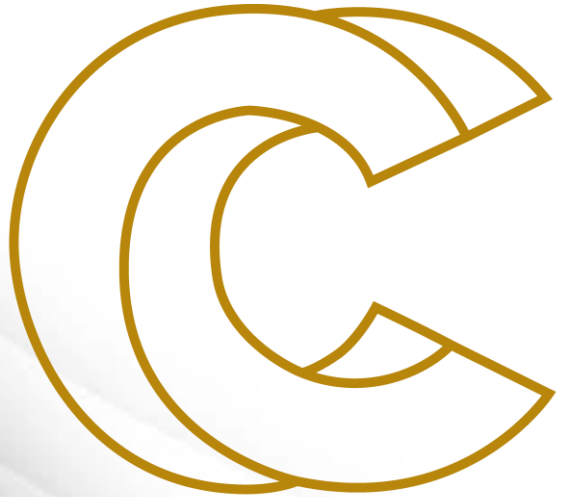


EURO^2

Introduction to HPC

By EuroCC Belgium

Outline of this presentation

EURO²

PART 1

- Introduction
  - Example of uses
  - The EuroHPC joint undertaking the EuroCC project
- Current status of the supercomputing infrastructures
  - Performance and the TOP500 list
  - Supercomputers in Europe and in Belgium

PART 2

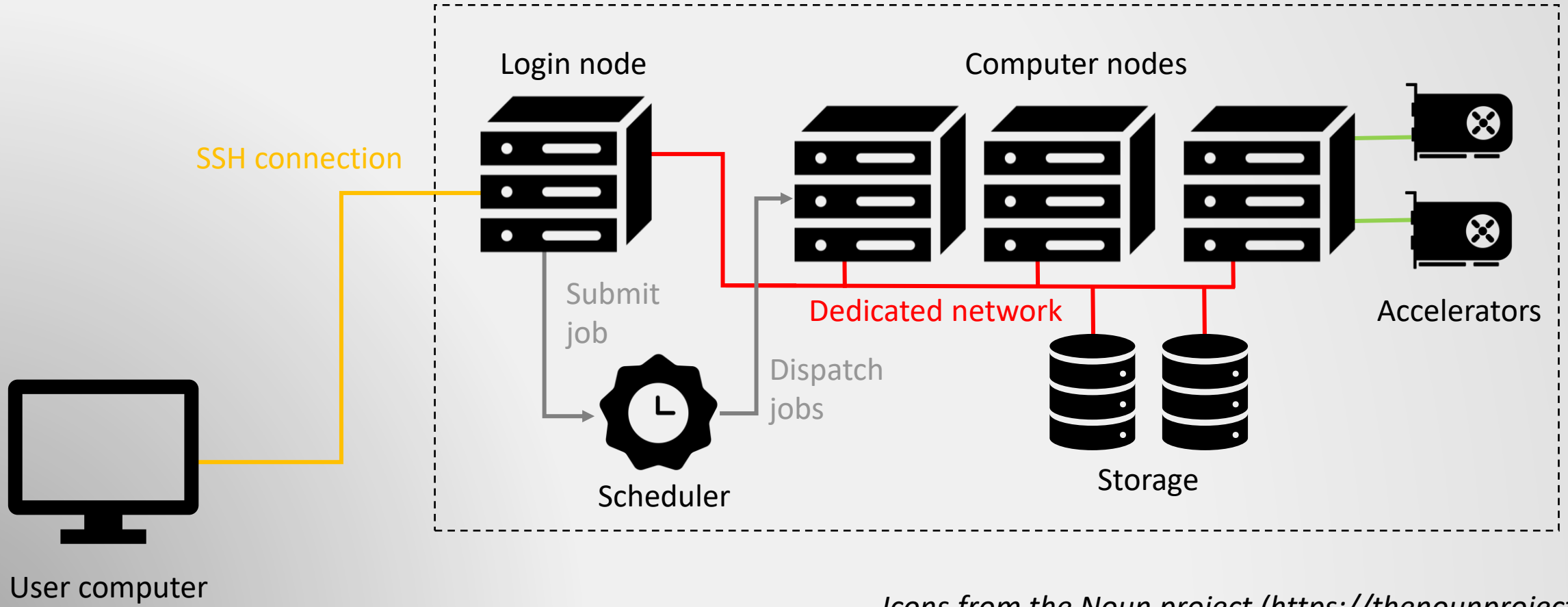
- Understand how a supercomputer works:
  - Architecture & Components
  - Interacting with supercomputers
- Understand how program can use such large resources, and what are the issues that needs to be overcome:
  - Parallelism
  - Parallelization issues

PART 2

What are supercomputers?

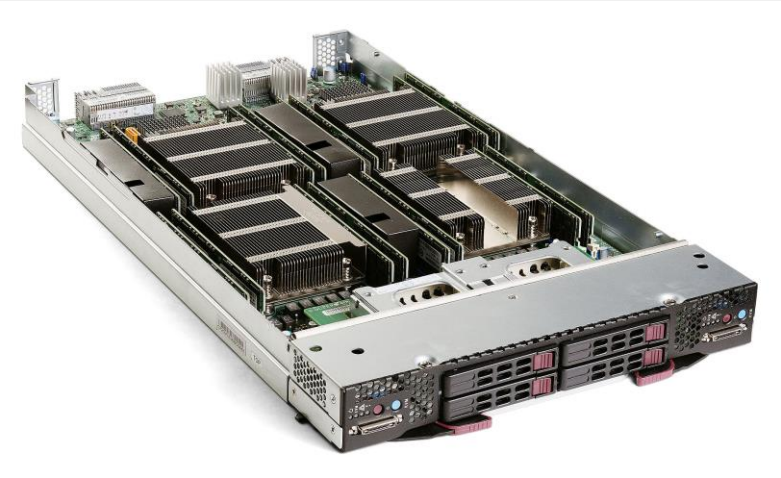
How they work

Anatomy of a cluster



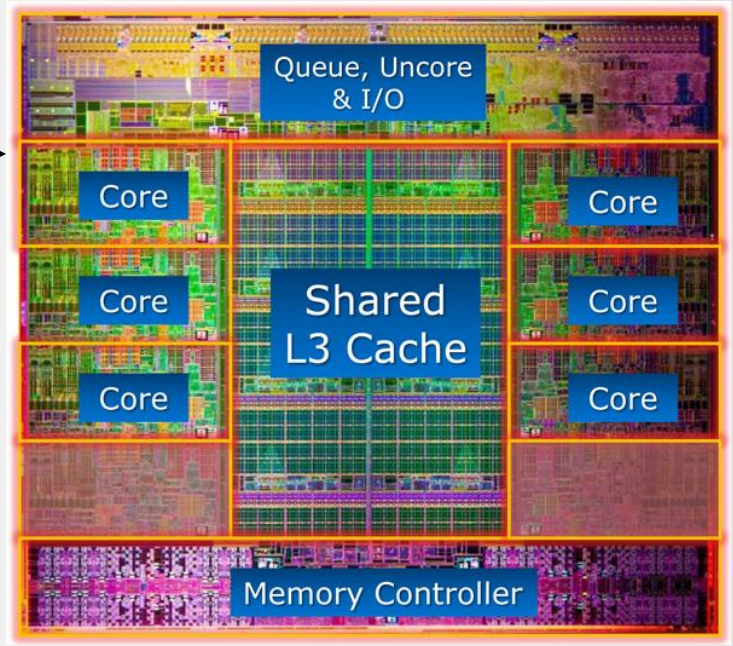
Icons from the Noun project (https://thenounproject.com/)

Components



Computer node (Wikipedia)
Acts basically like a computer

- Socket (CPU)
- RAM
- Networking
- Cooling
- Local storage
- …

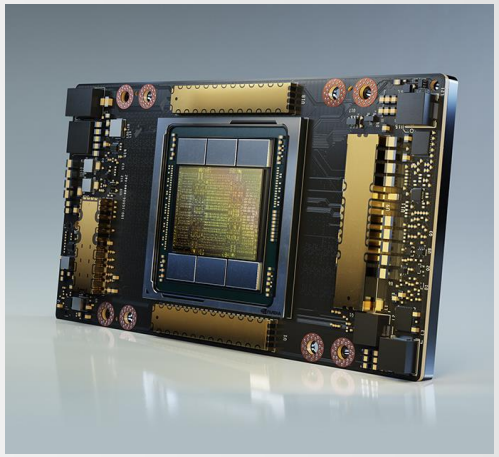


Intel sandybridge (die)
(https://www.anandtech.com/)

Components: GPU / Accelerators

- GPGPU: general-purpose computing on GPU
- Instead of a few powerful cores (CPUs), many less powerful cores
- Consumer grade GPU: provide good FLOPS for single precision operation, not for double precision
- Dedicated class of cluster grade GPU (e.g., NVIDIA Ampere or AMD Instinct)
- Future of HPC?

AMD Instinct (amd.com)

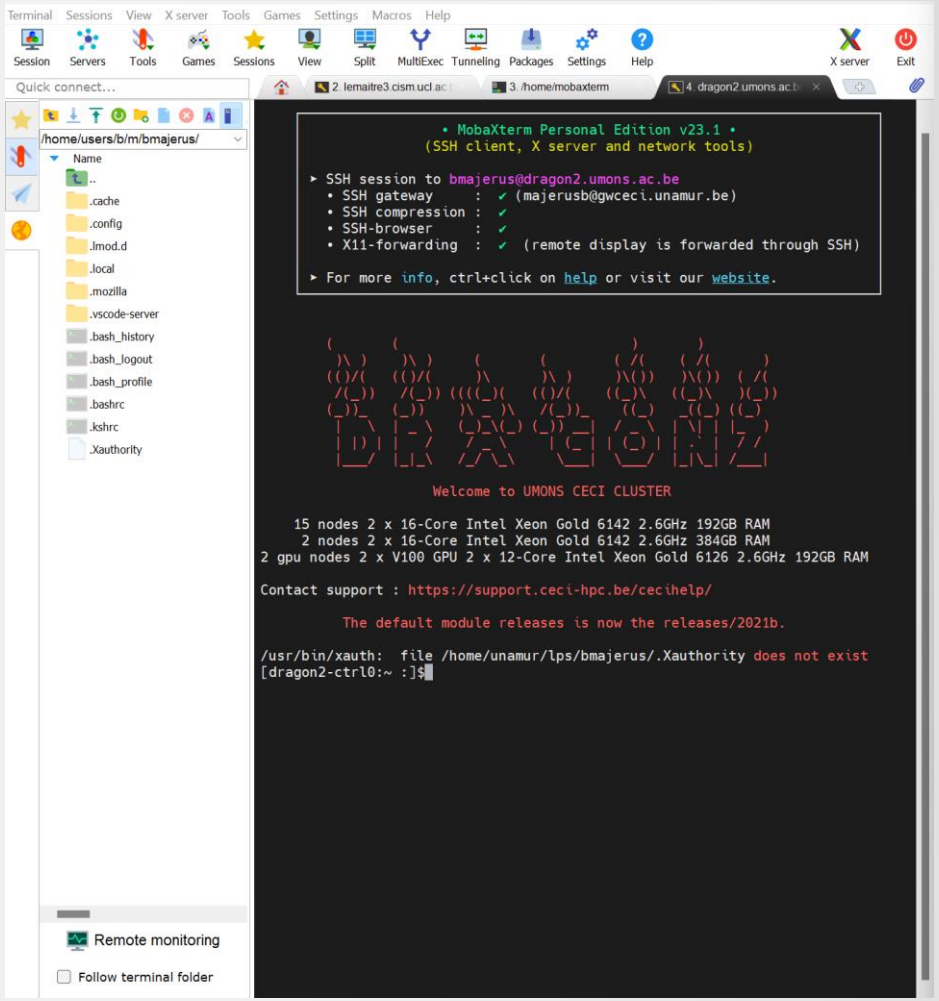


Ampere (nvidia.com)

Interaction with the supercomputer

You interact through the command line (Linux, Mac) or specific softwares (Windows, Mac).

- SSH client for connection
- Terminal for writing and receiving text
- Graphical user interface is possible (but not yet available everywhere)



The scheduler

- For each "job", the scheduler (e.g., SLURM) requires to know time/memory/number of processor and node and tries to fit the job when a slot is available.
- Different clusters have different purposes, e.g.,
  - High-memory application,
  - Small and fast jobs,
  - Nodes with accelerators, ...

```
#!/bin/bash
#submission script for Lemaitre3
#SBATCH --job-name=CNT
#SBATCH --time=24:00:00 # hh:mm:ss
#SBATCH --ntasks=128
#SBATCH --mem-per-cpu=2000 # megabytes
#SBATCH --partition=batch


module load Python/3.6.3-foss-2017b
module load libxc

srun gpaw-python cnt_ph.py >log_ph
~
~
~
~
~
~
~
~
~
```

How to use such large resources efficiently?

Solutions... and challenges

Particularities of supercomputing

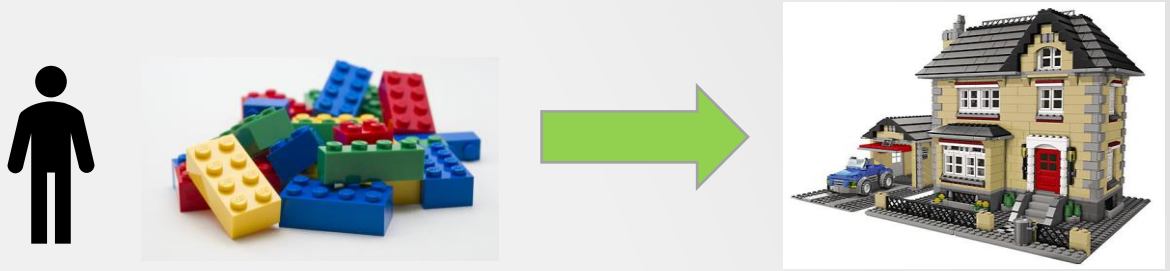
EURO²

- Advantage of supercomputing
  - Larger size: some problem requires large amount of memory
  - More speed: some problem requires long time to be solved
- Solution … And issues:
  - More memory (but storage hierarchy)
  - Parallelism (… has inherent difficulties)

Tackle speed: parallel computing

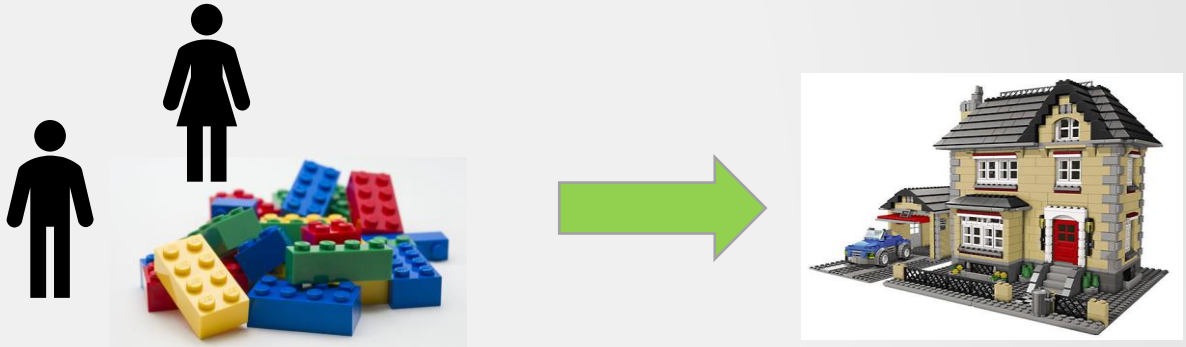
- To achieve such performances, the main idea is to rely on **parallel computing :** executing many operations in a single instance of time.
- But the program needs to be adapted for such purpose!

Serial world: 1 worker (person) to build



Images from LEGO (lego.com)

Parallel world: 2 workers to build → about 2 times faster



Speedup is never what you expect…

In most cases, a problem is never fully parallelizable (i.e., **embarrassingly** parallel)

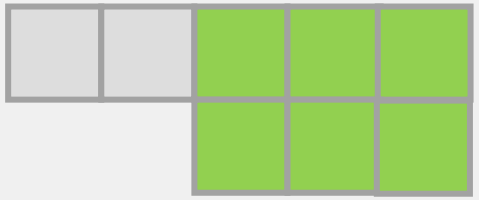
Non parallelizable (25%)

1 worker:

Parallelizable (75%)

$T_{tot,1} = 8$

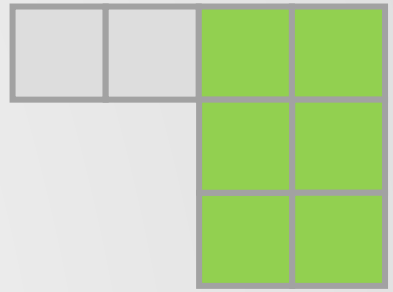
2 workers:



$T_{tot,2} = 5$

Speedup = $T_{tot,1} / T_{tot,2} = 1.6$

3 workers:



$T_{tot,3} = 4$

Speedup = $T_{tot,1} / T_{tot,3} = 2$

→ No matter how fast the parallel portion, we will always be limited by the **serial** part.

Speedup is never what you expect

Amdahl's Law:

$$S = \frac{1}{s + \frac{P}{N}}$$

S: actual speedup

s: serial portion of the code (in %)

P: parallel portion (in %)

N: number of processors

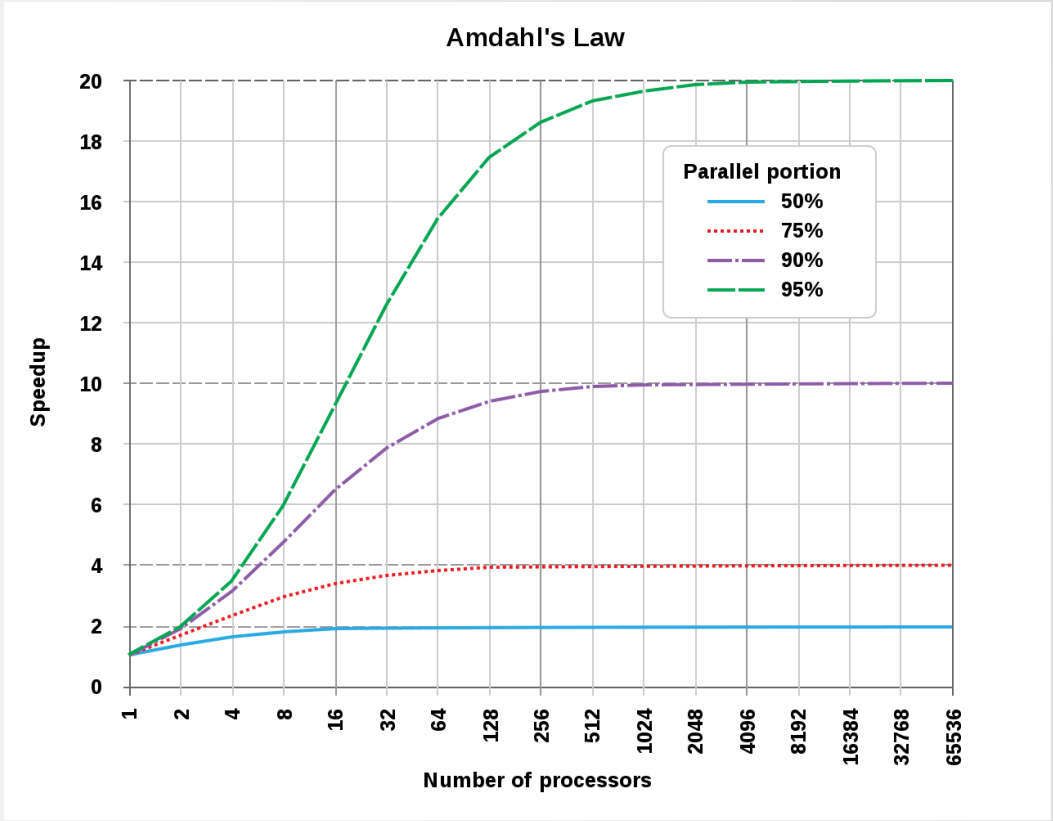
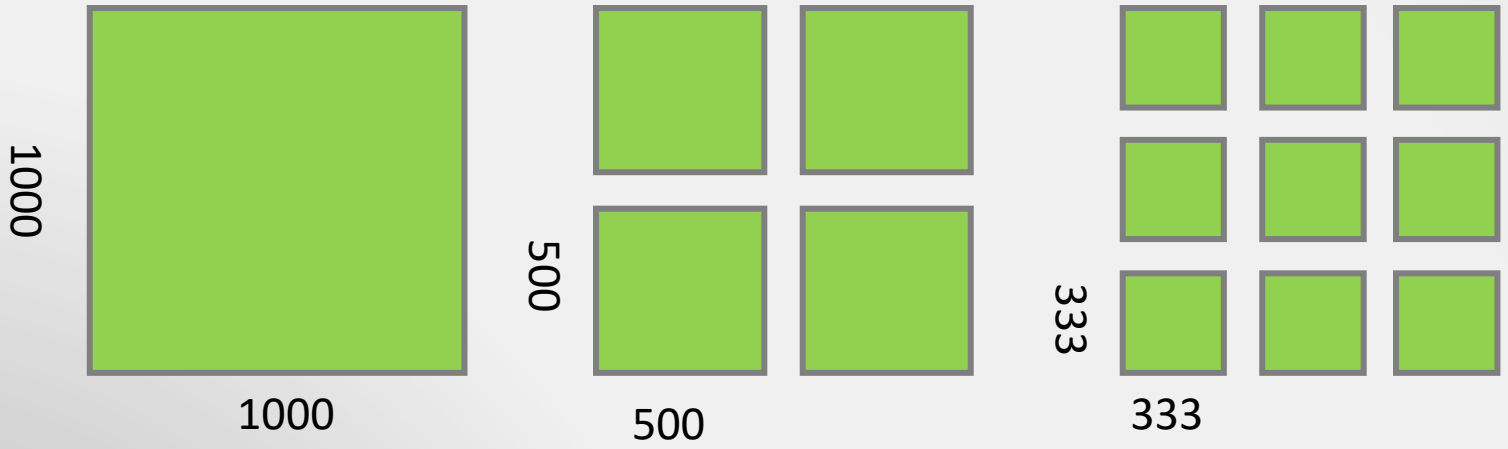


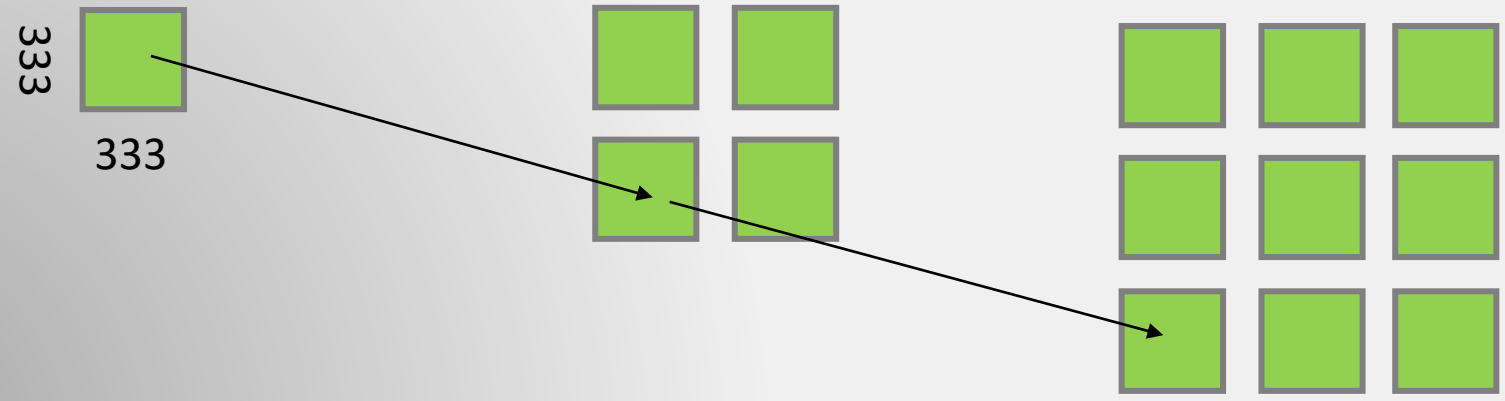
Image from Wikipedia (wikipedia.org)

This address the question "how much processors can I use for a given problem?"

Strong vs weak scaling



Strong scaling: size of the problem is constant and split across additional processors



Weak scaling: the size of the problem is the same for each processor.

Weak scaling is great

If we increase the size of the problem when more processors are added,

$$S = N - s\,(N - 1)$$

S: actual speedup

N: number of processors

S: serial fraction (in %)

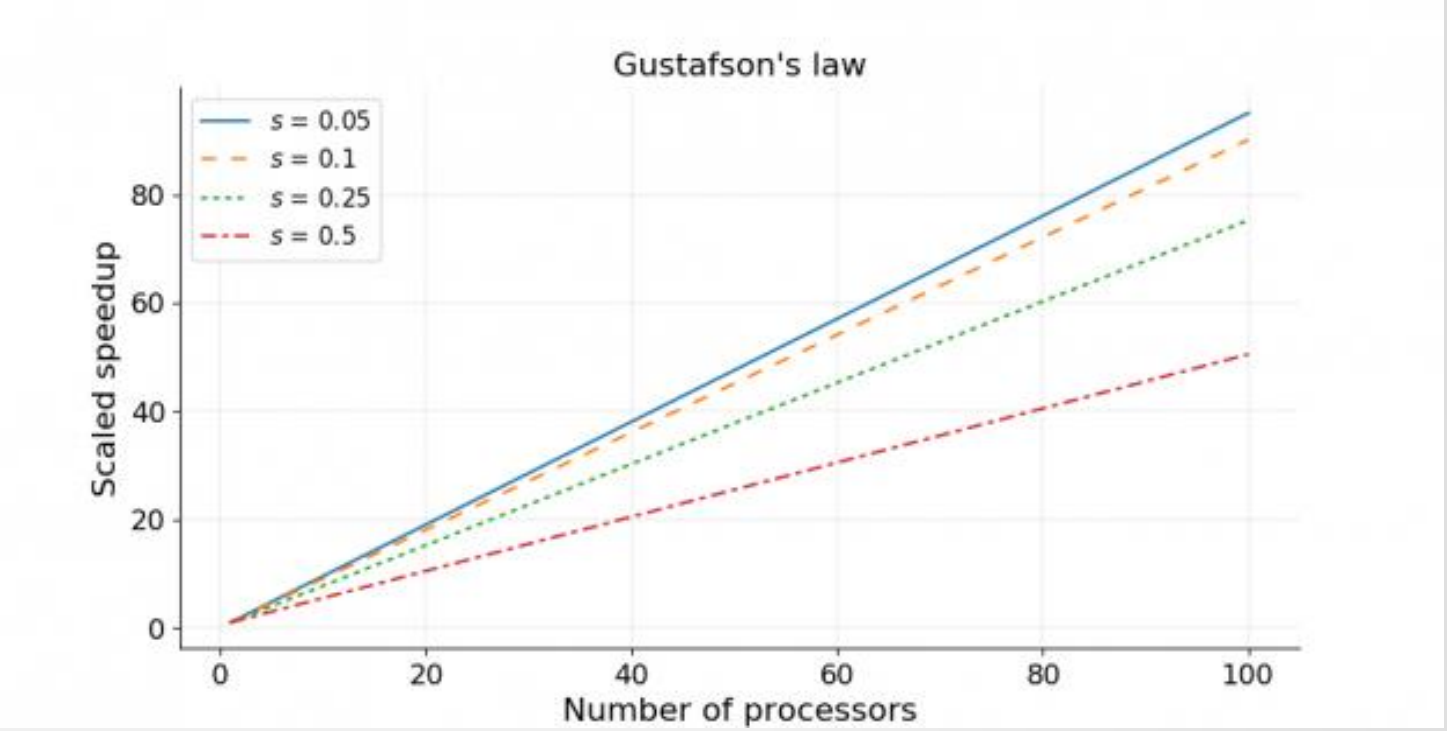
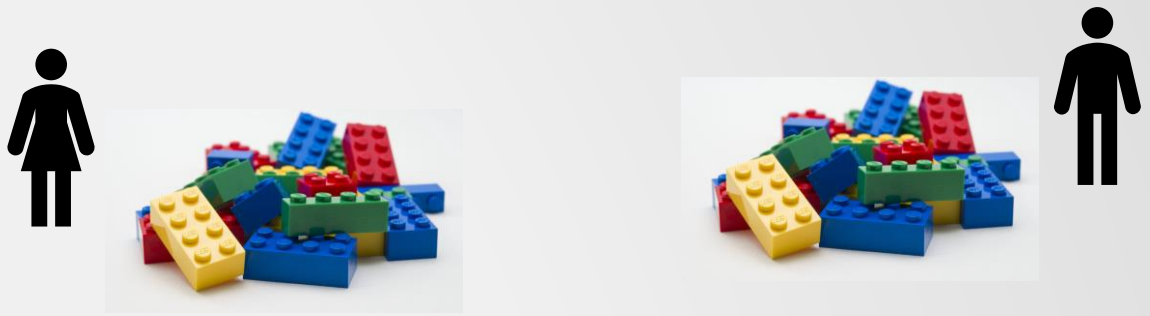


Image from Wikipedia (wikipedia.org)

This address the question "how much can I increase the size of my problem such that the execution time is the same as if I ran the problem with only one process?"

But there are always overheads

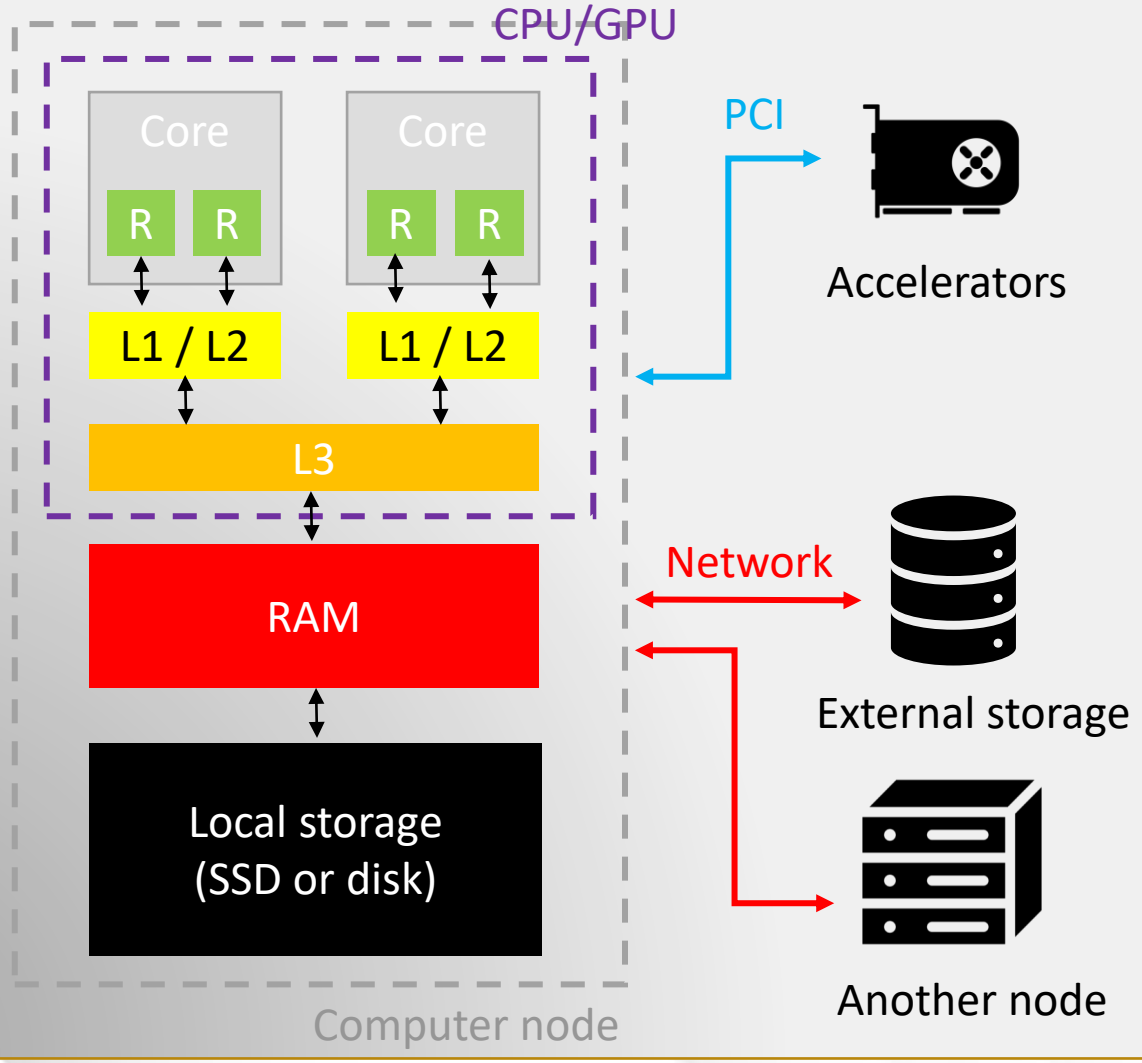
Shared memory model: all workers (or, here, person) are working on the same pool of data (or, here, lego pieces). Small overhead due to **synchronisation** (all person cannot work on the same part of the model at the same time).



Distributed memory model: each worker (or, here, person) is working on its own set of data (or, here, lego pieces). Generally, more efficient (no collaboration during work), but overhead due to **communication** (or, here, distribute pieces in the beginning and assembling the result at the end).

Generally, the two are mixed. It also requires a good **load balancing** (i.e., every person has the same amount of work to do). It is not that easy to achieve.

Also important: storage hierarchy



- The further away from the core, the slowest (but, generally, the more capacity)
- I/O may not be parallel
- Communication is a bottleneck when using multiple nodes
- Efficient movement of data to and from an accelerators

Icons from the Noun project (https://thenounproject.com/)

Tools of the trade

EURO²

Parallel programming:

- Vectorization (core level)
- Threading / OpenMP (node level)
- CUDA / HIP / OpenCL / OpenACC / OpenMP (accelerator level)
- Socket / MPI / PGAS (cluster level)

Optimized libraries:

- BLAS / LAPACK / MKL (linear algebra)
- FFTW (fast-Fourier transform)
- HDF5 / netCDF (parallel I/O)

Good news for computer rookies: Lots of scientific programs already built with parallel version and available on the supercomputers → no programming skills needed!

Conclusions

Conclusions

- Parallelization is at the core of the efficiency of supercomputers
- There are several parallelization paradigms with their own advantages and drawbacks
- You don’t need high programming skills to do supercomputing

→ More information on

- https://www.enccb.be/
- https://www.ceci-hpc.be/
- https://www.vscentrum.be/

Thanks!

Funded by
the European Union

EuroHPC
Joint Undertaking

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.