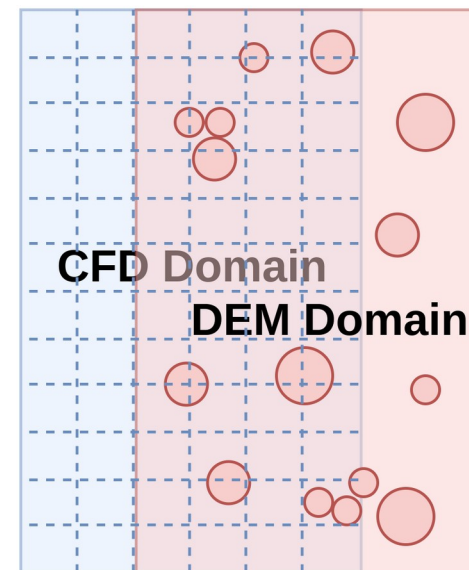
CFD-DEM coupling is required to capture the physics of biomass furnaces and offers unprecedented insight.



CFD-DEM Parallel Coupling: Challenges

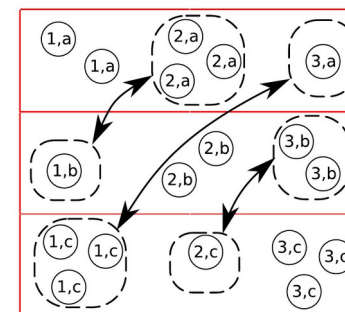
Challenges in CFD-XDEM parallel coupling

- Combine different independent software
- **Volume coupling** \Rightarrow Large amount of data to exchange
- Different distributions of the computation and of the data
- DEM data distribution is dynamic
- Data interpolation between meshes



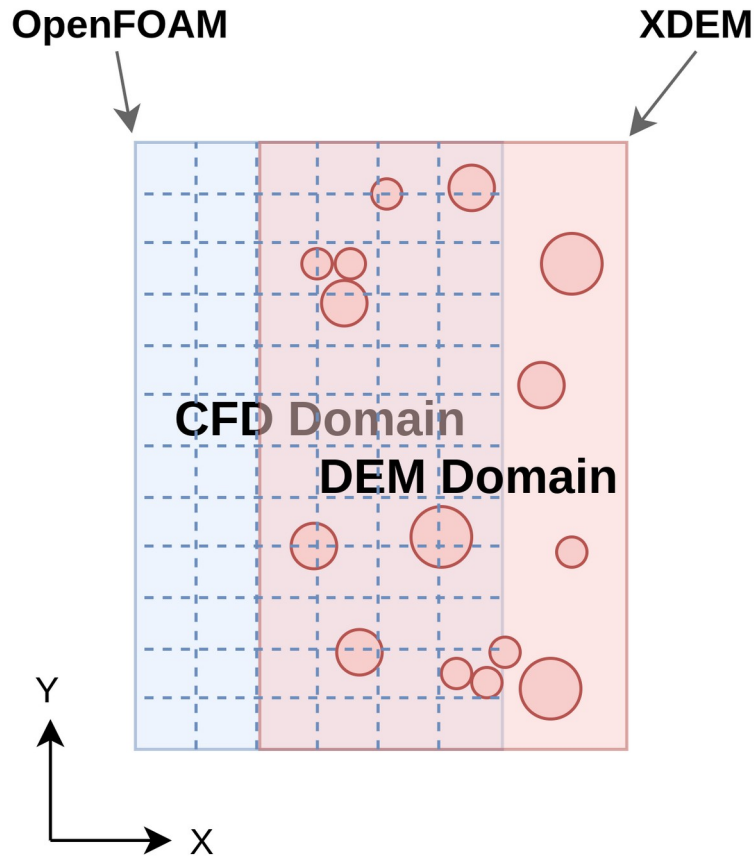
Classical Approaches

- Each software partitions its domain independently
- Data exchange in a peer-to-peer model



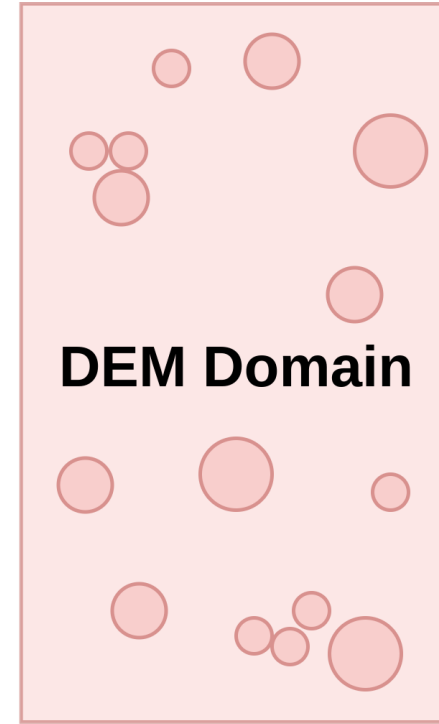
SediFoam [Sun2016]

CFD-DEM Parallel Coupling: Challenges



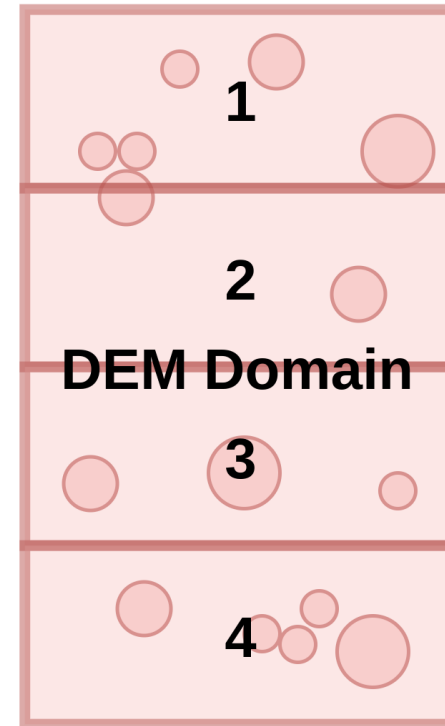
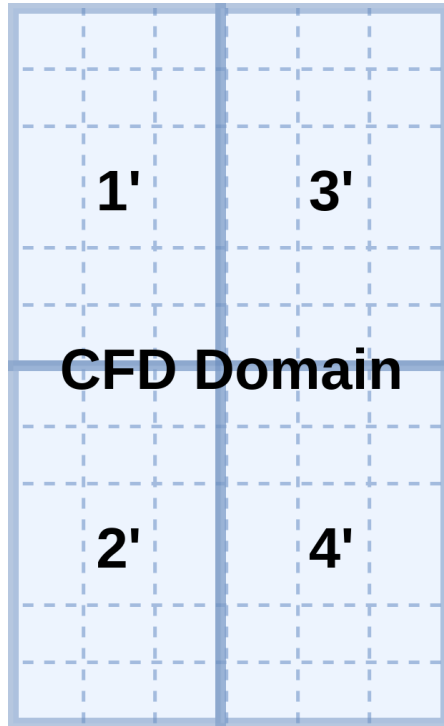
The domains overlap in space

CFD-DEM Parallel Coupling: Challenges



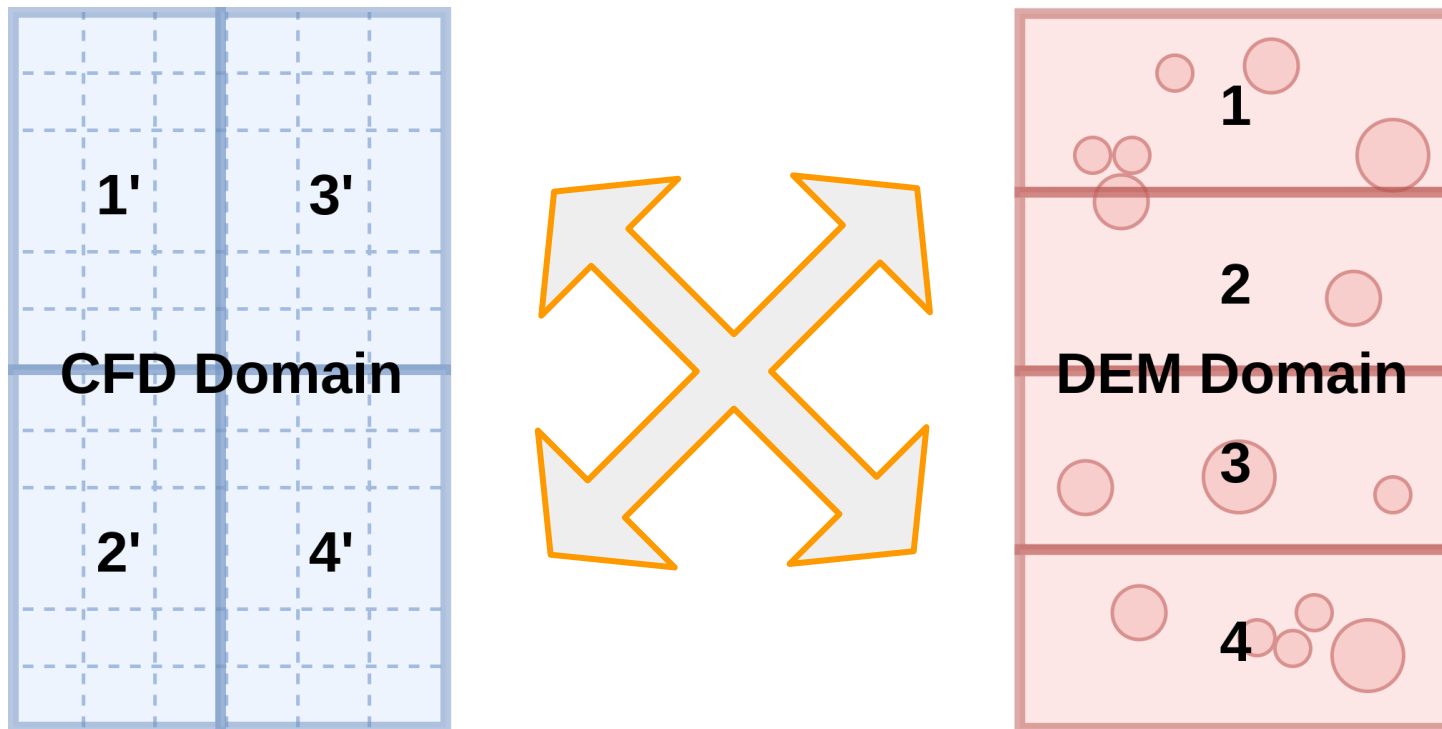
CFD-DEM Parallel Coupling: Challenges

Classical Approach: the domains are partitioned independently



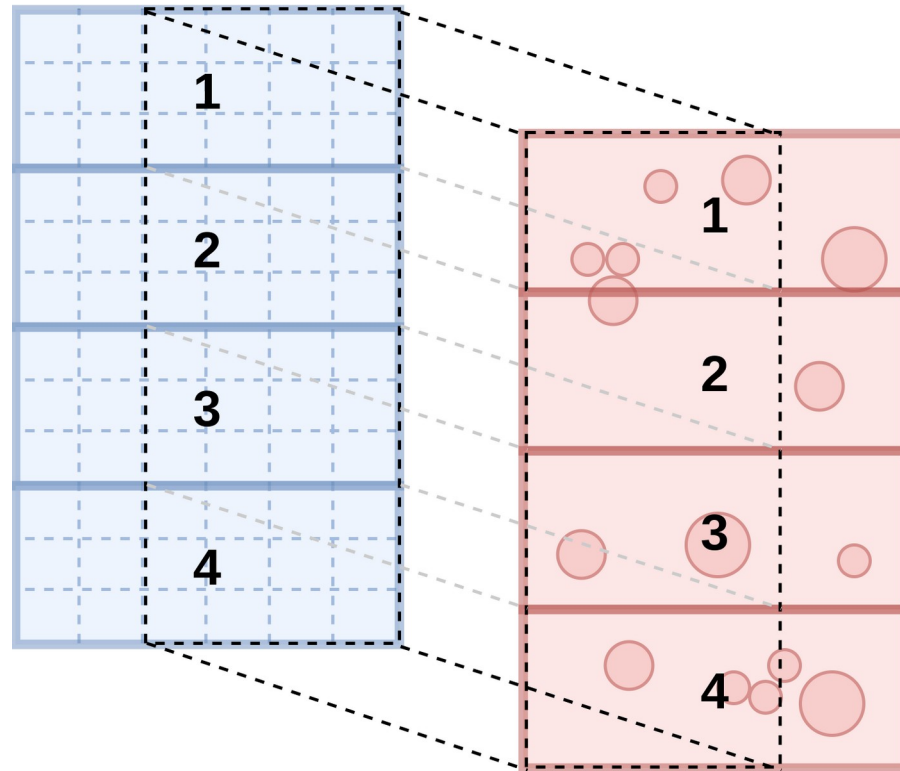
CFD-DEM Parallel Coupling: Challenges

Classical Approach: the domains are partitioned independently



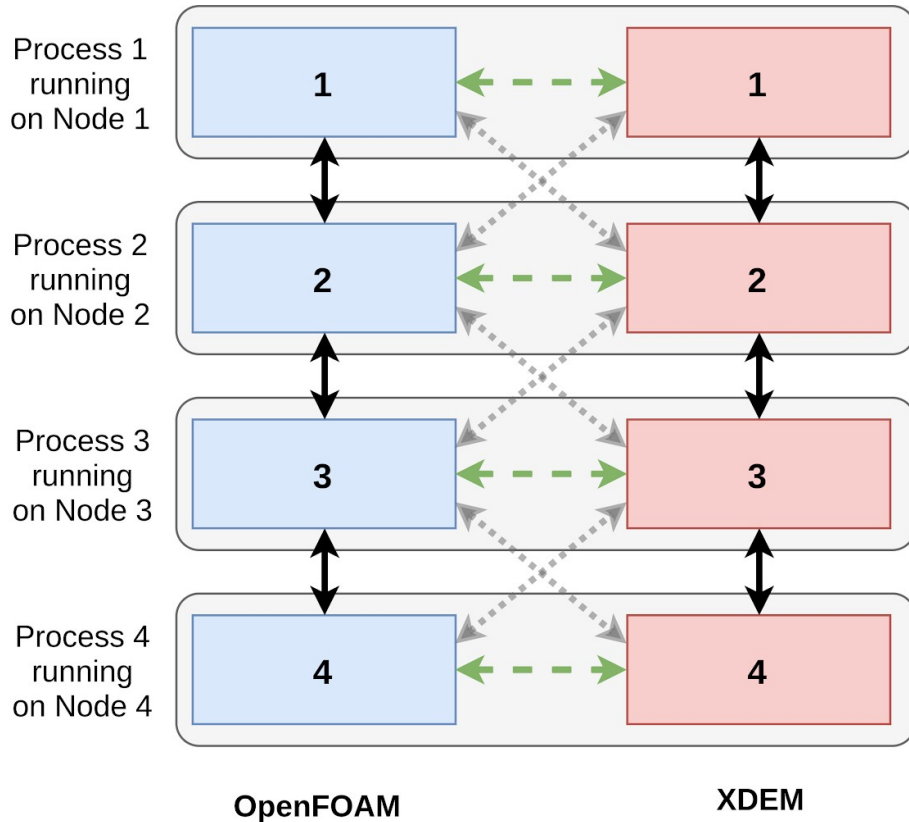
Complex pattern and large volume of communication

Co-located Partitioning Strategy



Domain elements co-located in domain space are assigned to the same partition

Co-located Partitioning Strategy: communication



Intra-physics Data Exchange



Inter-partition intra-physics Data Exchange
using MPI communication layer
with native implementation in OpenFOAM and XDEM

Inter-physics Data Exchange

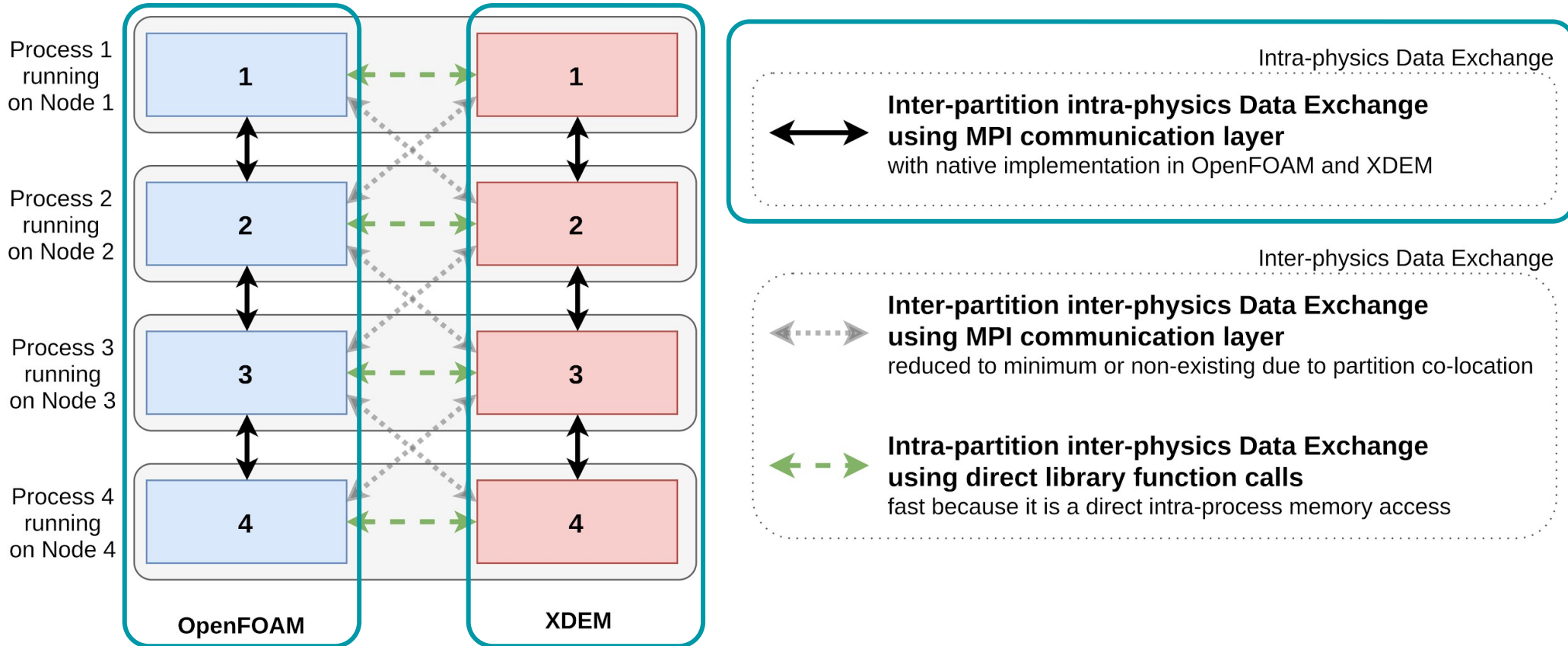


Inter-partition inter-physics Data Exchange
using MPI communication layer
reduced to minimum or non-existing due to partition co-location



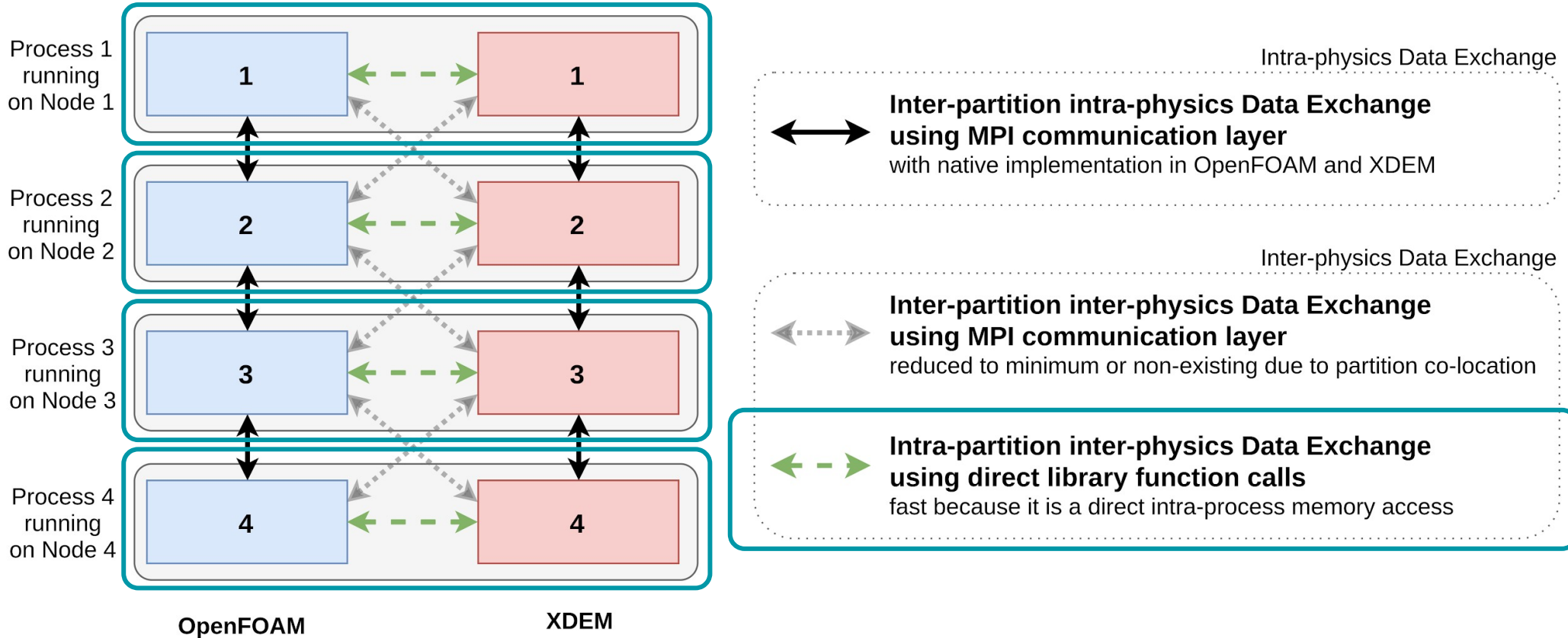
Intra-partition inter-physics Data Exchange
using direct library function calls
fast because it is a direct intra-process memory access

Co-located Partitioning Strategy: communication



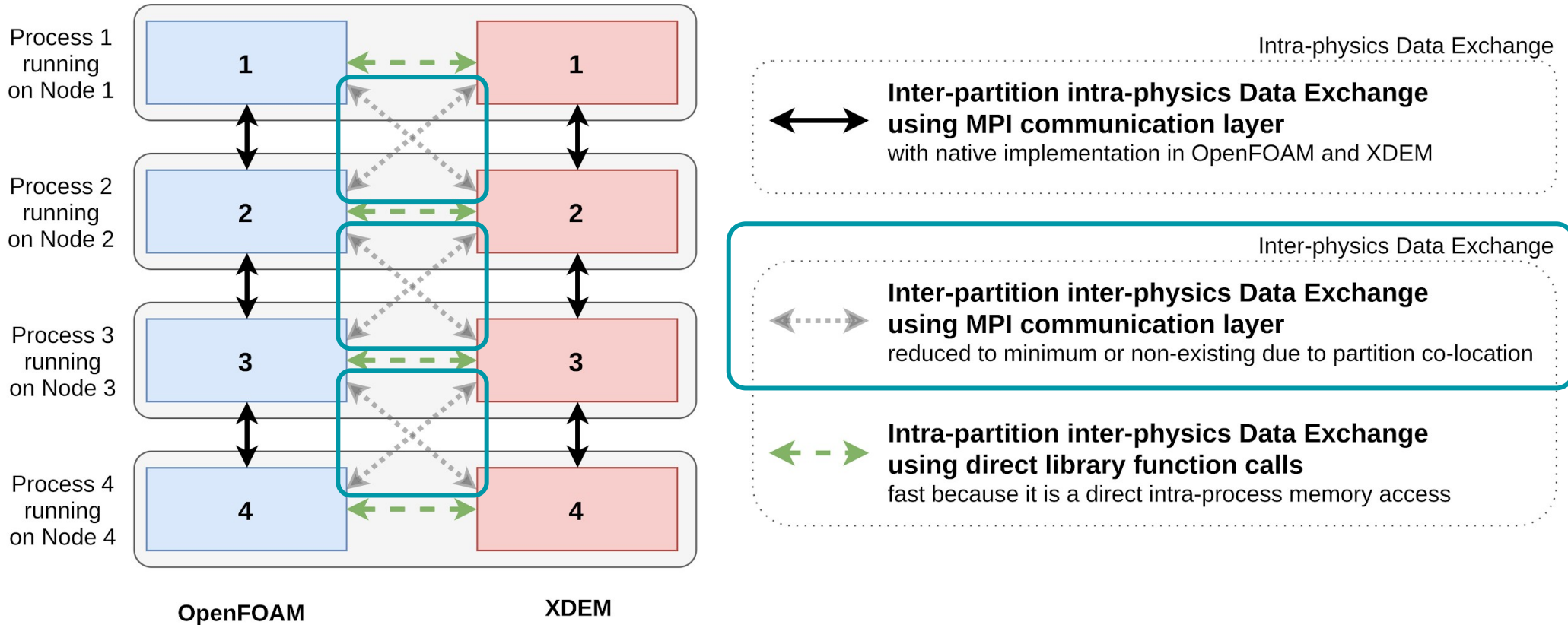
With native implementation of each software

Co-located Partitioning Strategy: communication



**Use direct intra-proces memory access
if the two software are linked into one executable,**

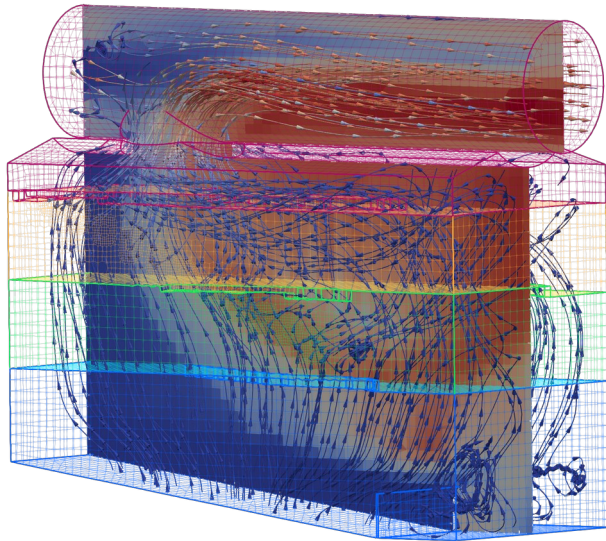
Co-located Partitioning Strategy: communication



Can be non-existing
if partitions are perfectly aligned

Volume Coupling for Biomass Furnace Simulation

Momentum, Heat and Mass transfer



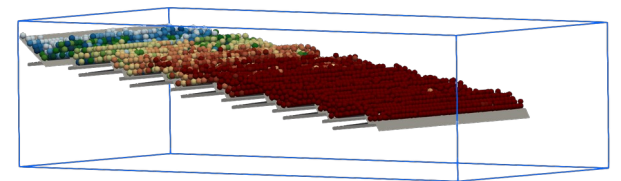
Fluid phase in OpenFOAM

CFD to DEM

- Fluid *velocity*, density, dynamic viscosity
- *Pressure gradient*
- Temperature
- Thermal conductivity
- Specific heat
- Diffusivity
- Species mass fraction (CH₄, CO₂, CO, H₂, H₂O, N₂, O₂, Tar)

- Porosity
- Momentum source (*acceleration*, ω)
- Heat source
- Mass sources (CH₄, CO₂, CO, H₂, H₂O, N₂, O₂, Tar)

DEM to CFD

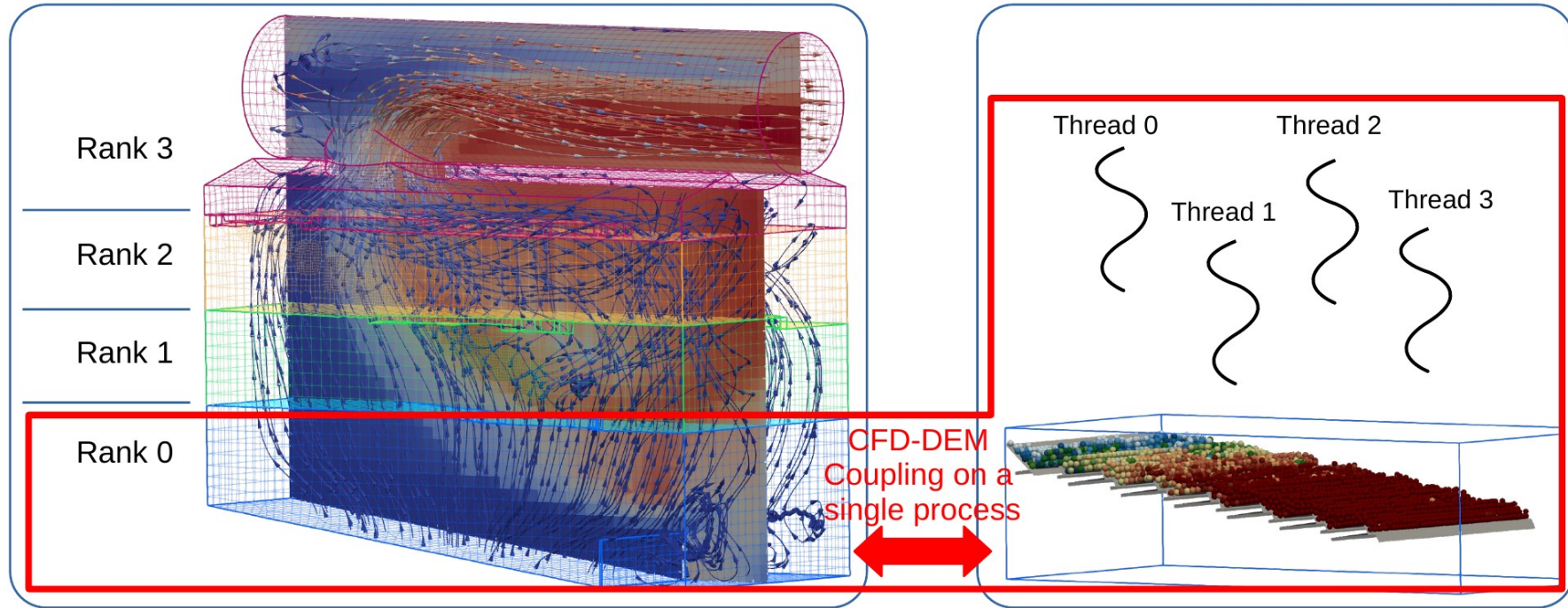


Particles in XDEM

Parallelization approach for Biomass Furnace Simulation

OpenFOAM parallelized with MPI

XDEM parallelized with OpenMP



Co-located partitioning → Account for the spatial-locality of the data between the two solvers

Overlapping domains are **co-located**
Solvers linked as one executable

⇒ No inter-partition inter-physics communication
⇒ Fast intra-partition inter-physics data exchange

Biomass Furnace Setup

based on an experimental furnace at Enerstena UAB in Lithuania

Furnace

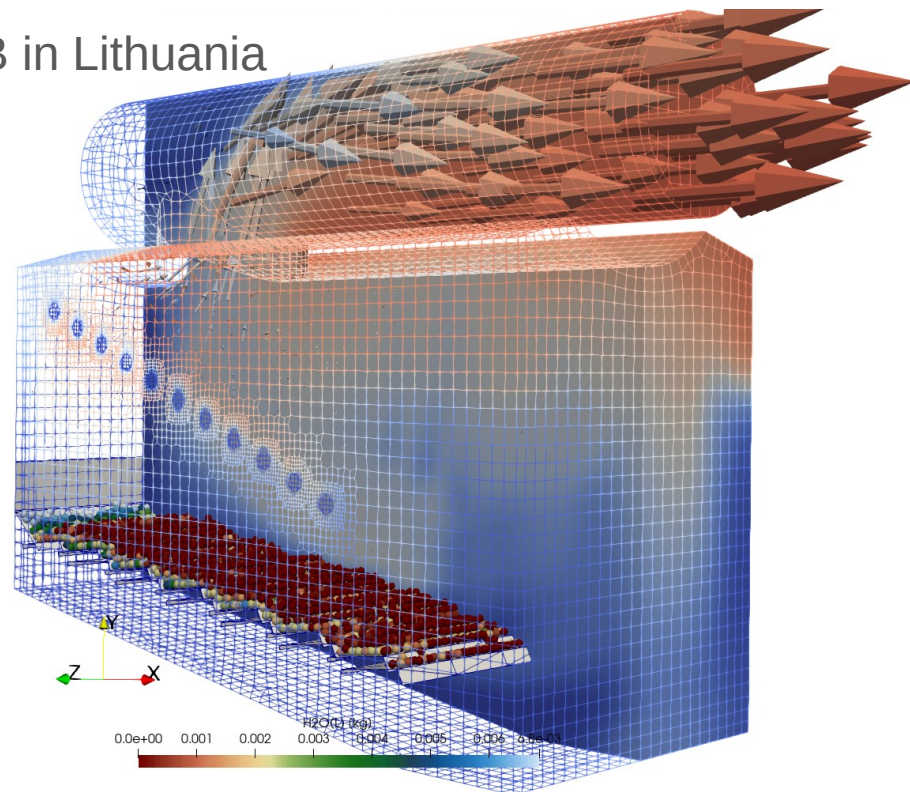
- Dimensions of 2.51m × 1.14m × 2.07m (L × W × H)
- Top exhaust pipe of 0.6m diameter
- 6 primary air inlets from the bottom
- 11 secondary air inlets on each side
- 1 tertiary air inlet on the exhaust pipe

Grates

- 8 static grates and
- 6 moving grates with an
- average slope of 7.5 degrees

Fuel bed

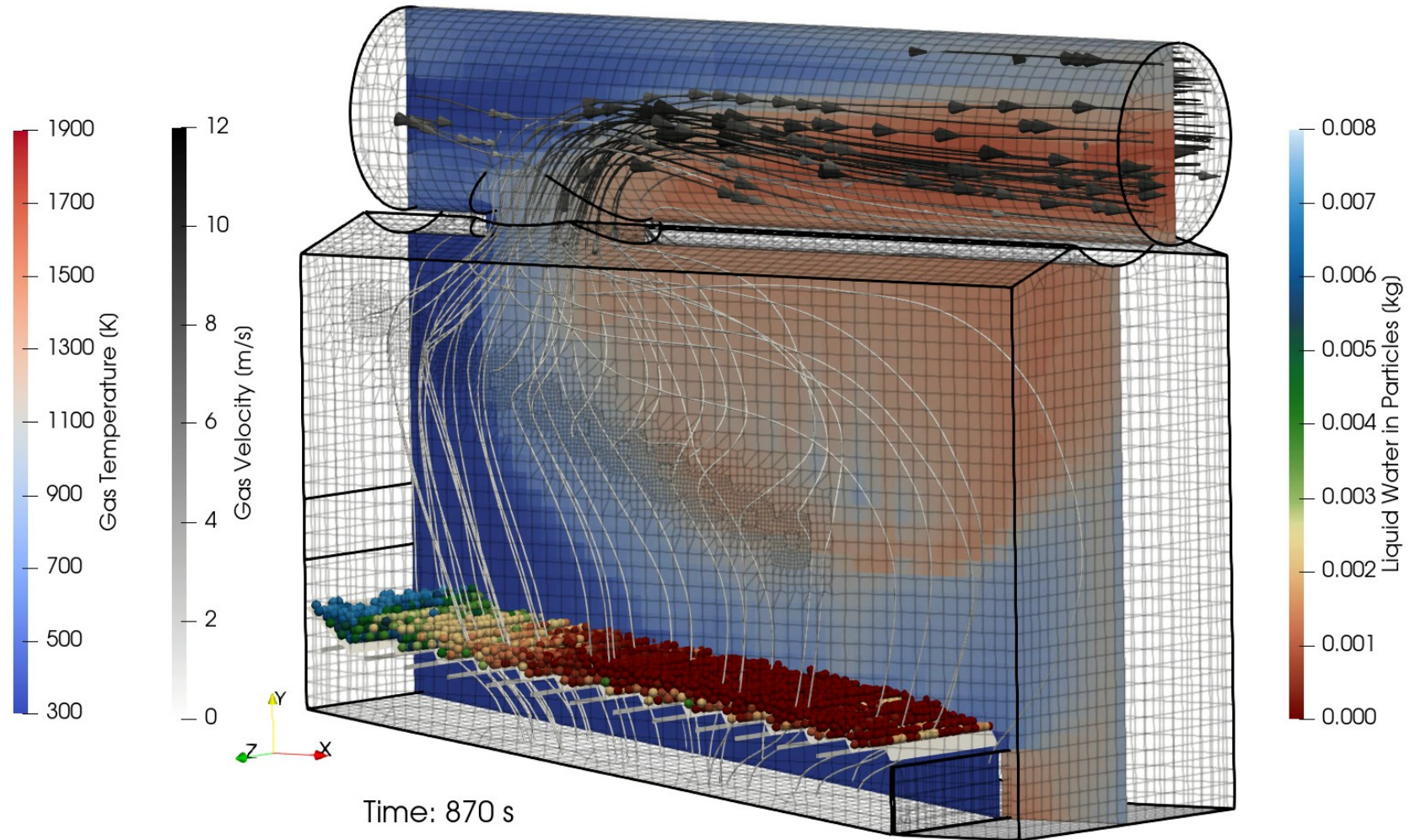
- Initial fuel bed height is 10cm
- Wood particles of 3cm diameter with 40% humidity
- Injected at the top side of the grates at a rate of 439kg/h



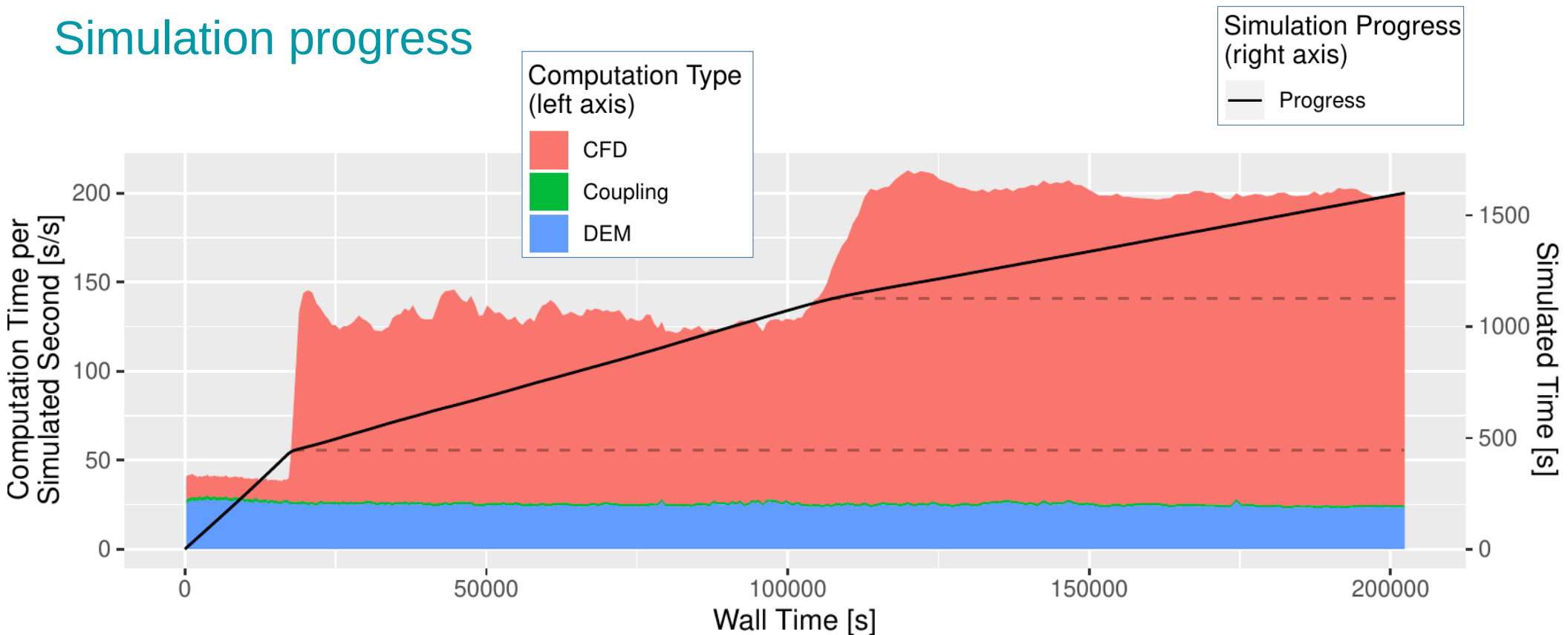
- CFD mesh with 60,001 cells
- 9,141 particles initially

Following performance measurements were carried out on the *Barbora* cluster of the IT4Innovations HPC platform.

Biomass Furnace simulation using XDEM+OpenFOAM



Simulation progress



- At 445s of simulated time, lighting-up of the furnace
- Around 1125s, furnace reaches the steady state (all hot gases are burning)

⇒ Workload between CFD and DEM changes with the simulation progress

References

References on HPC, Distributed Parallel Programming

The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software

Herb Sutter, Dr. Dobb's Journal, 30(3), 2005. <http://www.gotw.ca/publications/concurrency-ddj.htm>

Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering

Ian Foster, 1995. <https://www.mcs.anl.gov/~itf/dbpp/>

Modern Operating Systems, Andrew Tanenbaum, 1992.

The Art of High Performance Computing, Victor Eijkhout, updated in 2022, <https://theartofhpc.com/>

- Volume 1: **Introduction to High-Performance Scientific Computing**
- Volume 2: **Parallel Programming for Science and Engineering**
- Volume 3: **Introduction Scientific Programming in Modern C++ and Fortran**

Using MPI: Portable Parallel Programming with the Message Passing Interface

Gropp et al., 2014. <https://mitpress.mit.edu/books/using-mpi-third-edition>

MPI Standard. <https://www.mpi-forum.org/docs/>

OpenMP Specifications. <https://www.openmp.org/specifications/>

Tools to work with the Roofline Model

CS Roofline Toolkit, Berkeley Lab

<https://bitbucket.org/berkeleylab/cs-roofline-toolkit/>

LIKWID, RRZE-HPC

<https://github.com/RRZE-HPC/likwid>

Intel Advisor, Intel

<https://software.intel.com/en-us/advisor>

References on the Roofline Model

Roofline: An Insightful Visual Performance Model for Multicore Architectures

Williams et al., CACM, 2009. DOI: [10.1145/1498765.1498785](https://doi.org/10.1145/1498765.1498785)

Performance Tuning of Scientific Codes with the Roofline Model, Williams et al., SC'18 Tutorial, 2018

<https://crd.lbl.gov/assets/Uploads/SC18-Roofline-1-intro.pdf>

Applying the roofline model, Ofenbeck et al., ISPASS, 2014

DOI: [10.1109/ISPASS.2014.6844463](https://doi.org/10.1109/ISPASS.2014.6844463)

References on HPC for XDEM

Parallel Multi-Physics Simulation of Biomass Furnace and Cloud-based Workflow for SMEs

Besseron et al. PEARC'22, 2022. DOI: [10.1145/3491418.3530294](https://doi.org/10.1145/3491418.3530294)

Large Scale Parallel Simulation For Extended Discrete Element Method

Mainassara Chekaraou A. W., PhD Thesis, 2020. <http://hdl.handle.net/10993/46418>

Predicting near-optimal skin distance in Verlet buffer approach for Discrete Element Method

Mainassara Chekaraou et al., PDCO'20, 2020. DOI: [10.1109/IPDPSW50202.2020.00093](https://doi.org/10.1109/IPDPSW50202.2020.00093)

A parallel dual-grid multiscale approach to CFD-DEM couplings

Pozzetti et al., Journal of Computational Physics, 2019. DOI: [10.1016/j.jcp.2018.11.030](https://doi.org/10.1016/j.jcp.2018.11.030)

The XDEM Multi-physics and Multi-scale Simulation Technology: Review on DEM-CFD Coupling, Methodology and Engineering Applications, Peters et al., Particuology, 2019. DOI: [10.1016/j.partic.2018.04.005](https://doi.org/10.1016/j.partic.2018.04.005)

Hybrid MPI+OpenMP Implementation of eXtended Discrete Element Method

Mainassara Chekaraou et al., WAMCA'18. DOI: [10.1109/CAHPC.2018.8645880](https://doi.org/10.1109/CAHPC.2018.8645880)

Unified Design for Parallel Execution of Coupled Simulations using the Discrete Particle Method

Besseron et al., PARENG'13, 2013. DOI: [10.4203/ccp.101.49](https://doi.org/10.4203/ccp.101.49)

Thank you for your attention !

Question?

13th International SuperComputing Camp
<https://sc-camp.org>

SC-Camp-Hybrid 2022

Bucaramanga, Colombia

November 24-26, 2022

+

Guadalajara, México

December 8-10, 2022

+

Online



Xavier Bessonon

LuXDEM Research Team
University of Luxembourg

<https://luxdem.uni.lu>



UNIVERSITÉ DU
LUXEMBOURG