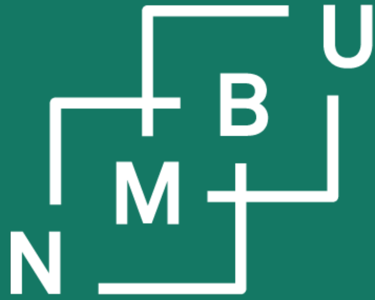
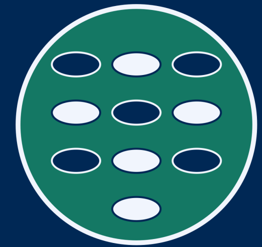


Norges miljø- og biovitenskapelige universitet



INF205
Resource-efficient programming

5 Parallel data

5.1 Domain decomposition
5.2 Linked cells
5.3 Message passing serialization
5.4 Parallel input/output

23rd November 2022

Reuse of external code

Norwegian University of Life Sciences

Are you legally allowed to use the external code?

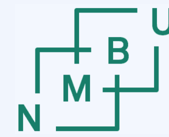
- You need a license; which is it? Check its terms and conditions.
  - Some licenses, even if they allow you to reuse the code and create derivative works, cannot be combined with each other.
  - For example, the GPL and CC NC licenses cannot be combined.
  - To alleviate this issue, libraries are often released under the LGPL.

How about the code examples from the INF205 lecture material?

- Released under the conditions of the CC BY-NC-SA 4.0 License.

Would it not be plagiarism or fraud to submit others' material?

- It is, if you submit others' developments *as if they were your own*.
- If it is not absolutely clear from your submission that you are reusing somebody else's work (when you actually are), it may be a fraud attempt.
- That is also the case for the lecture material; it must be clear also to the second examiner, who is external, that it is others' material being reused.



Norwegian University of Life Sciences

Reuse of external code (libraries)

Should we use external libraries, or should we develop all from scratch?

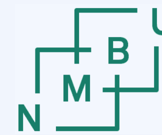
- It is one of the learning outcomes to work with external libraries.
- But we have seen that even the STL can be sometimes beaten by simple bespoke code that you write yourself for a special purpose.
- With your project code you are meant to demonstrate what you have learnt. Your own development must not be totally trivial.

If you use a library solution for something that *can also be done in a simple way* by hand (and is roughly on scope for INF205), why not *try out both* an own implementation and the library, comparing their performance?

But if you are reusing a complicated algorithm, data structure, or file format, going beyond the content of INF205, and there is a library, just use the library!

Examples: Balanced trees, graphics formats, Fast Fourier Transform, …

More questions about project work



Norwegian University of Life Sciences

Makefiles not working under Windows:

- Sad but true. (That's why production code does not come with a Makefile.)

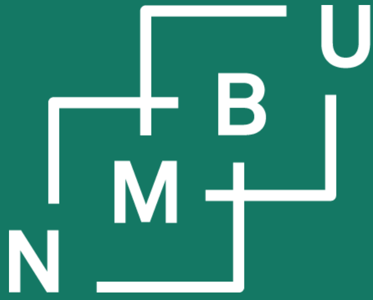
Where do we find our Orion account/login data?

- You should have received them as a comment to week 43 group status.
- Your login name is inf205-22-xx, where xx is your group number.
- 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.

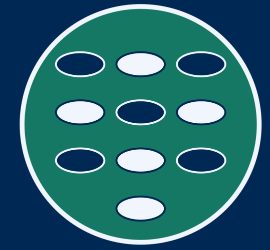
Almost final opportunity to clarify issues about the group projects at a meeting where we are all together. What information is urgently needed?

What would we need to discuss together right now?

What is still unclear but might be clarified over the coming few days?



Noregs miljø- og biovitskaplege universitet

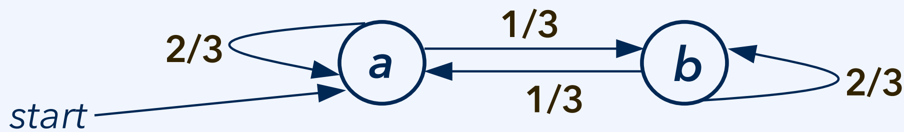


5 Parallel data

5.1 Decomposition schemes

Concurrent Markov chains

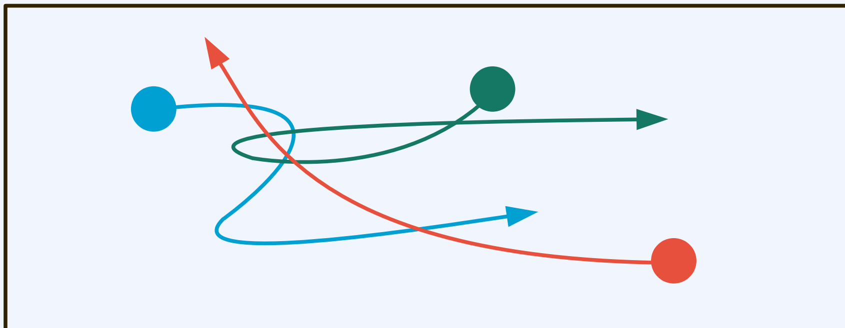
A Markov chain is a sequence of states in a probabilistic discrete event system.



abbabbbaaba …
ababbaaabba …

Some problems deal with **stochastic exploration** or sampling of a **large space**.

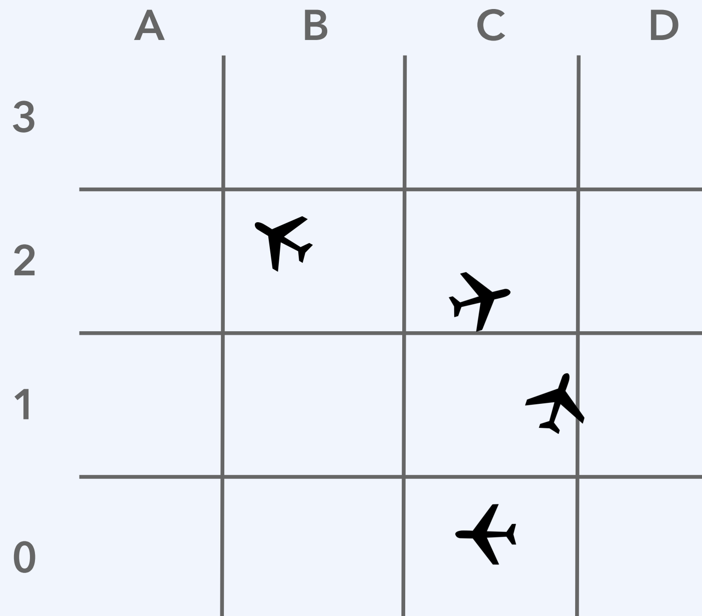
For example, in our problem with N spherical particles, we are exploring a $3N$ dimensional configuration space. In such a case, **Monte Carlo methods** can exploit *concurrency* from the fact that *multiple Markov chains are independent*.

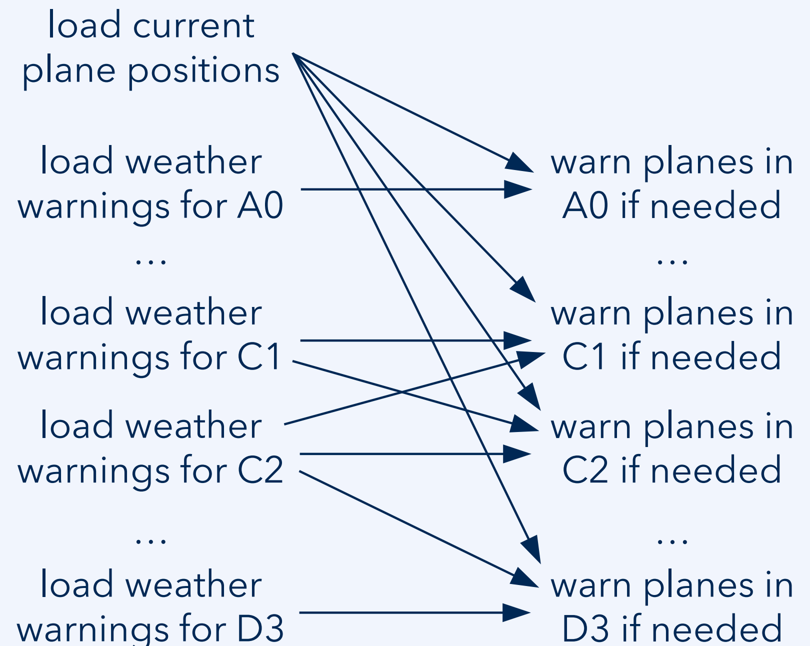


Different processes can explore the huge space completely separate from each other. All processes can access the whole space – it is not split up into subregions.

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.





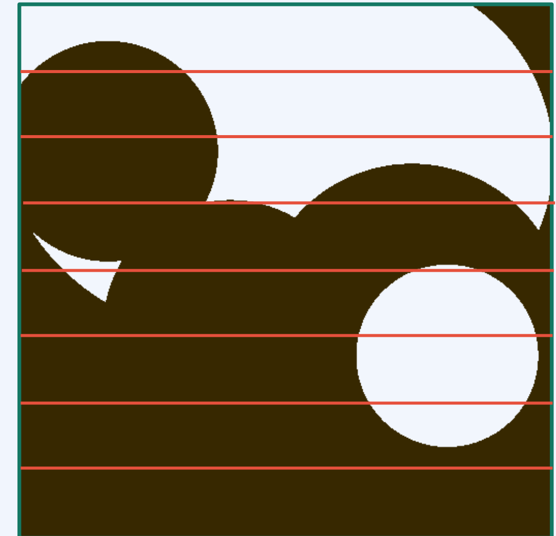
Domain decomposition can be applied to each of the INF205 project topics!

Norwegian University of Life Sciences

Example: Two-dimensional landscape

In the **charmap-output** example, generation of a random benchmark image is parallelized by domain decomposition, dividing the square shape into stripes:

$a \cdot a$ pixel image



```
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

int ny = 1 + (a-1)/size;
int yoffset = rank*ny;
if(rank == size-1) ny = a - yoffset;

diskgraphics::Charmap cm(dv, 0, a, yoffset, ny);
```

round up a/size (pointing to `1 + (a-1)/size`)

Why is this better than rounding down?

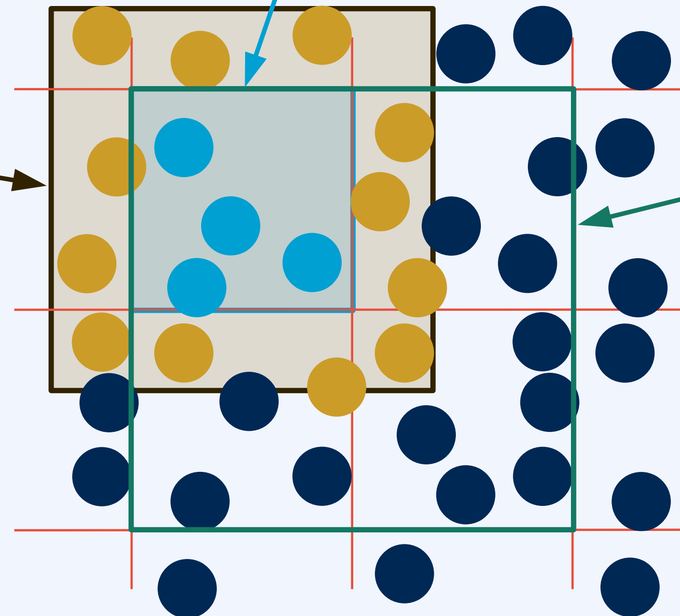
Each process allocates a *rectangular character map* (stripes, see above) and computes *only the corresponding pixel values* from the vector of circular disks.

Example: Three-dimensional box

In the **sphere-config-par** 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

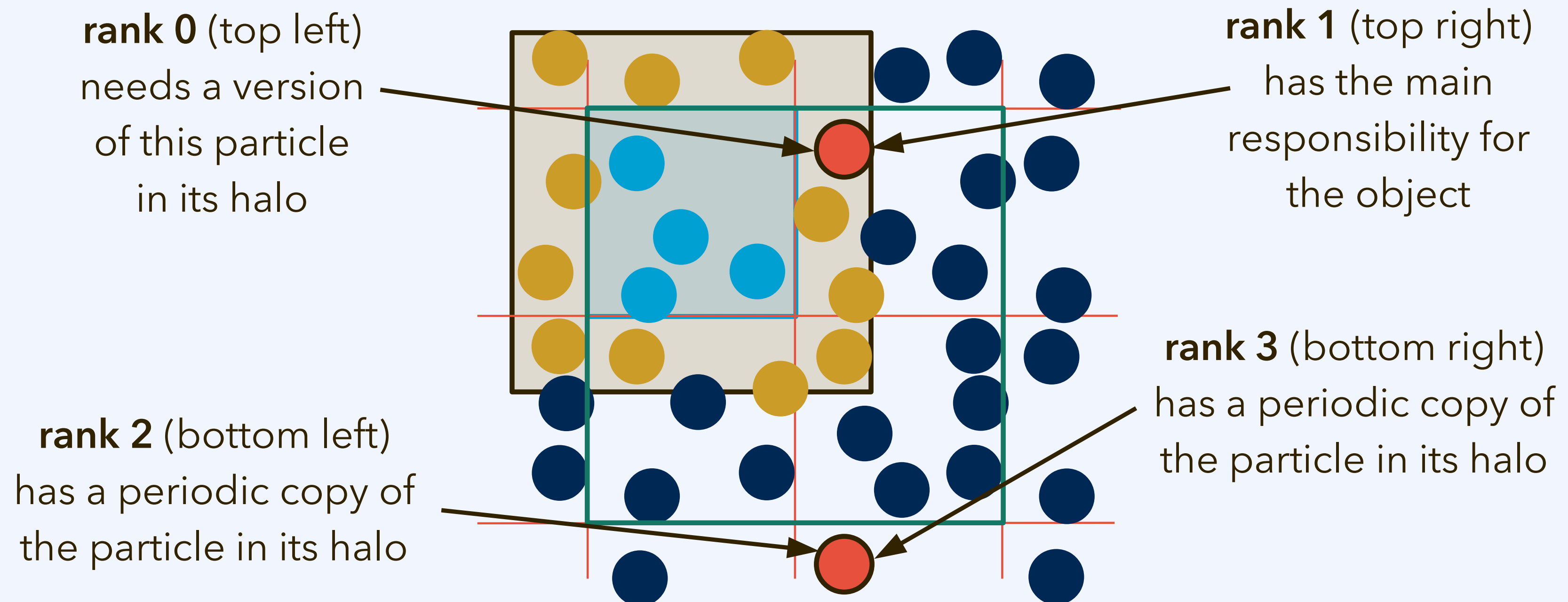
subdomain from MPI rank:

```
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: 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.)



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: Three-dimensional box

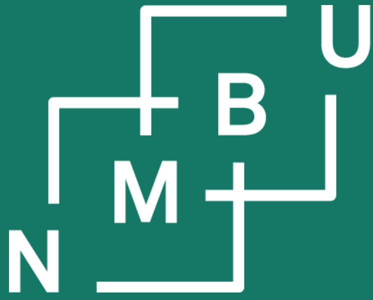
Let us do a straightforward performance test with 2^{15} particles:

- Using the sequential version compiled from the same code base
  - ./eval-seq 32768-particles.dat 3.334
- Using the parallel version, but with only one MPI rank
  - mpirun -np 1 ./eval-par 32768-particles.dat 3.334 1 1 1
- Scale up to eight ranks, on the presentation laptop
  - mpirun --oversubscribe -np 8 ./eval-par 32768-particles.dat 3.334 2 2 2
- Scale up to 18 ranks, on the presentation laptop
  - mpirun --oversubscribe -np 18 ./eval-par 32768-particles.dat 3.334 3 3 2

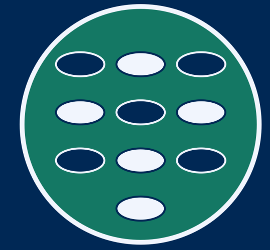
Discussion #1: How can we explain the observed behaviour?

We have looked into *domain decomposition* in detail. These methods all have in common that the *responsibility for the data items is split up* in some space.

Discussion #2: What other kinds of decomposition schemes can you think of?



Noregs miljø- og biovitskaplege universitet



5 Parallel data

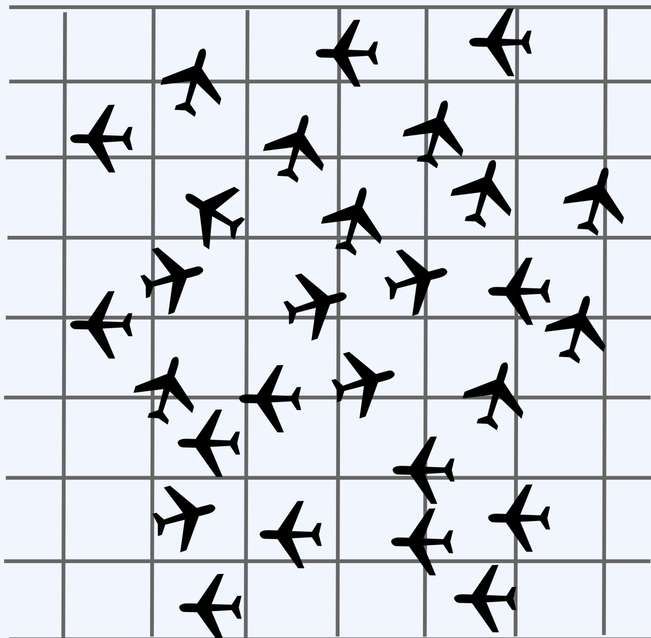
5.1 Decomposition schemes

5.2 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.

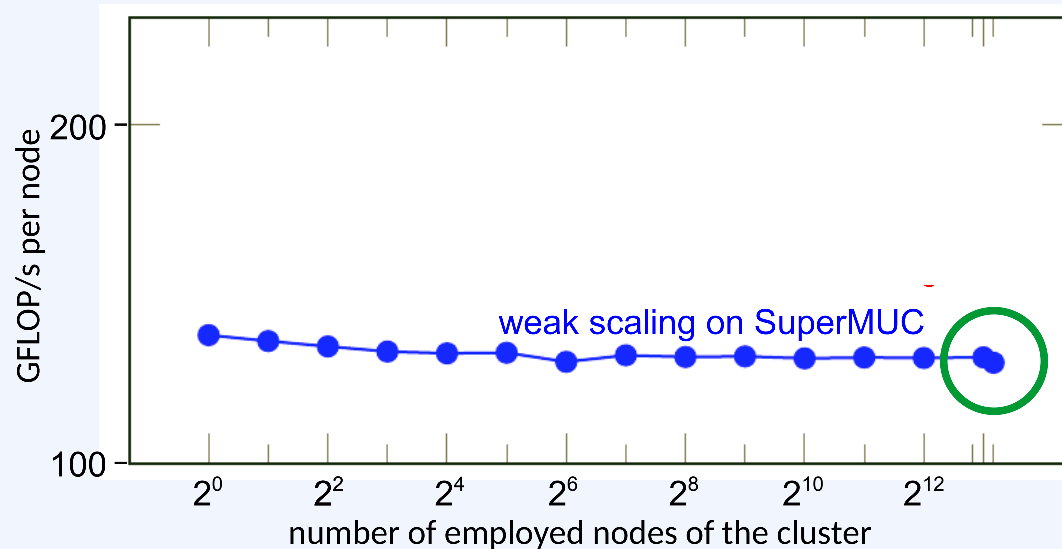
Linked cells + domain decomposition

Norwegian University of Life Sciences

SuperMUC (Garching):
SandyBridge architecture

http://www.ls1-mardyn.de/

(large systems 1: molecular dynamics)



TECHNISCHE UNIVERSITÄT MÜNCHEN

N = 4 125 000 000 000

2013 molecular dynamics world record[1]

SuperMUC weak scaling

Computational Molecular Engineering

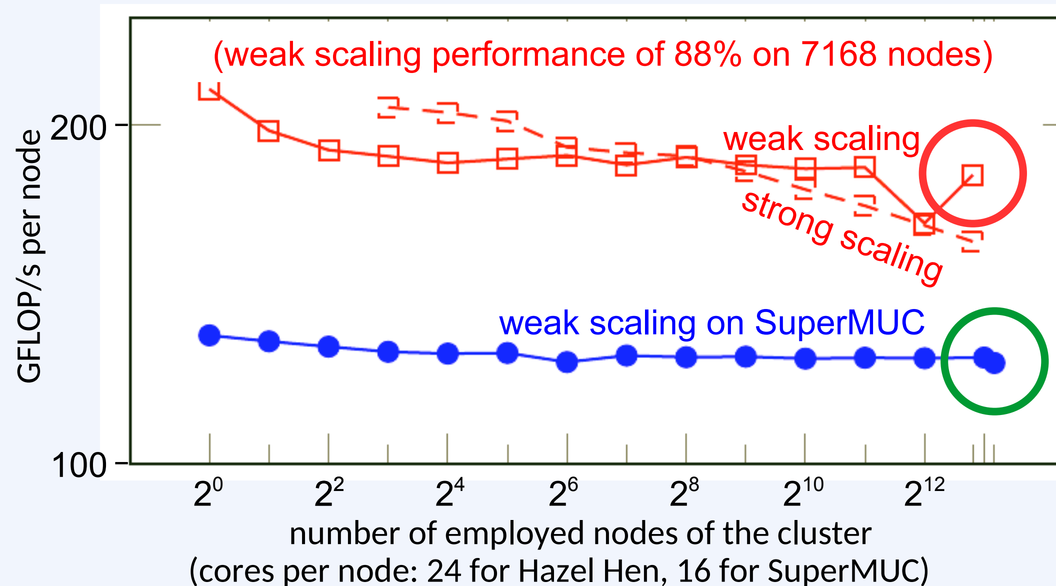
[1]W. Eckhardt, A. Heinecke, R. Bader, M. Brehm, N. Hammer, H. Huber, H.-G. Kleinhenz, J. Vrabec, H. Hasse, M. Horsch, M. Bernreuther, C. W. Glass, C. Niethammer, A. Bode & H.-J. Bungartz, *Proc. ISC 2013*, LNCS 7905, 1 – 12, **2013**.

Linked cells + domain decomposition

Norwegian University of Life Sciences

Hazel Hen (Stuttgart):
Haswell architecture

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



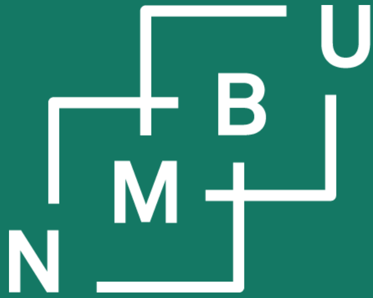
number of employed nodes of the cluster
(cores per node: 24 for Hazel Hen, 16 for SuperMUC)

N = 21 000 000 000 000
2019 molecular dynamics world record[1]

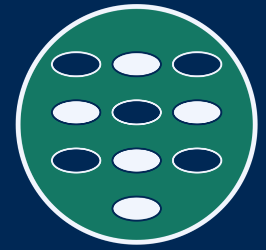
Hazel Hen
weak scaling

Computational
Molecular Engineering

[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**.



Noregs miljø- og biovitskaplege universitet



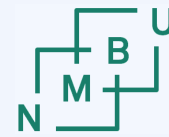
5 Parallel data

5.1 Decomposition schemes

5.2 Linked cells

5.3 Message-passing serialization

23rd November 2022



Norwegian University of Life Sciences

The need for serialization of data

To transfer data through a communication channel as a message, the data items and their parts need to be serialized (ordered) in a well-defined way that is understood both by the sender and the receiver.

- As a file, if file I/O is the mechanism by which data are exchanged.
  - We will have a look into parallel file I/O, using MPI-IO.
- As a contiguous chunk of memory, *if the exchange is memory-based*.
  - In case of message-passing parallelization, *this is a critical step*!

The challenge:

- Going from procedural programming to OOP, we regrouped elementary data items to become less well arranged for this purpose.
- Advanced dynamic data structures are not contiguous in memory.
- While an object technically has a fixed size in memory, the content of a dynamic container object has variable size.

Serialization: Unwrap the data structure

Week 38/39, we went from **low-level** to **object-oriented** data structures:

Low-level oriented arrangement of data: (**sphere-collisions-low-level.zip**)

```
int count_collisions(
   int N, float size[], float coordx[],
   float coordy[], float coordz[]
);

int main() {
   …
   cin >> N;
   float* size = new float[N]();
   float* coordx = new float[N]();
   float* coordy = new float[N]();
   float* coordz = new float[N]();
   …
   int result = count_collisions(
      N, size, coordx, coordy, coordz
   );
   …
}
```

Object-oriented arrangement of data: (**sphere-collisions-struct.zip**)

```
struct Sphere {
   float size = 0.0;
   float coords[3] = {0.0, 0.0, 0.0};
};

int count_collisions(int N, Sphere spheres[]);

int main() {
   …
   cin >> N;
   Sphere* spheres = new Sphere[N]();
   …
   int result = count_collisions(N, spheres);
   …
}
```

```
MPI_Send(
   coordx, N, MPI_FLOAT,
   target_rank, tag,
   MPI_COMM_WORLD
);
```

```
MPI_Recv(
   coordx, N, MPI_FLOAT,
   source_rank, tag,
   MPI_COMM_WORLD,
   MPI_STATUS_IGNORE
);
```

Just for inter-process communication, we can convert back to the **low level**.

Stream-based serialization

Norwegian University of Life Sciences

Observation:

- It is not straightforward to unwrap more complex data structures.
- We were already using streams for serialization, in particular file I/O.
- The same stream serialization can be used to transfer objects via MPI.

If a **stringstream** s is used to store the data, the method s.str().c_str() can be used for sending a char array, *e.g.*, with MPI_Send.

Size in characters: s.str().size()   // +1 for '\0' attention, pitfall!

overloaded operators in **graph-stream** (**graph.cpp**)

```
std::istream& operator>>(
  std::istream& is, Graph& g
) {
  g.in(&is);
  return is;
}
std::ostream& operator<<(
  std::ostream& os, const
Graph& g
) {
  g.out(&os);
  return os;
}
```

Prerequisite: The input and output methods (and operators) must be aligned.

Stream-based serialization

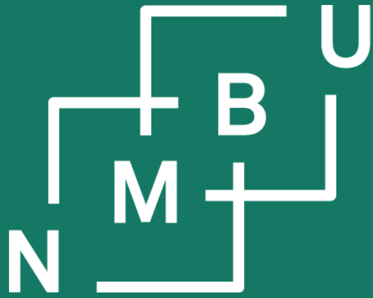
Example code **graph-stream**:

```
if(rank == 0) {
  // open in-filestream
  std::ifstream indata(argv[1]);

  // read graph object from file
  indata >> g;
  indata.close();

  // write into stringstream
  std::stringstream text << g;

  // inform recipient about content size
  message_size = text.str().size() + 1;
  MPI_Send(
    &message_size, 1, MPI_INT,
    1, 1, MPI_COMM_WORLD
  );

  // send content to recipient
  MPI_Send(
    text.str().c_str(), message_size,
    MPI_CHAR, 1, 2, MPI_COMM_WORLD
  );
}
```

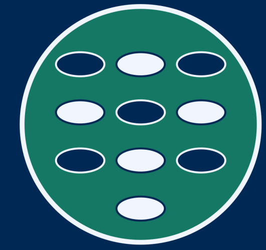
message tag 1

message tag 2

```
if(rank == 1) {
  // get information about the content size
  MPI_Recv(
    &message_size, 1, MPI_INT, 0, 1,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE
  );

  // allocate buffer and receive the content
  char* buffer = new char[message_size]();
  MPI_Recv(
    buffer, message_size, MPI_CHAR,
    0, 2, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE
  );

  // write into stringstream
  std::stringstream text << buffer;
  delete[] buffer;
  buffer = nullptr;

  // read graph object from stringstream
  text >> g;
}
```



Noregs miljø- og biovitskaplege universitet



5 Parallel data

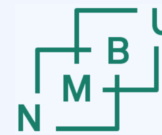
5.1 Decomposition schemes

5.2 Linked cells

5.3 Message-passing serialization

5.4 Parallel input/output

23rd November 2022



Parallel I/O: The challenge

File input requirement:

- Typically, at **startup**, plus possibly when a user **ingests data**.

File output requirement:

- Typically, upon **termination**, plus possibly when a user **extracts data**.
- But also for **checkpointing**:[1] The overall execution state must be saved.

How can this be done using (non-parallel) **sequential I/O** such as an **fstream**?

- One process (such as rank 0) scatters/gathers and reads/writes for all.
- Processes write into the same file, but one after another.
- Each process writes its own separate file; they are postprocessed later.

As the application scenario is **scaled up** (and computational resources also), I/O file sizes usually grow in proportion, and more processes are involved.

But these operations are **not concurrent** – they limit the scalability of the code.

[1]See *e.g.* fault-tolerance discussion by A. Skjellum, D. Schafer, arXiv:2112.10814 [cs.DC], **2021**.

Split-file output (with postprocessing)

Norwegian University of Life Sciences

Example **charmap-output**: All ranks write separate files.

mpirun -np 8 /generator-parallel … benchmark.pxl …

creates file benchmark.pxl.0 to file benchmark.pxl.7.

When needed, the files can be concatenated:

cat benchmark.pxl.* > benchmark.pxl

The concatenated file can be further exported to BMP.

```
./generator-sequential 16384 10 benchmark.pxl benchmark.vct
Image edge size:  16384 pixels (16384 x 16384)
No. random disks: 10
Pixel output to:  benchmark.pxl
Vector output to: benchmark.vct

===
Parallel environment setup:   0 s
Character map generation:     2.93694 s
Character map file output:    3.3157 s
Parallel environment cleanup: 0 s
===
Total program execution time: 6.25265 s
```

```
mpirun -np 8 ./generator-parallel 16384 10 benchmark.pxl …
Image edge size:  16384 pixels (16384 x 16384)
No. random disks: 10
Pixel output to:  benchmark.pxl.0
Vector output to: benchmark.vct

===
Parallel environment setup:   0.24087 s
Character map generation:     1.20272 s
Character map file output:    1.57878 s
Parallel environment cleanup: 0.141087 s
===
Total program execution time: 3.16346 s
```

Split-file output: Charmap example

Only one "long" data item needs to be exchanged, **synchronizing the random number generators**. In this case it is an advantage, not a disadvantage, to use a **deterministic pseudo-random number generator**.

Attention: Most file formats have a sort of header, and they can have a coda. This needs to be managed correctly when using split-file output.

Parallel I/O using MPI-IO

Norwegian University of Life Sciences

int MPI_File_open(
  MPI_Comm comm, const char* fname, int mode, MPI_Info info, MPI_File* fh
)

MPI_INFO_NULL

int MPI_File_set_view(
  MPI_File fh, MPI_Offset displacement,
  MPI_Datatype etype, MPI_Datatype filetype,
  const char *datarep, MPI_Info info
)

"native"   MPI_INFO_NULL

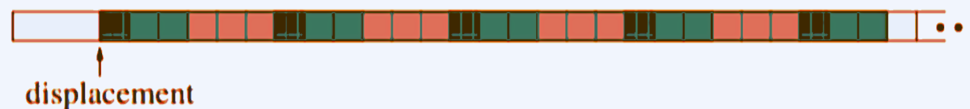


Figure 14.2: Partitioning a file among parallel processes

MPI 4.0 standard, Chapter 14

Access modes:

MPI_MODE_RDONLY (read-only)

MPI_MODE_WRONLY (write-only)

MPI_MODE_RDWR (read-write)

MPI_MODE_APPEND (start from the end of file)

...

(Example: File view as a series of blocks.)

Parallel I/O using MPI-IO: Example

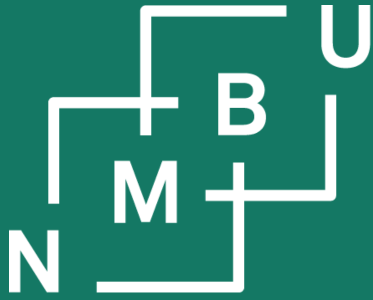
Example **mpi-io-demo.cpp**:

```
#ifdef USE_MPI
  size_t elements = cm.get_sizex() * cm.get_sizey();
  unsigned char* content = cm.access_data();

  // displacement, etype, and file type information
  MPI_Offset displacement = header_size
                            + yoffset*a*sizeof(unsigned char);
  MPI_Datatype etype = MPI_UNSIGNED_CHAR;
  MPI_Datatype etype_array;
  MPI_Type_contiguous(elements, etype, &etype_array);
  MPI_Type_commit(&etype_array);

  // open the file
  MPI_File fh;
  MPI_File_open(
    MPI_COMM_WORLD, pixout_fname.c_str(), MPI_MODE_WRONLY, MPI_INFO_NULL, &fh
  );

  // now create a "view" consisting of the displacement, the etype, and the file type
  MPI_File_set_view(fh, displacement, etype, etype_array, "native", MPI_INFO_NULL);
  MPI_File_write(fh, content, elements, etype, MPI_STATUS_IGNORE);
  MPI_File_close(&fh);
#endif
```

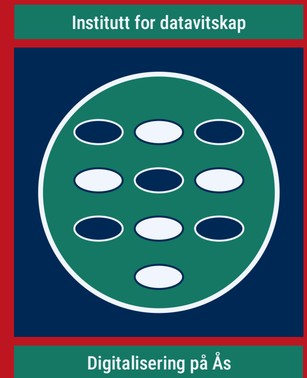
Attention: Here, we need to know exactly at what position in the file every rank writes.

We need to know how long the **header part of the file** and the **output from all the lower ranks** is going to be.

Discussion: Will this run faster or slower than with split-file output?



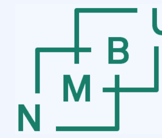
Noregs miljø- og biovitskaplege universitet



Conclusion

23rd November 2022

Quantifying the individual contribution



Norwegian University of Life Sciences

For **week 47 group report**, submit a statement on how responsibilities are distributed in your groups, as percentages/fractions of the respective effort:

- "Data structures" aspect
  - You self-assign responsibility in line with your division of work.
  - Member no. 1 had C_{11}% contribution, no. 2 had C_{12}%, no. 3 had C_{13}%.
  - Contributions to aspect must add up to unity, $C_{11} + C_{12} + C_{13} = 100\%$.
  - Can be any distribution from "it's one person's work" to "all did 1/3."
- "Algorithm and performance" aspect
  - Same logic as above, $C_{21} + C_{22} + C_{23} = 100\%$.
- "Concurrency" aspect
  - Same logic as above, $C_{31} + C_{32} + C_{33} = 100\%$.

This information is used to calculate the **individualized part** (10%) of the grade.

Individualized part of the grade

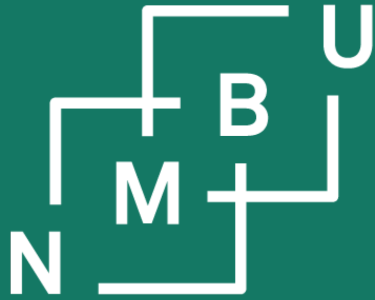
How does it work?

Assume that group member j has taken responsibility for x_{1j} fraction of the aspect “data structures,” x_{2j} for the aspect “algorithm and performance,” and x_{3j} for the aspect “concurrency.” The group grades for these are g_1, g_2, and g_3.

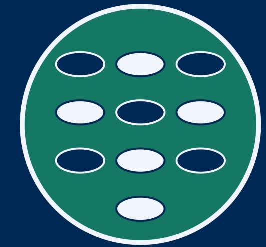
- $C_{1j} + C_{2j} + C_{3j}$ for group member j does not need to add up to 100%.
- The sum of all $c_j = (C_{1j} + C_{2j} + C_{3j})/3$ over all group members j is 100%.
- The individualized grade for j is then simply the weighted average $(g_1C_{1j} + g_2C_{2j} + g_3C_{3j}) / 3c_j$. It is scaled to contribute 10% to the total.

Two exceptions:

- If $10\% < c_j < 25\%$, the **individualized grade** for j becomes **zero**.
- If $c_j \leq 10\%$, the **overall grade** for j becomes **zero**, or F as a letter grade.



Norges miljø- og biovitenskapelige universitet



INF205

Resource-efficient programming

5 Parallel data

5.1 Domain decomposition

5.2 Linked cells

5.3 Message passing serialization

5.4 Parallel input/output

23rd November 2022