Norges miljø- og
biovitenskapelige
universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205
Resource-efficient programming

4 Concurrency

4.1 Parallel programming
4.2 Message passing interface
4.3 Domain decomposition

4.4 Robotics middleware
4.5 Concurrency theory
4.6 Parallel process models

Weekly glossary concepts

What are essential concepts from this lecture?

Let us include them in the **INF205 glossary**.[1]

Bokmål: “Samtidighet”

Nynorsk: (not found)

???

concurrency

synchronization

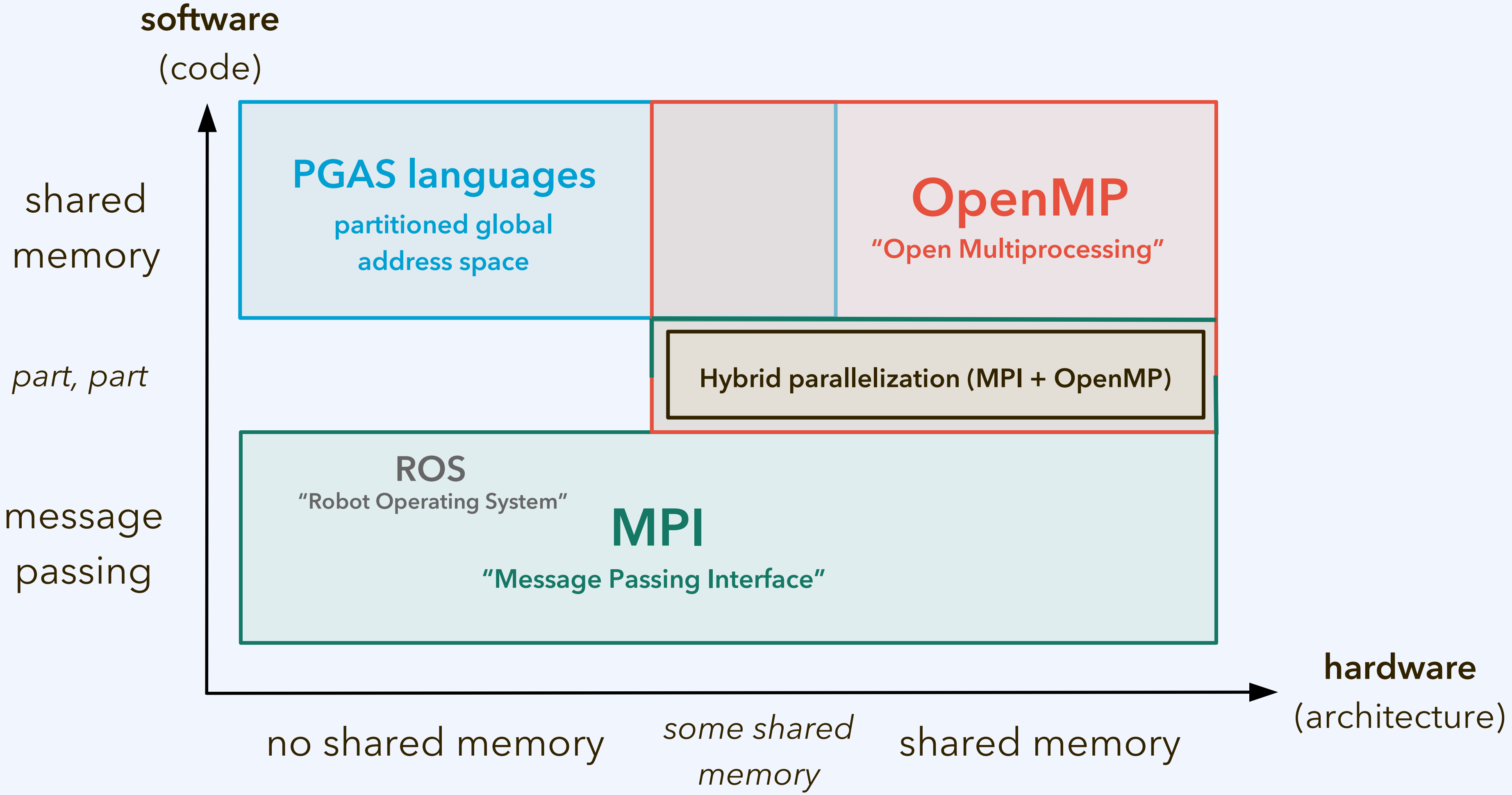
???

Amdahl’s law

???

[1] https://home.bawue.de/~horsch/teaching/inf205/glossary-en.html

Software vs. hardware architecture



MPI ping-pong example

Example file: pingpong.zip

*"increment counter, then read **1** item of type **int64_t** from **&counter**"*

*"send it **to rank 1**"*

```
if(rank == 0)
  MPI_Send(&(++counter), 1, MPI_INT64_T, 1, 1, MPI_COMM_WORLD);

if(rank == 1)
  MPI_Recv(
    &buffer, 1, MPI_INT64_T, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE
  );
```

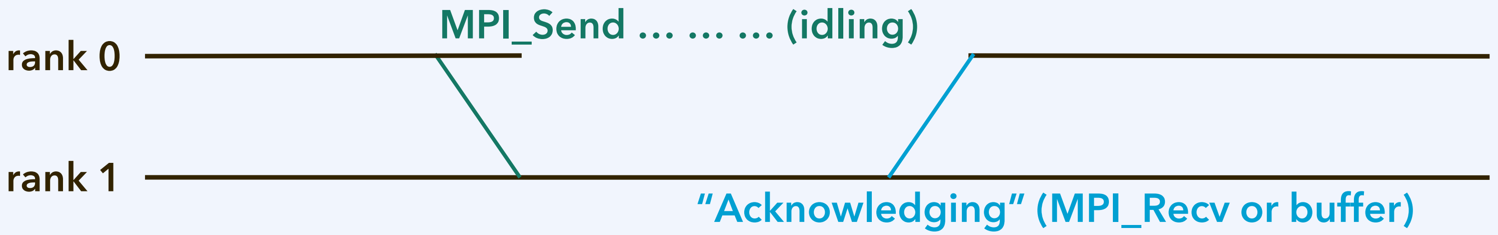
the **tag** for send and receive must be the same

*"write **1** item of type **int64_t** to **&buffer**"*

*"receive it **from rank 0**"*

One of the processes (say, rank 0) will reach the send/receive first.

Blocking communication: That process is **idle**, waiting for the other process.



Collective communication

Send/receive is done from *one sender* process to *one recipient* process.
In a **collective communication** step, *all the MPI ranks participate* jointly.

- **Broadcast:** MPI_Bcast(buffer, count, type, root, handle)
  After the broadcast, *all processes' buffers* contain the value that used to be in the buffer of the root process. Rank 0 is often used as the root process.

- **Scatter:** MPI_Scatter(content, count, type, buffer, count, type, root, handle)
  Like broadcast, but *content* is *split (scattered) over the recipients' buffers*.

- **Reduce:** MPI_Reduce(content, buffer, count, type, *operation*, root, handle)
  Content from all the processes is *aggregated* into the buffer of the root process. For example, add up all the values (with *MPI_SUM* as *operation*).

- **Gather:** MPI_Gather(content, count, type, buffer, count, type, root, handle)
  The gather operation is the *opposite of scatter*. Split content from all processes is written into one big buffer at the root process.

Collective communication

Gathering operation (all ranks to all ranks):

- MPI_Allgather(local_chunk, 3, MPI_CHAR, content, 3, MPI_CHAR, …)

Scatter operation (all ranks to all ranks):

- MPI_Allreduce(local_chunk, reduced, 3, MPI_BYTE, MPI_MAX, …)

```
Scattering content[15] to local_chunk[3].
      rank 0: 'a' 'b' 'c'
      rank 1: 'd' 'e' 'f'
      rank 2: 'g' 'h' 'i'
      …
Gathering using MPI_Allgather.
      rank 0: 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o'
      rank 1: 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o'
      rank 2: 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o'
      …
Reducing local chunks into 'reduced' using MPI_Allreduce with MPI_MAX.
      rank 0: 'm' 'n' 'o'
      rank 1: 'm' 'n' 'o'
      rank 2: 'm' 'n' 'o'
      …
```

Name	Meaning
MPI_MAX	**maximum**
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bit-wise and
MPI_LOR	logical or
MPI_BOR	bit-wise or
MPI_LXOR	logical xor
MPI_BXOR	bit-wise xor
MPI_MAXLOC	max value, location
MPI_MINLOC	min value, location

Example file: collective-communication.zip

Collective communication

What MPI operation(s) would we use for the following?

- There are n processes (ranks).
- Each rank generates $k = 65536$ floating-point random numbers between 0 and 1.
- Now there are $k \cdot n$ random numbers. We would like all of them together to become a **unit vector** $\mathbf{x} = (x_0, \ldots, x_{kn-1})$ such that $\mathbf{x}^2 = 1$.
- We definitely don't want to send all the values to all processes, especially if k becomes even greater, but do this as efficiently as possible.

Discussed MPI operations

MPI_Send	MPI_Isend
MPI_Recv	MPI_Irecv
MPI_Wait	MPI_Test
MPI_Bcast	MPI_Ibcast
MPI_Scatter	MPI_Iscatter
MPI_Reduce	MPI_Ireduce
MPI_Gather	MPI_Igather
MPI_Allgather	MPI_Iallgather
MPI_Allreduce	MPI_Iallreduce

(See **unit-vector-incomplete.cpp**, where the implementation is missing.)

Noregs miljø- og
biovitskaplege
universitet

Institutt for datavitskap

Digitalisering på Ås

4 Concurrency

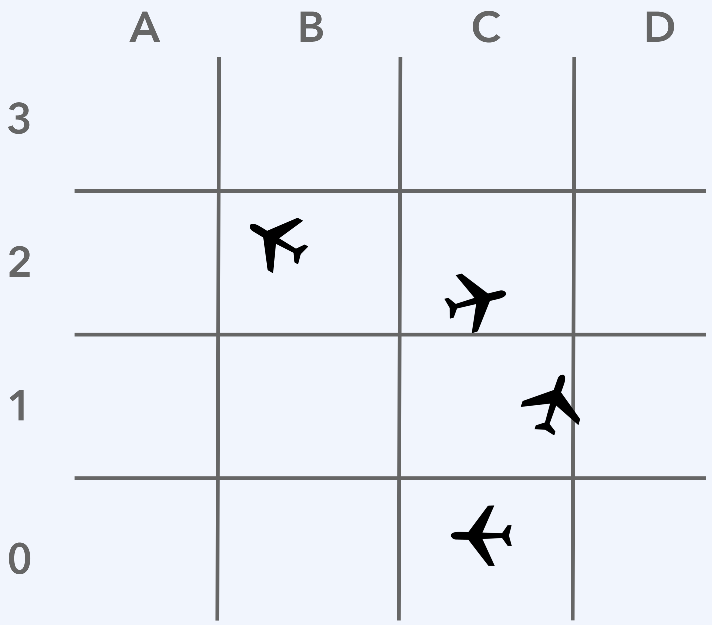
4.1 Parallel programming

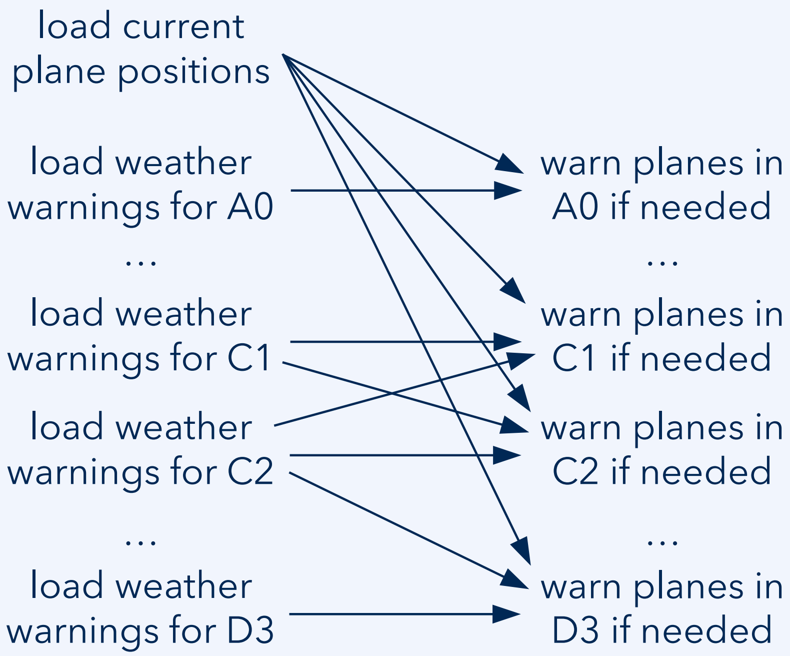
4.2 Message passing

4.3 Domain decomposition

Space-like concurrency in the data

Domain decomposition is characterized by two features:
First, parallelization is based on the concurrency inherent in (some) data.
Second, these data are seen as constituting a space, or as located in a space.





load current plane positions

load weather warnings for A0

…

load weather warnings for C1

load weather warnings for C2

…

load weather warnings for D3

warn planes in A0 if needed

…

warn planes in C1 if needed

warn planes in C2 if needed

…

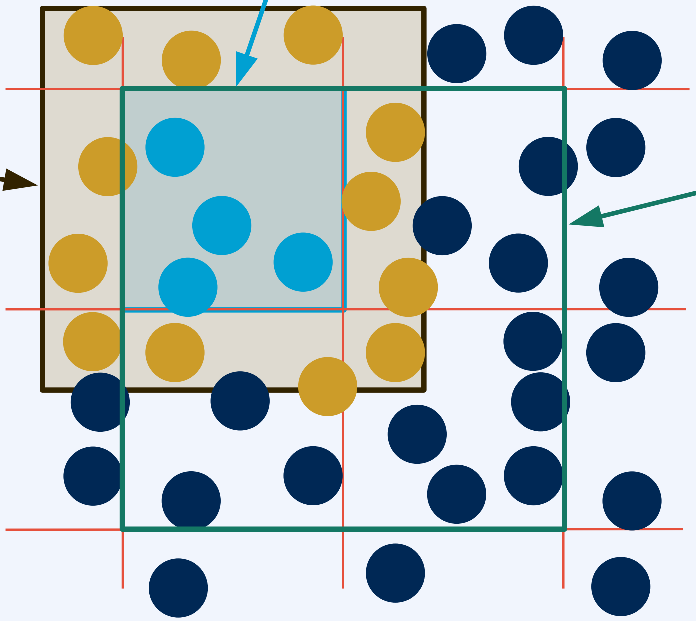
warn planes in D3 if needed

Example: Three-dimensional box

In the **sphere-config-par.zip** example, a 3D domain decomposition is implemented:

one of the **local boxes** into which the system is divided for parallelization

halo region of the loxal box: the process is not responsible for this information, but needs to know it



global 3D system containing all the “original” versions of the particles

subdomain from MPI rank:

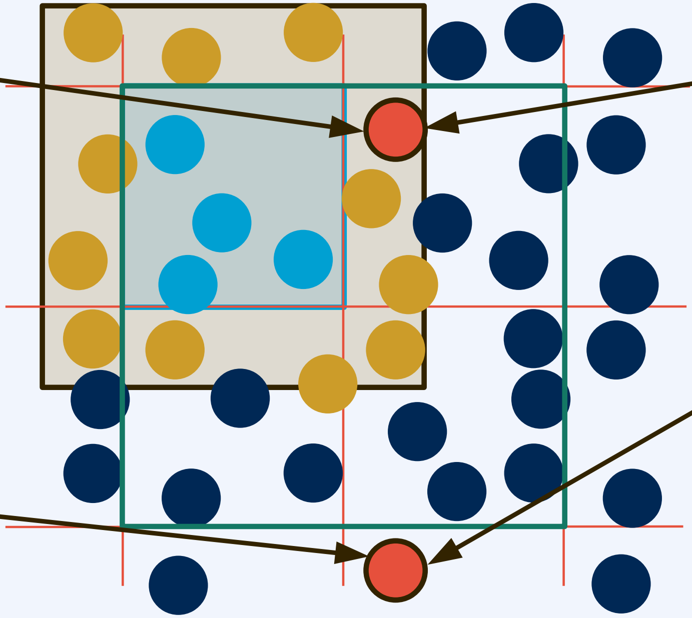
```
int remainder = rank;
boxrank[0] = remainder /
    (boxes[1] * boxes[2]);
remainder -= boxrank[0] *
   boxes[1] * boxes[2];
boxrank[1] = remainder / boxes[2];
remainder -= boxrank[1] * boxes[2];
boxrank[2] = remainder;
```

Example file: sphere-config-par.zip

Example: Three-dimensional box

2D representation here just because the slide is two-dimensional

Attention: For a single particle read in from the input file, **multiple copies** can now exist in several ranks.
(In our implementation, these have the same particle ID.)



rank 0 (top left) needs a version of this particle in its halo

rank 1 (top right) has the main responsibility for the object

rank 2 (bottom left) has a periodic copy of the particle in its halo

rank 3 (bottom right) has a periodic copy of the particle in its halo

1. If an object is *updated or moved*, adjacent ranks may need to be informed.
2. Attention *not to double-count* objects, or pairs; see Box::count_overlaps().

Example file: sphere-config-par.zip

How to run the parallel code on Orion

- Login via ssh inf205-22-xx@login.orion.nmbu.no.
- Documentation available at https://orion.nmbu.no/.
- You must be on the VPN (https://na.nmbu.no/) to access any of these.

Advice on the login process:

- Create a folder ~/.ssh on the remote system (login.orion.nmbu.no).
- Copy your local ssh public key to ~/.ssh/authorized_keys.
  - If you don't have one, create it with the command ssh-keygen.
- You can now use ssh and scp without entering your password.
- For all temporary storage on Orion, use the folder $SCRATCH.

Sample instructions: protocol-orion_ssh.txt

How to run the parallel code on Orion

- Login via ssh inf205-22-xx@login.orion.nmbu.no.
- Documentation available at https://orion.nmbu.no/.
- You must be on the VPN (https://na.nmbu.no/) to access any of these.

Modules: On Orion, you need to load modules to select your favourite environment. To load OpenMPI, the command is module load OpenMPI.

Scratch folder and home folder: Do not use your home directory for single-use files or any very large data. These should go on $SCRATCH.

Don’t run jobs on the login node: Never make this mistake! It will slow down all other users’ work on the login node, and they will get angry.

Do run jobs via bash scripts (batch files) and the **submission** command **qsub**.

Sample instructions: protocol-orion_compile-and-run.txt

How to run the parallel code on Orion

The following file can be submitted to the Slurm scheduler using “**qsub**”.
See also the overview file: pbs-to-slurm-translation-sheet.pdf.

```
#!/bin/bash
#SBATCH --tasks-per-node=24
#SBATCH --nodes=2
#SBATCH --time=00:01:00
#SBATCH --job-name=sphere-test-job
#SBATCH --partition=smallmem
#SBATCH --mail-user=XXXXXX.XXXXXX@nmbu.no
#SBATCH --mail-type=ALL

cd /mnt/SCRATCH/inf205-2024-XX/sphere-test-job

module load OpenMPI

mpirun -np 48 /mnt/users/inf205-2024-XX/bin/eval-par 32768-particles.dat 3.334 3 4 4
```

Example file: sphere-test-job.qsub

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

Fifth worksheet

INF205

8th April 2024

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Concurrency

4.1 Parallel programming

4.2 Message passing

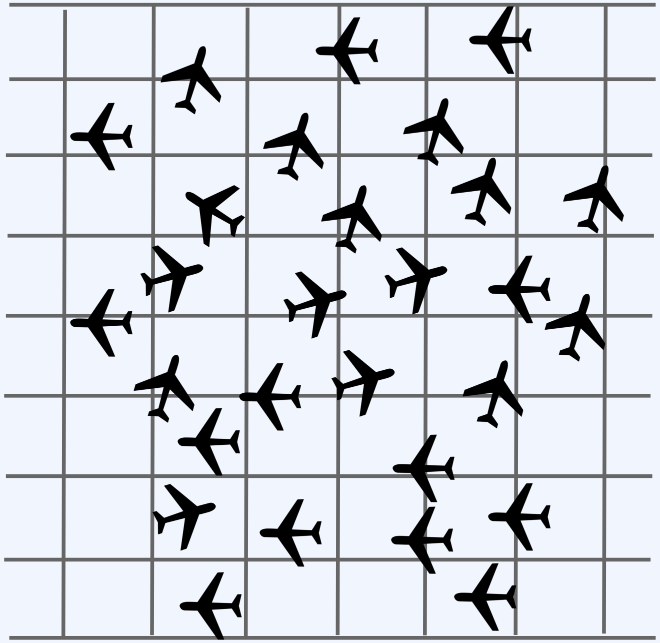
4.3 Domain decomposition

related technique: Linked cells

Linked cell data structure

Objective: Deal with interactions between objects that are close to each other ("short-range interactions") in a Cartesian space, without testing O(n^2) pairs.

Idea: Divide an area or volume into interconnected cells, and sort interacting objects into these cells according to their coordinates.



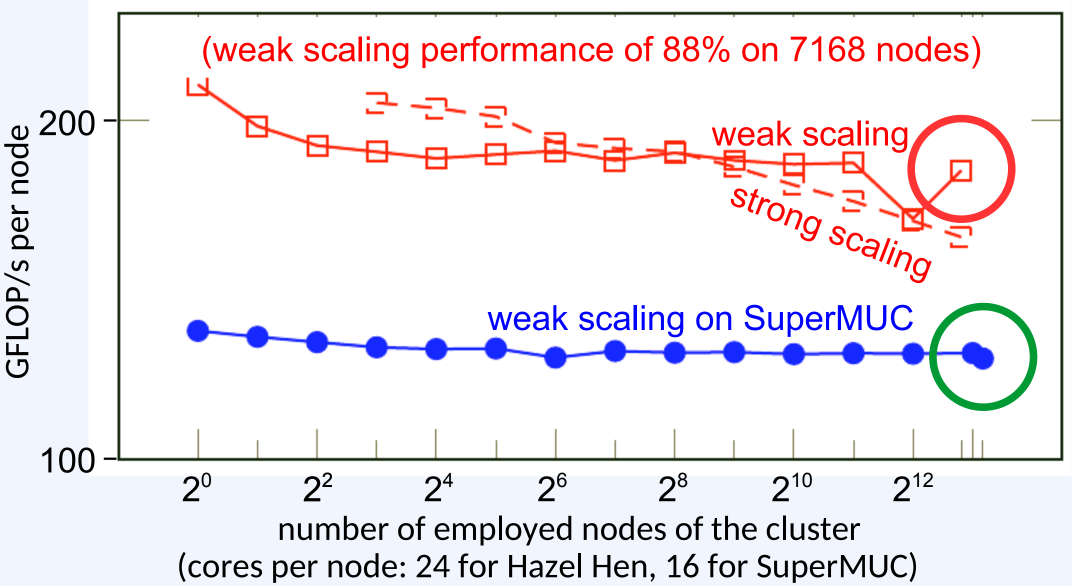
Assuming that the density of objects has an upper bound to to the nature of the problem, processing **all interacting pairs** is now **in O(n) instead of O(n^2)**, once the objects are in cells.

Sequentially, with a single process, this works just as well as in parallel. Being connected by the same logic, it is very common to *combine linked cells with domain decomposition* for particle-based methods.

Molecular dynamics world record

Hazel Hen (Stuttgart):
Haswell architecture

http://www.ls1-mardyn.de/ (large systems 1: molecular dynamics)



N = 21 000 000 000 000

2019 molecular dynamics world record[1]

Hazel Hen
weak scaling

[1] N. Tchipev, S. Seckler, M. Heinen, J. Vrabec, F. Gratl, M. Horsch, M. Bernreuther, C. W. Glass, C. Niethammer, N. Hammer, B. Krischok, M. Resch, D. Kranzlmüller, H. Hasse, H.-J. Bungartz, P. Neumann, Int. J. HPC Appl. 33(5), 838 – 854, **2019**.

Computational
Molecular Engineering

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Concurrency

4.1 Parallel programming

4.2 Message passing

4.3 Domain decomposition

… or embarassing parallelism?

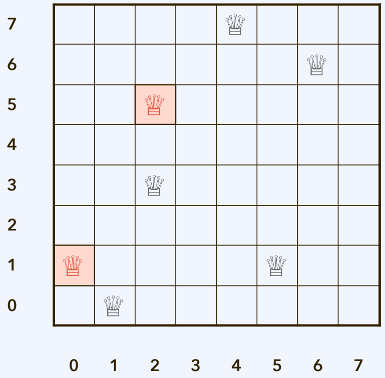
Depth-first search + backtracking

Consider the chessboard example.

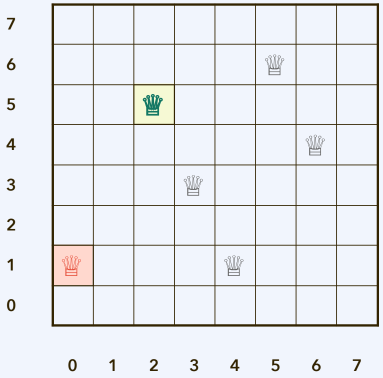
- A typical domain decomposition would split up the board into regions.
- Instead, we can split up the state space,[1] the space of configurations.

Assume that we have 2^k MPI ranks, for example, four ranks.
Now select $k = 2$ fields at random, for example A = (2, 5) and B = (0, 1).

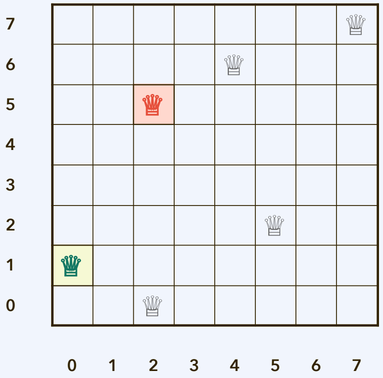
MPI rank 0: No queens on A and B.



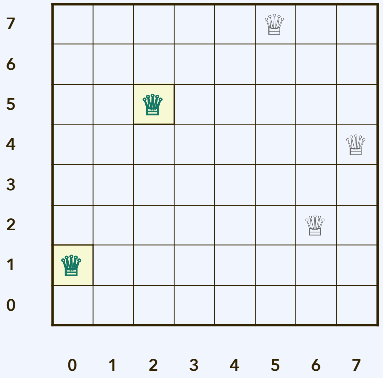
MPI rank 1: Queen on A, but not on B.



MPI rank 2: Queen on B, but not on A.



MPI rank 3: Queens on both A and B.



[1] The state space can be called a *configuration space*. That is particularly the case when it consists of variables describing the positions of objects.

Monte Carlo (MC) methods

High-dimensional state spaces: If many variables q_1, …, q_k are needed to describe a system's state, this means that the **state space**[1] is **high-dimensional**.

- In our problem with N spherical particles, we are exploring a $3N$-dimensional configuration space.
- For N queens on a board, the configuration space is $2N$-dimensional.

In Monte Carlo (MC) methods, these different variables are typically fused into one high-dimensional vector $\mathbf{q}$, the **configuration**.

MC methods are efficient at solving an otherwise untractable problem: The **average value** of some quantity $f(\mathbf{q})$ **over the whole state space**.

[1] The state space can be called a *configuration space*. That is particularly the case when it consists of variables describing the positions of objects. In statistical mechanics, it is common to consider *positions and momenta (or velocities) together* as what is called the *phase space*.

Monte Carlo (MC) methods

Most elementary and **original MC method**: Select *m* independent, uniformly distributed random sample configurations out of the state space.

- The result is the average of $f(\mathbf{q})$ over the random samples, computed as an approximation for the average over the whole state space.
- Classical illustration of the idea: Opinion poll done on random people.

MC methods are efficient at solving an otherwise untractable problem: The **average value** of some quantity $f(\mathbf{q})$ **over the whole state space**.

- Example: **How many threats between queens** are there **on average**?
- How would we need to change the queens-count-threats.zip code to do this? How much more efficient is the MC approach, compared to going through all the possible configurations?

Problems addressed by MC methods

Most elementary and **original MC method**: Select m independent, uniformly distributed random sample configurations out of the state space.

- The result is the average of $f(\mathbf{q})$ over the random samples, computed as an approximation for the average over the whole state space.
- Classical illustration of the idea: Opinion poll done on random people.

Often we are interested in **weighted averages**, looking at a quantity $\rho(\mathbf{q})\, f(\mathbf{q})$. The weight function $\rho(\mathbf{q})$ can have a role such as the probability of a state.[1] Then it can happen that almost all values from a uniform sample have $\rho(\mathbf{q}) \approx 0$.

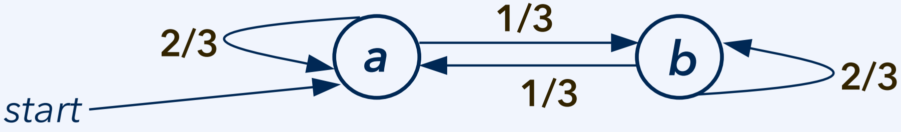
In such cases, the **Metropolis method** is used:

- Change configuration $\mathbf{q}$ by a random small amount, yielding some $\mathbf{q}'$.
  - Accept the change with 100% probability if $\rho(\mathbf{q}') > \rho(\mathbf{q})$.
  - Otherwise, accept the change with the probability $\rho(\mathbf{q}') \,/\, \rho(\mathbf{q})$.

[1] Then, $\rho(\mathbf{q})$ is called the *density* of the state space, or the configuration or phase space density.

Markov chains: Monte Carlo method(s)

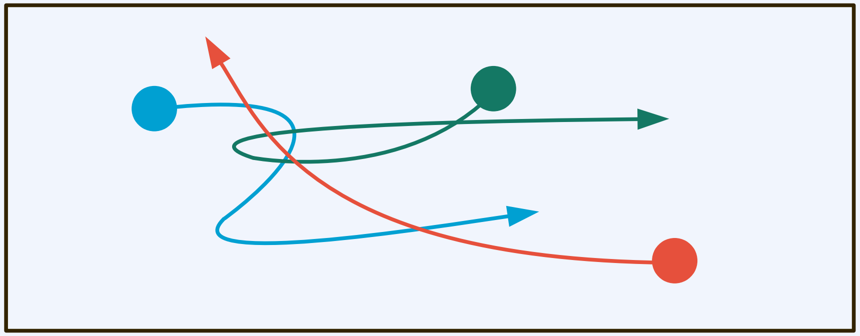
A Markov chain is a sequence of states in a probabilistic discrete event system.



abbabbbaaba …
ababbaaabba …

The sequence of configurations in a Metropolis Monte Carlo simulation is such a Markov chain. So are many variants of it, or other common solutions to problems that require the **stochastic exploration** or sampling of a **large space**.

The *concurrency* here is due to that *multiple Markov chains are independent*.



Processes/threads can explore the state space separate from each other. They work with independent configurations. It is not necessary to implement a domain decomposition.

MC simulation as a sampling technique

The MC method comes from statistical mechanics, where systems are often considered at a given temperature *T*. In statistical mechanics, the phase space density of a system at thermal equilibrium with its surroundings (constant *T*) is

$$\rho(\mathbf{q}) = \exp(-U / T),$$

where $U(\mathbf{q})$ is the internal energy as a function of the system's state. When only positions are considered (not velocities), this is the **potential** energy $U^{\mathrm{pot}}(\mathbf{q})$.

This can be used as a sampling technique even where "temperature" has no natural meaning. The temperature then becomes a parameter of the method.

Discussion: Can we use this to speed up finding a desired configuration on a chess board? We need to select a potential U^{pot}, such as the number of threats.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

E-R diagrams on draw.io

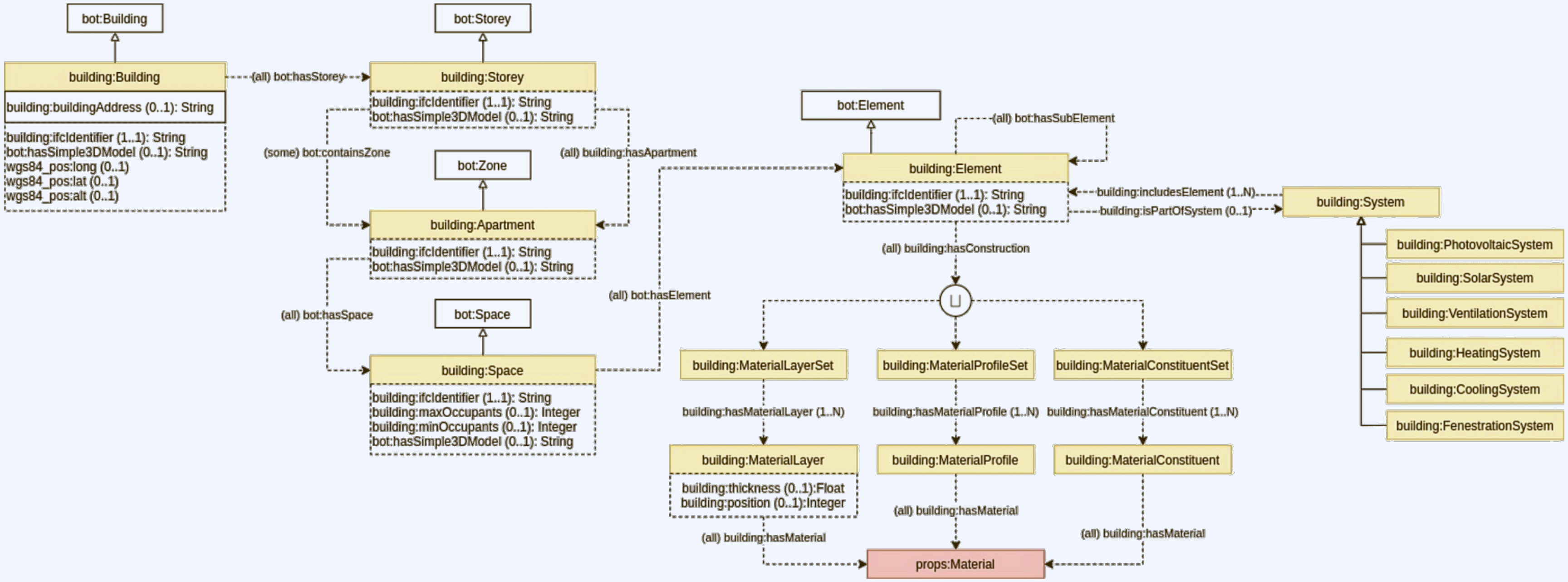
INF205

8th April 2024

E-R diagrams on draw.io and Chowlk[1, 2]

The draw.io tool can be used for E-R diagrams using a variety of conventions.

With Chowlk by Poveda Villalón *et al.*,[1, 2] these can be converted to ontologies.


bot:Building
building:Building
building:buildingAddress (0..1): String
building:ifcIdentifier (1..1): String
bot:hasSimple3DModel (0..1): String
wgs84_pos:long (0..1)
wgs84_pos:lat (0..1)
wgs84_pos:alt (0..1)
(all) bot:hasStorey
bot:Storey
building:Storey
building:ifcIdentifier (1..1): String
bot:hasSimple3DModel (0..1): String
(some) bot:containsZone
(all) building:hasApartment
bot:Zone
building:Apartment
building:ifcIdentifier (1..1): String
bot:hasSimple3DModel (0..1): String
(all) bot:hasSpace
bot:Space
building:Space
building:ifcIdentifier (1..1): String
building:maxOccupants (0..1): Integer
building:minOccupants (0..1): Integer
bot:hasSimple3DModel (0..1): String
(all) bot:hasElement
bot:Element
(all) bot:hasSubElement
building:Element
building:ifcIdentifier (1..1): String
bot:hasSimple3DModel (0..1): String
building:includesElement (1..N)
building:isPartOfSystem (0..1)
building:System
building:PhotovoltaicSystem
building:SolarSystem
building:VentilationSystem
building:HeatingSystem
building:CoolingSystem
building:FenestrationSystem
(all) building:hasConstruction
⊔
building:MaterialLayerSet
building:MaterialProfileSet
building:MaterialConstituentSet
building:hasMaterialLayer (1..N)
building:hasMaterialProfile (1..N)
building:hasMaterialConstituent (1..N)
building:MaterialLayer
building:thickness (0..1):Float
building:position (0..1):Integer
building:MaterialProfile
building:MaterialConstituent
(all) building:hasMaterial
(all) building:hasMaterial
(all) building:hasMaterial
props:Material

[1] M. Poveda Villalón *et al.*, in *Proc. VOILA23*, *CEUR Works. Proc.* **3508**: 2 (link to paper), **2023**.

[2] Chowlk template: https://chowlk.linkeddata.es/static/resources/chowlk-library-complete.xml
Lightweight version: https://chowlk.linkeddata.es/static/resources/chowlk-library-lightweight.xml

Noregs miljø- og
biovitskaplege
universitet

Institutt for datavitskap

Digitalisering på Ås

Conclusion

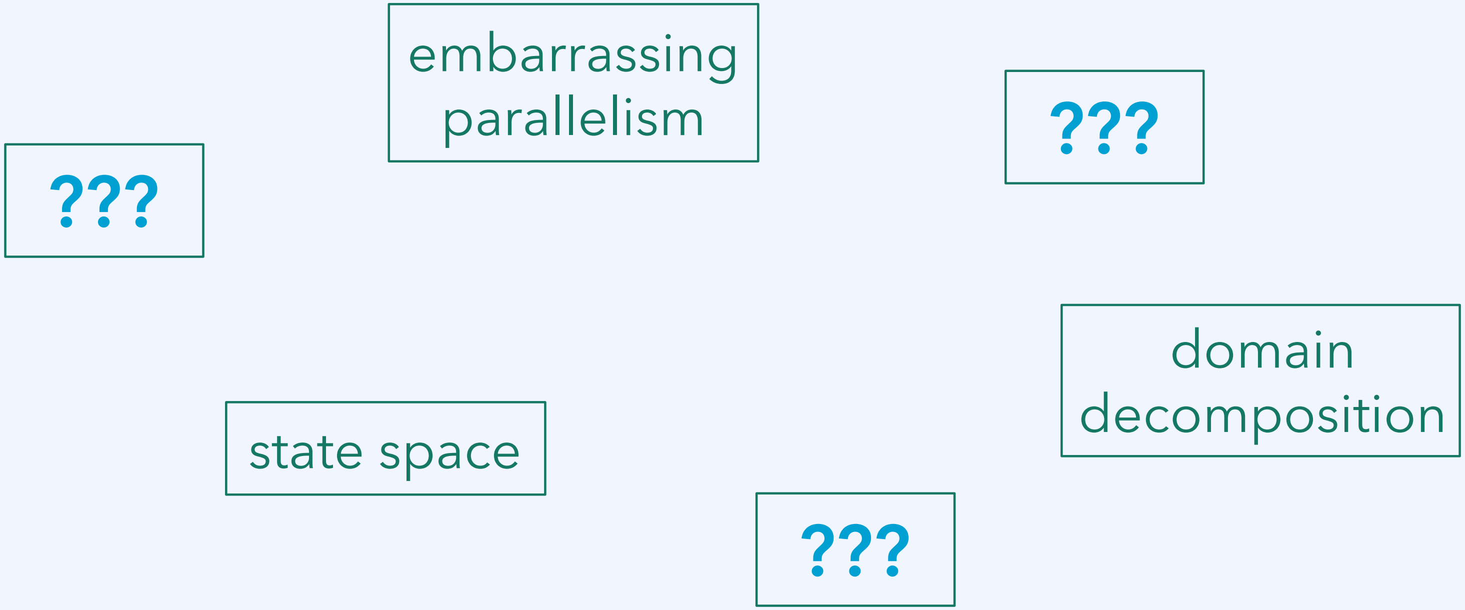
INF205

8th April 2024

Weekly glossary concepts

What are essential concepts from the previous lecture?

Let us include them in the **INF205 glossary**.[1]



embarrassing parallelism

???

???

domain decomposition

state space

???

[1] https://home.bawue.de/~horsch/teaching/inf205/glossary-en.html

Norges miljø- og
biovitenskapelige
universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205
Resource-efficient programming

4 Concurrency

4.1 Parallel programming

4.2 Message passing interface

4.3 Domain decomposition

4.4 Robotics middleware

4.5 Concurrency theory

4.6 Parallel process models