Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205
Resource-efficient programming

3 Concurrency

3.1 Parallel programming
3.2 Message passing interface (MPI)

Concurrency: Why does it matter?

Assume a scenario where we can split a code into a fraction f that can be parellelized and the remainder $1 - f$ that is always sequential, never parallel.

Adding two vectors c[i] = a[i] + b[i], for i from 0 to 99 999, can be parallelized. Waiting for new instructions from the user cannot be parallelized.

Speedup is the *factor by which runtime decreases*; here, due to parallelization.

Amdahl's law:

- Runtime with a single process is given by some $t_1 = (1 - f)t_1 + ft_1$.
- Now assume that we are parallelizing the code as perfectly as possible:
  - With n parallel processes, the runtime becomes $t_n = (1 - f)t_1 + ft_1/n$.
- Now assume that we have infinite computing resources at our hands:
  - With infinite parallel processes, the runtime becomes $t_\infty = (1 - f)\,t_1$.
- The **maximum possible speedup** for our code is $S_\infty = t_\infty/t_1 = 1 / (1 - f)$.

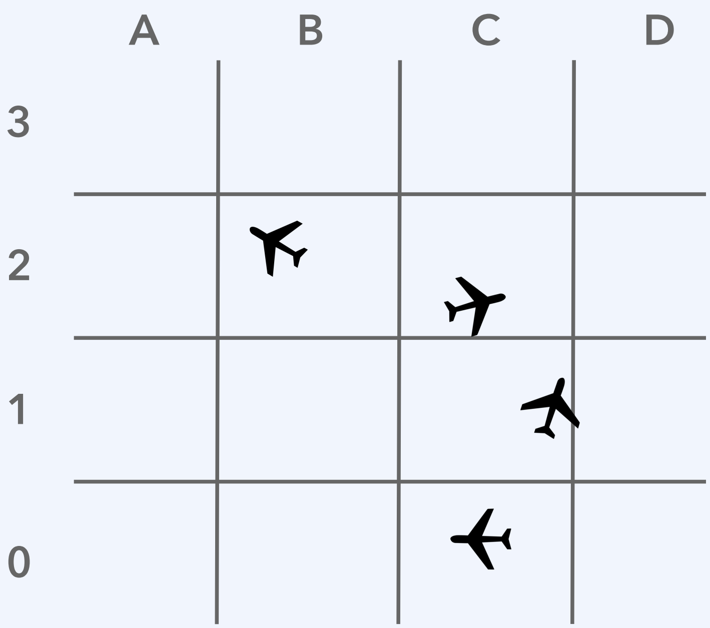
If f = 99% can be parallelized, speedup can never be greater than $S_\infty = 100$.

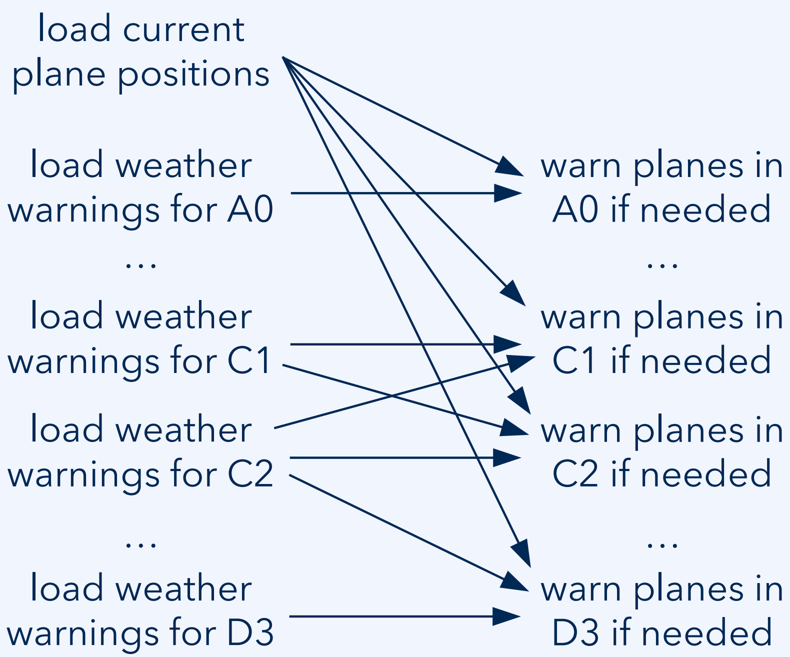
Norwegian University of Life Sciences

Concurrency: Why does it matter?

Parallel programming is about efficiently exploiting a parallel architecture.

Concurrency as a topic is about what can be parallelized and what cannot.





Domain decomposition is one of the techniques for this kind of concurrency.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

3 Concurrency

3.1 Parallel programming

INF205

19th October 2022

Norwegian University of Life Sciences

Message passing

Message passing is the most general paradigm of parallel programming.

It can be carried out *irrespective whether* or not the *processes* (can also be called **ranks** in MPI) are executed on the same computing node and *have shared memory access*. It only assumes that they can exchange messages.

Challenges of message passing based parallelization:

- Synchronization (waiting) while processes need to talk to each other.
- What if there are very many processes, do they all message each other?

- What if the recipient would already have had access to the data?
- Processes need to figure out what information they must give to others.

In high performance computing, message-passing based parallelization is usually done using **MPI**, the message passing interface.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

3 Concurrency

3.1 Parallel programming

<u>3.2 Message passing interface</u>

19th October 2022

MPI: Getting started

The target systems of MPI programs are often *clusters with thousands of cores*.

However, the code is not usually developed on these systems, but on the programmers' usual working environment. Even on a laptop/workstation, MPI makes you realize a *speedup*, since today these are all *multicore systems*.

To get started install an MPI environment, *e.g.*, **Open MPI** (package **openmpi**).

The **compiler command** becomes "**mpiCC ...**" or similar (instead of "g++ ...").
The *binary executable* produced by the compiler *will not run on its own*!

Instead: **mpirun -np** <number of processes> <executable>

This creates a number of parallel processes with ranks starting from 0.
Often the *process with rank 0* takes the role of the "master" or "scheduler".

See also the Open MPI documentation: https://www.open-mpi.org/doc/v4.1/

MPI: Getting started

An MPI program needs to *initialize* and *finalize* the MPI environment.
Every process needs to *know its rank* (and, usually, the *number of processes*).

```
#include <mpi.h>

int main(int argc, char** argv)
{
  MPI_Init(&argc, &argv);

  int rank = 0;  // what is the rank of this process?
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  int size = 0;  // how many processes are there?
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  …  // here comes the actual program

  MPI_Finalize();
}
```

(See the **mpi-primes** example code.)

Often the rank no. of a process, together with the number of processes, is already enough input to implement a basic parallelization scheme.

This is also the case for our prime-number test example:

5	7	11	13	17	19	23 …
0	0	1	1	2	2	3 …

MPI send and receive

The most basic communication step is send/receive from one rank to another.

```
int MPI_Send(
  void* content, int count, MPI_Datatype type,
  int destination_rank, int tag, MPI_Comm handle
);
```

```
int MPI_Recv(
  void* buffer, int count, MPI_Datatype type,
  int source_rank, int tag, MPI_Comm handle,
  MPI_Status* status );
```

content is the address from which the source data are read; it is often an array, but can also be a pointer to a single data item

buffer is an address to which the received data can be written; the programmer needs to take care of memory allocation, *etc.*

count is the number of data items

type is their type as an MPI environment expression (*e.g.*, MPI_SHORT_INT, MPI_INT64_T, MPI_FLOAT, …)

tag is an identifier; send and receive must have the same tag

destination_rank is the rank of the process with the matching MPI_Recv(…) operation

source_rank is the rank of the process with the matching MPI_Send(…) operation

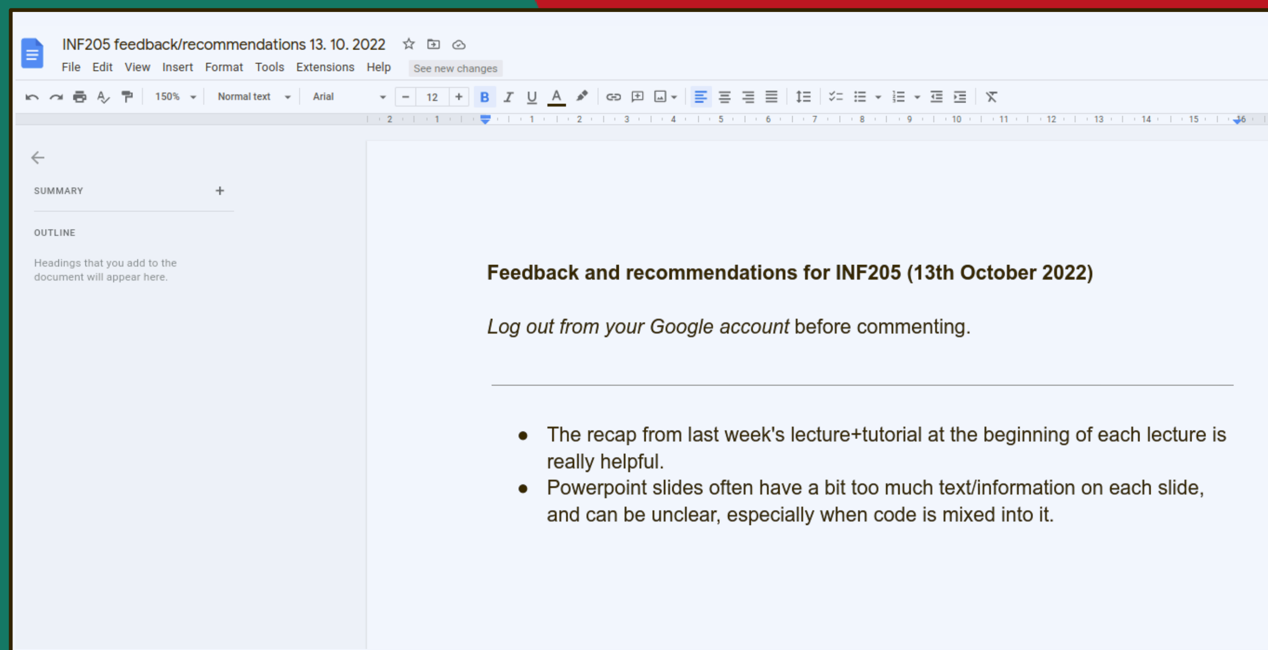
(Standard values from handle and status are MPI_COMM_WORLD and MPI_STATUS_IGNORE.)

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

Conclusion


INF205 feedback/recommendations 13. 10. 2022

Feedback and recommendations for INF205 (13th October 2022)

Log out from your Google account before commenting.

- The recap from last week's lecture+tutorial at the beginning of each lecture is really helpful.
- Powerpoint slides often have a bit too much text/information on each slide, and can be unclear, especially when code is mixed into it.

Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205
Resource-efficient programming

3 Concurrency

3.1 Parallel programming

3.2 Message passing interface (MPI)

19th October 2022