Using cholesky easily

Equipe IDCS

February 4 and 5, 2025

Institut Polytechnique de Paris

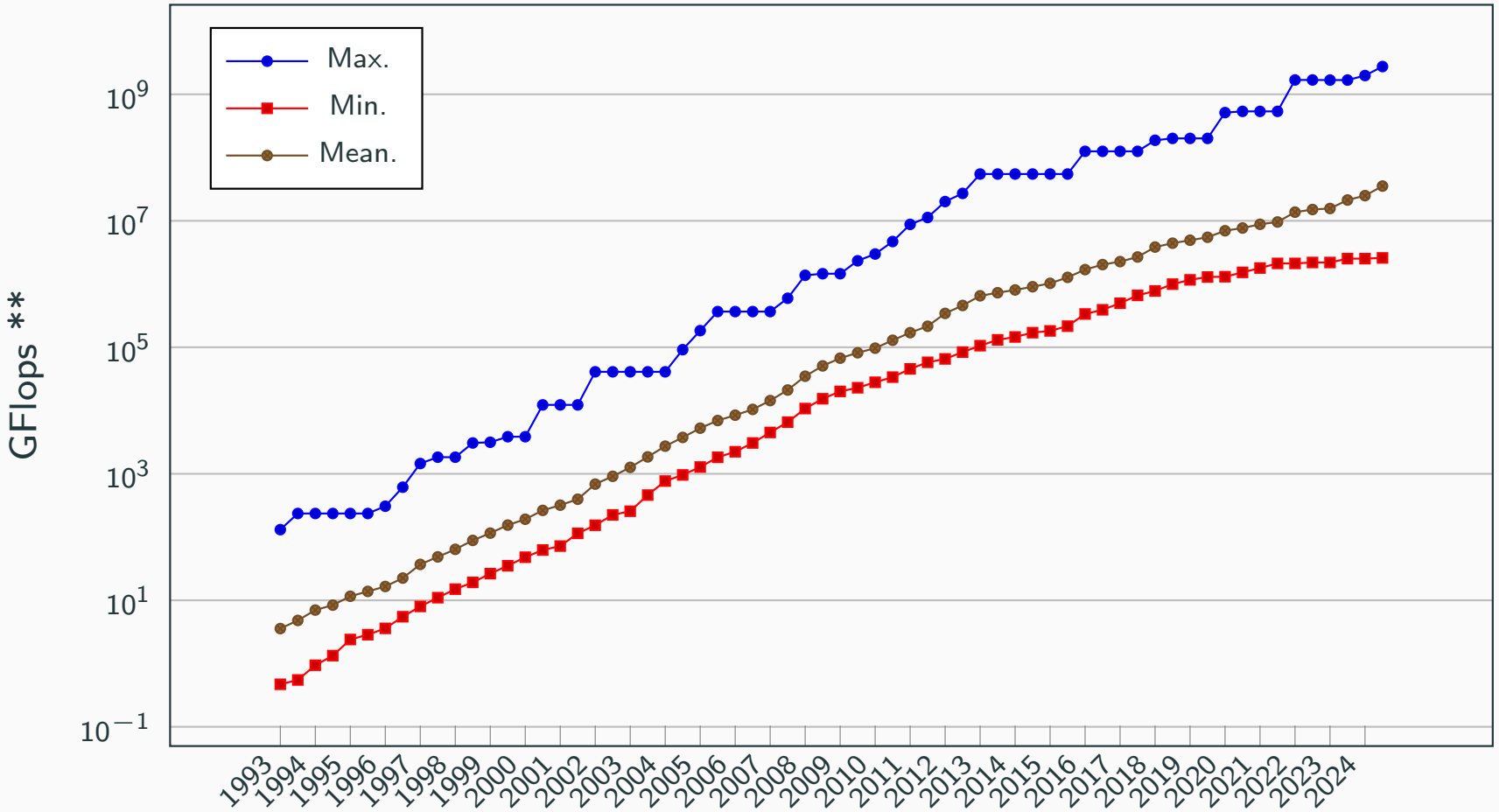
Table of contents

Introduction to parallel architecture

Table of contents

Context

Peak performance of the top 500 supercomputers of the world over the years *

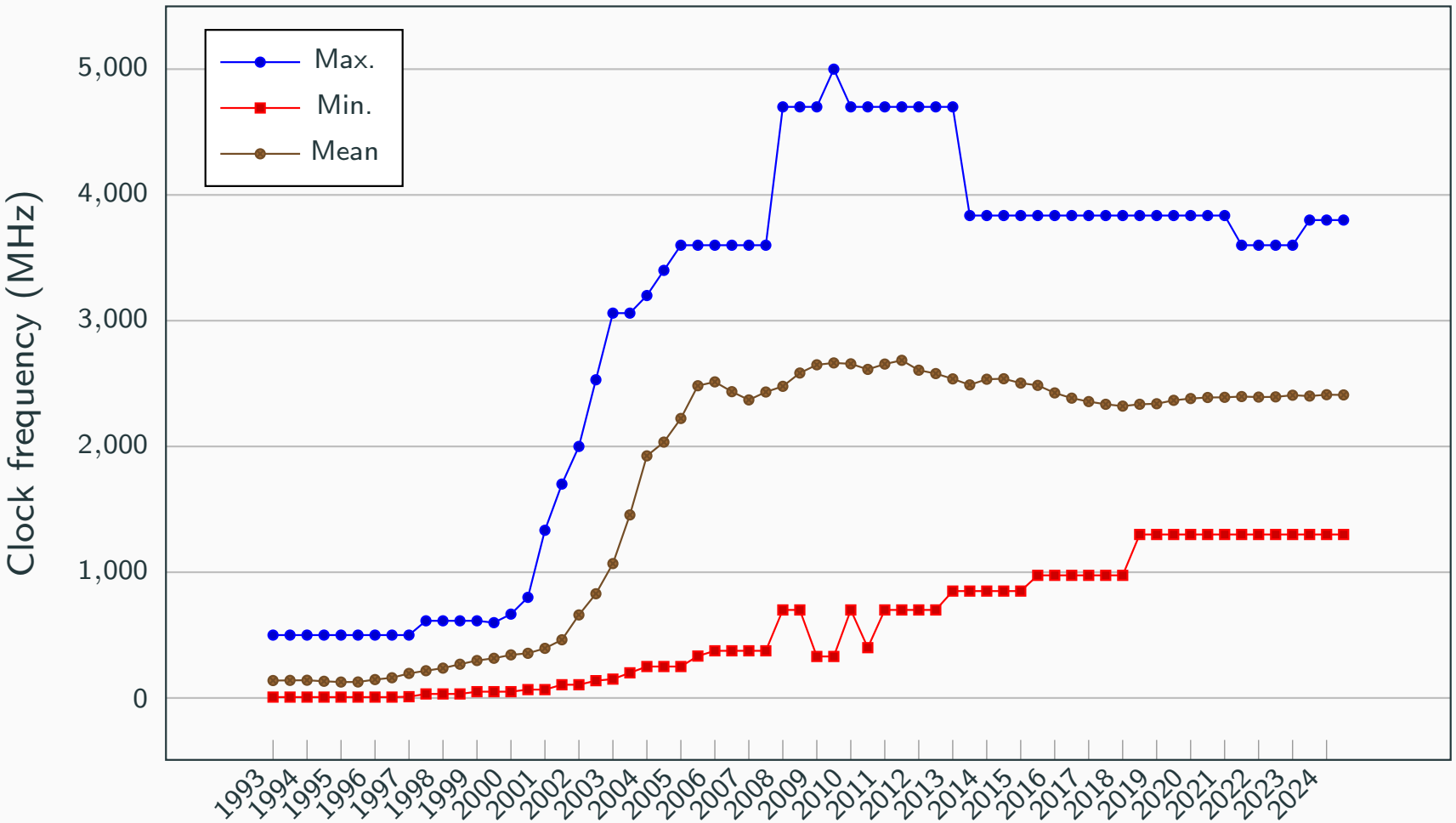


(*) Source : The TOP500 list (`https://www.top500.org`)

(**) Flops = Floating-point operations per second

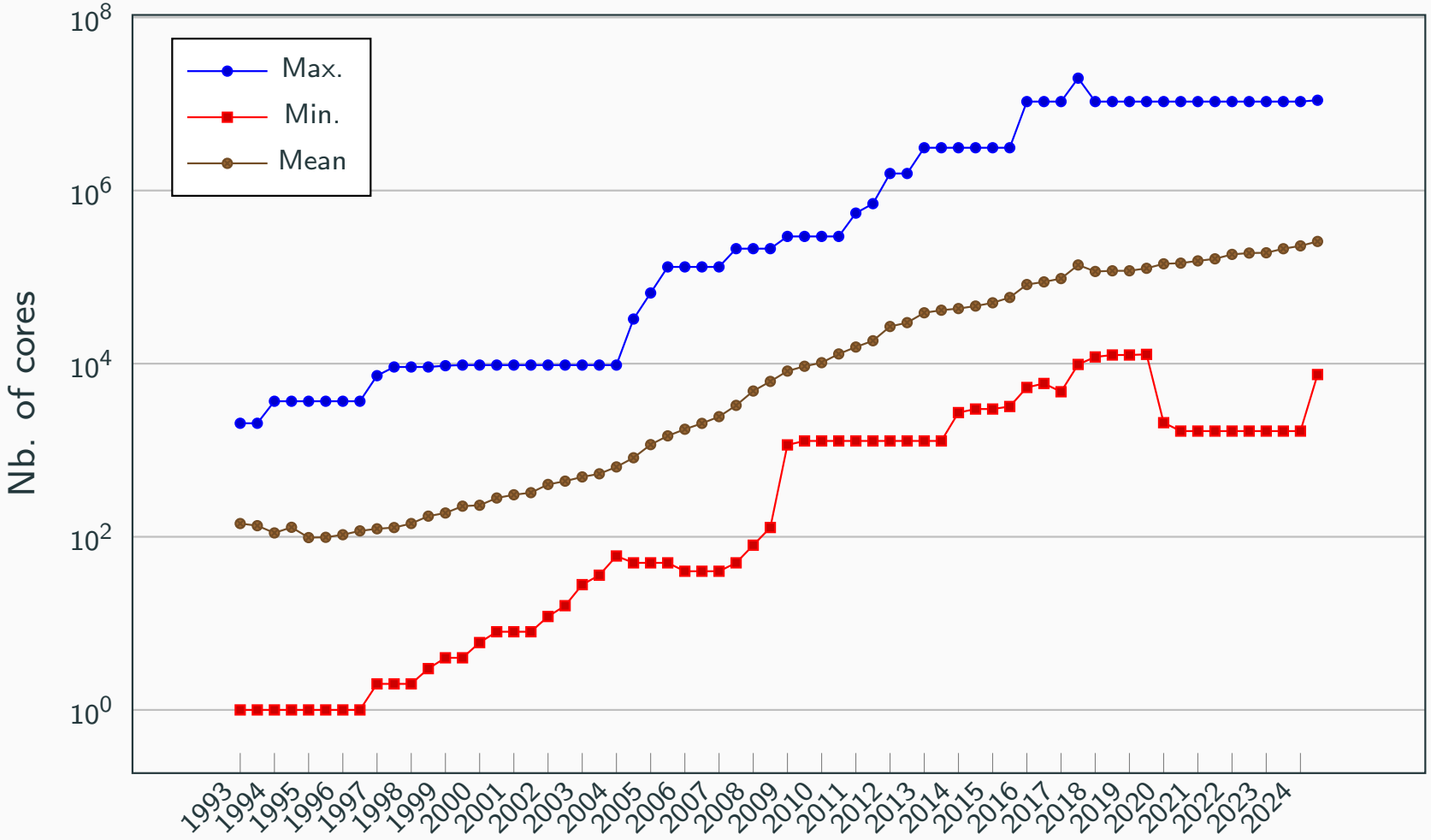
Context

Clock frequency of the top 500 supercomputers of the world over the years



Context

Number of cores of the top 500 supercomputers of the world over the years



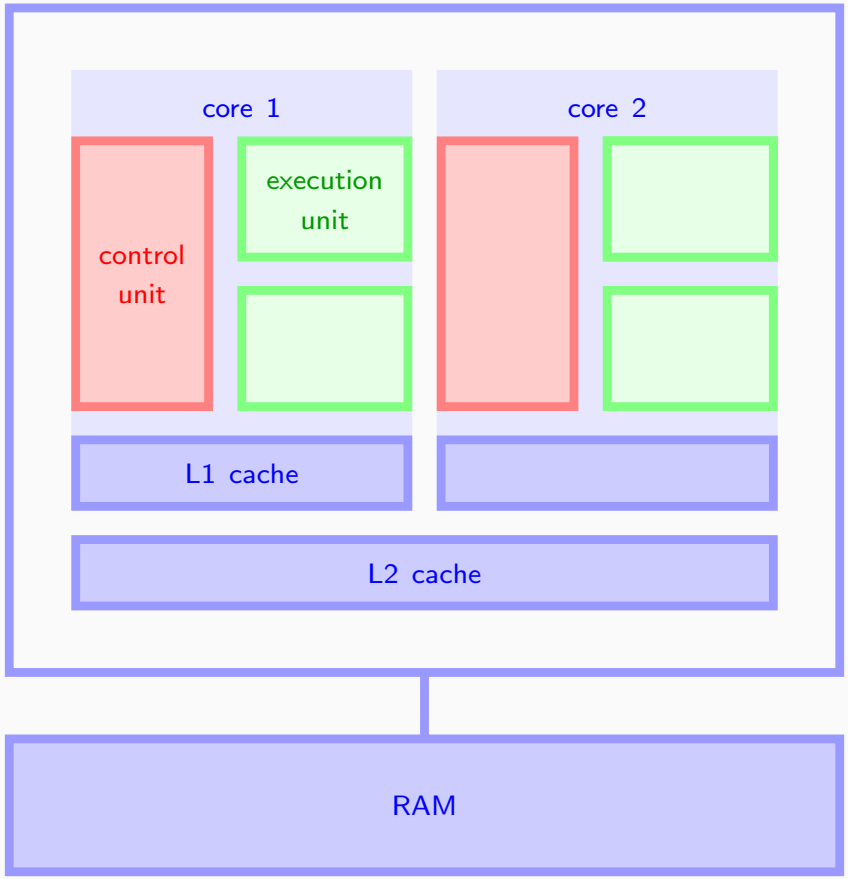
Parallel Computer Architecture

- In order to increase the computing power within a single machine, manufacturers offer parallel computer by aggregating CPUs.

Multi-core processor

A multi-core processor is a processor on a single integrated circuit (IC) with two or more separate central processing units (CPUs), called cores

- Control and execution unit, L1 cache duplicated ;
- L2, L2 caches, I/O unit are shared ;
- Example : Intel Cascade Lake up to 20 cores.

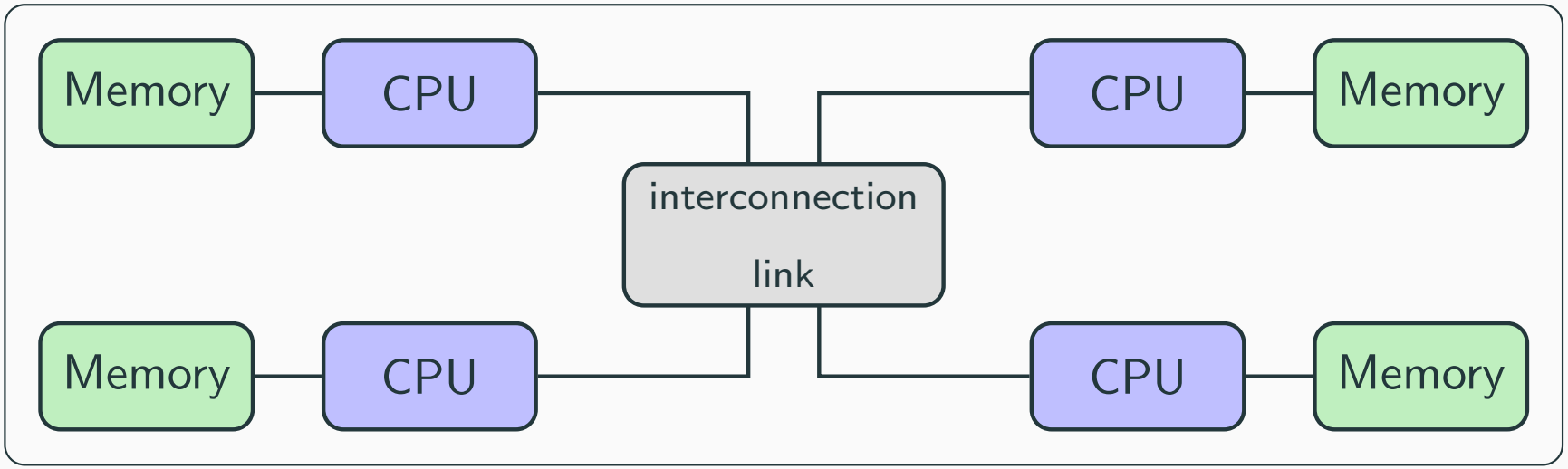


Parallel Computer Architecture

- In order to increase the computing power within a single machine, manufacturers offer parallel computer by aggregating processors.
- Parallel computers differ in the way the memory is accessed:
  - shared memory architecture
  - distributed memory architecture

Shared memory architecture

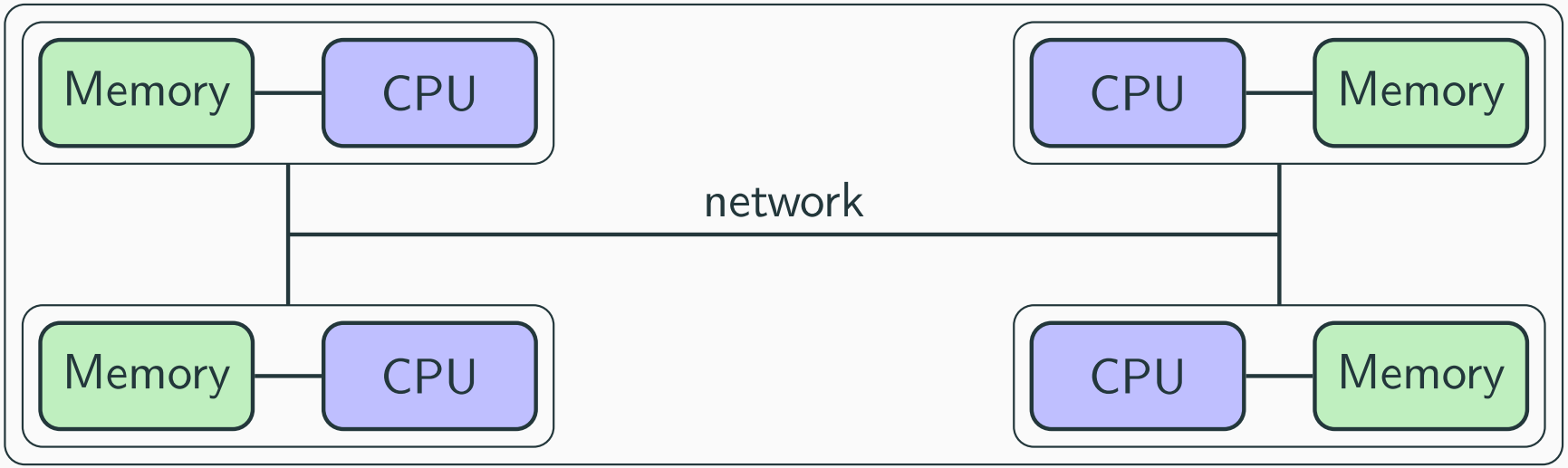
- Shared memory architecture refer to a parallel computer with several central processing units (CPUs) share full access to a common memory.
- NUMA (*Non-Uniform Memory Access*) architecture :



- Each processor is connected with its own dedicated memory.
- Each memory combines to make a single address space so that each processor can access the entire memory.
- The access time to memory is nonuniform : acces to local memory is faster than access to remote memory.

Distributed memory system

- Distributed memory system refers to multiprocessors computer system in which each processor has its own private memory.
- Each processor is connected to each other by a network.
- Computational tasks can only operate on local data, and if remote data are required, the computational task must communicate with one or more remote processors.

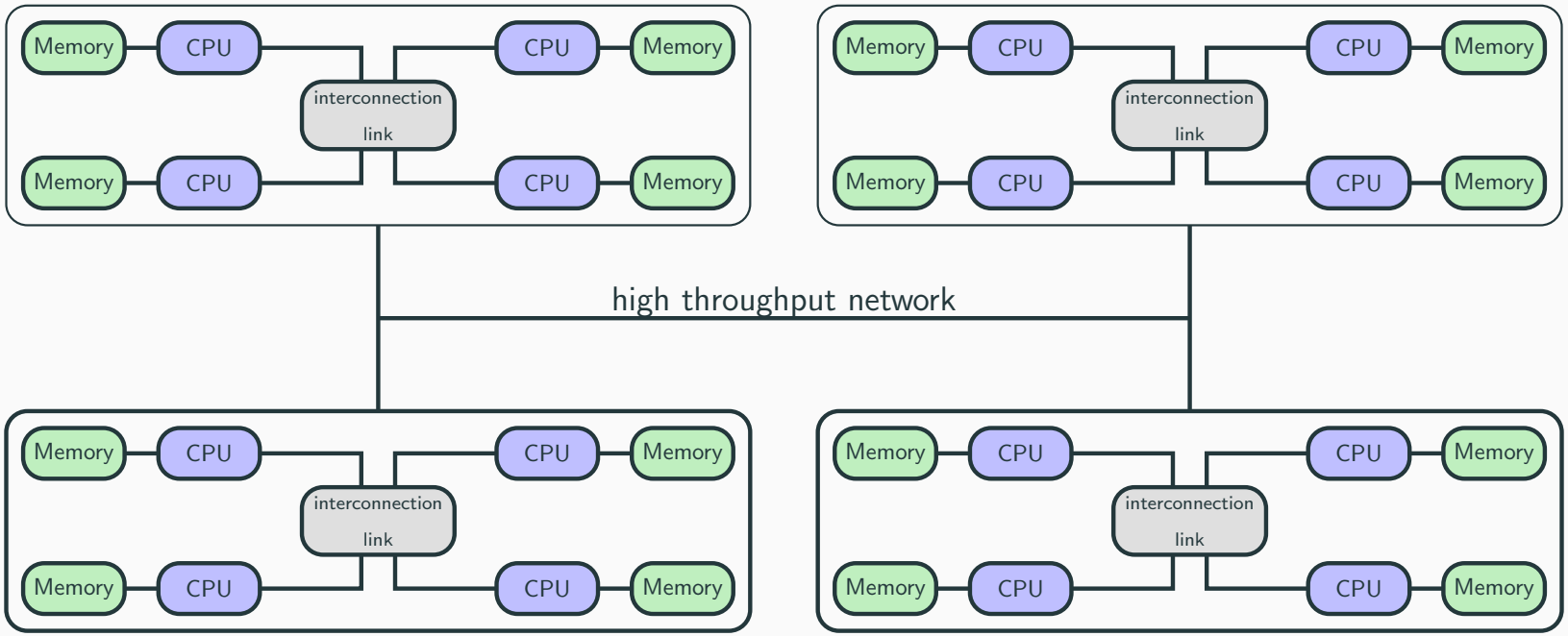


Distributed memory system : network

- The network is a key point of this type of architecture since it determines the data access speed of a neighboring processor.
- The network have to provide high bandwidth and low latency.

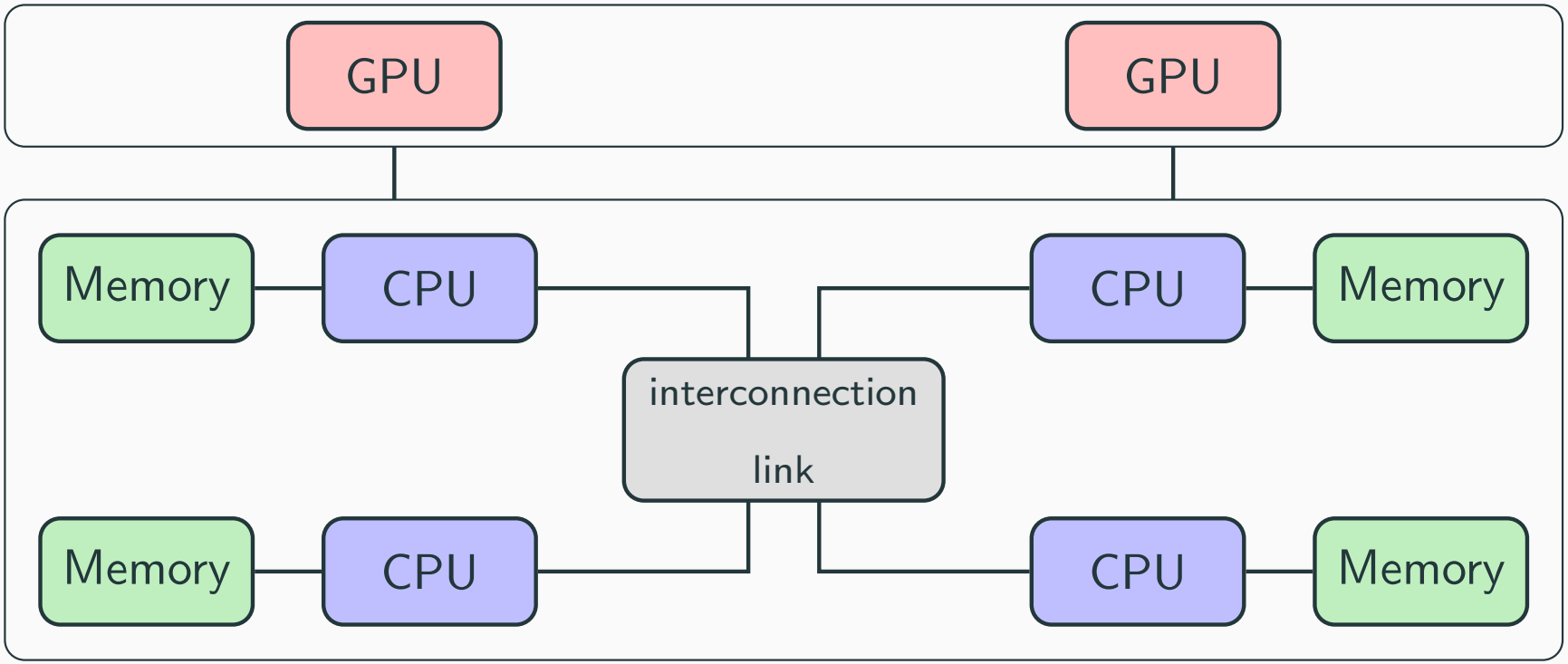
Hybrid architecture

- Today, most HPC systems combine shared memory and distributed memory architectures.
- These systems are made up of shared memory systems (computing nodes) linked together by an interconnection network.



GPU

- General-purpose computing on graphics processing units (GPGPU) is the use of a GPU, which typically handles computation only for computer graphics, to perform computation instead CPU.
- More and more HPC systems have computing nodes equipped with one or several GPU.



Top 500

The TOP 500 ranks and details the 500 most powerful computer systems in the world (available on www.top500.org)

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461

List of the 5 most powerful computer systems (November 2024)

France ranking

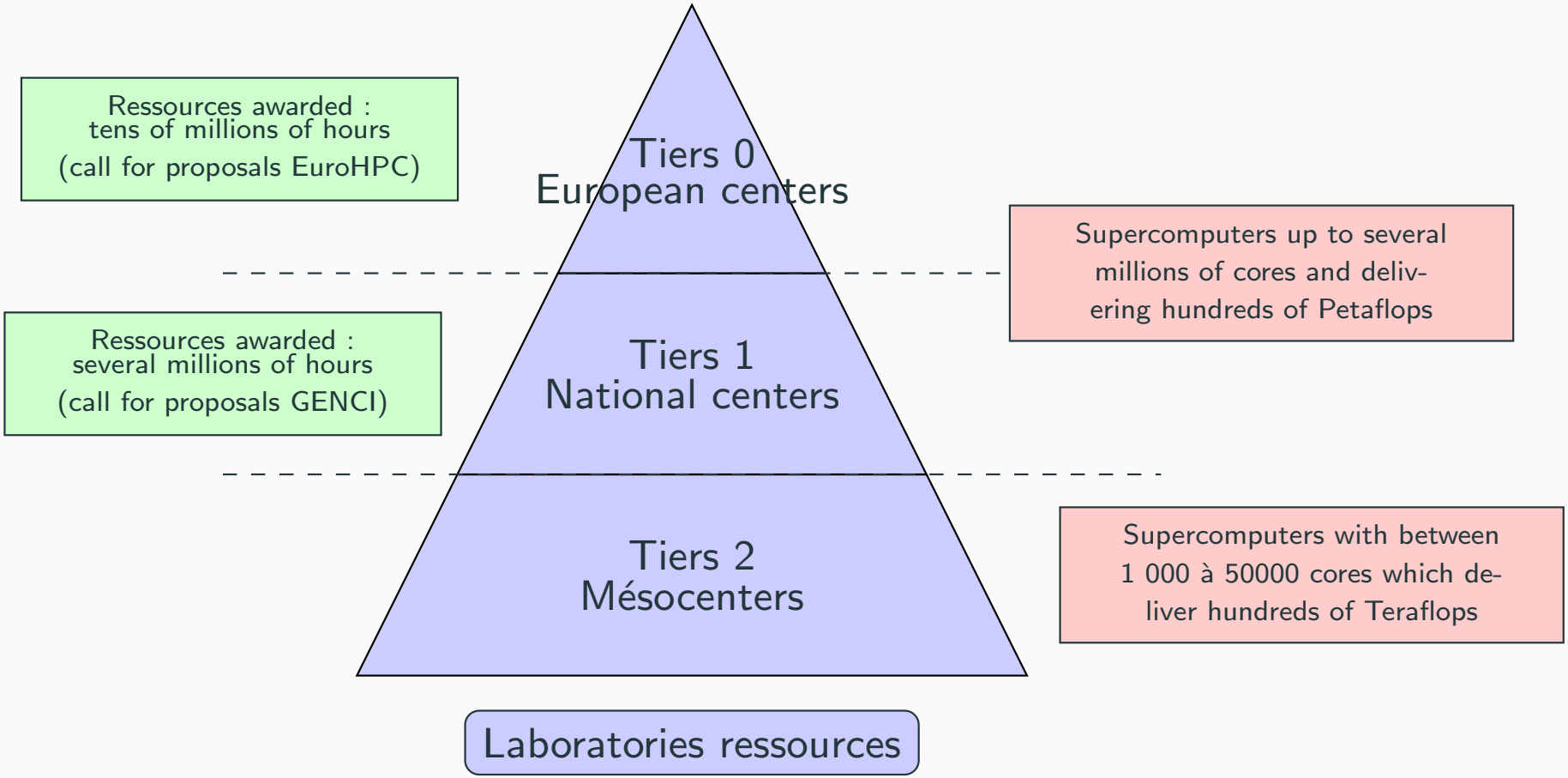
Country	Nb. of supercomputer	Theoretical peak perf.
United States	172 (34.4%)	10264 Pflops
China	63 (12.6%)	548 Pflops
Germany	41 (8,2%)	586 Pflops
Japan	34 (6,8%)	1249 Pflops
France	**24 (4,8%)**	**431 Pflops**
Italy	14 (2.8%)	1097 Pflops

Source : Top 500 list, November 2024

General informations

Table of contents

Organization of computing resources in France for research



- Examples of EuroHPC project available sur https://eurohpc-ju.europa.eu

Mesocenter Cholesky

- The mesocenter Cholesky was born from the initiative of research units of IP Paris : CMLS, CPHT, CMAP, LPP.
- Two labs joined the project (LMS, UMA, LOB in progress).
- The system administration and the user support is carried out by a collaboration of engineers of each partner (IDCS team).
- User guide available on https://meso-ipp.gitlab.labos.polytechnique.fr/user_doc/

Account creation

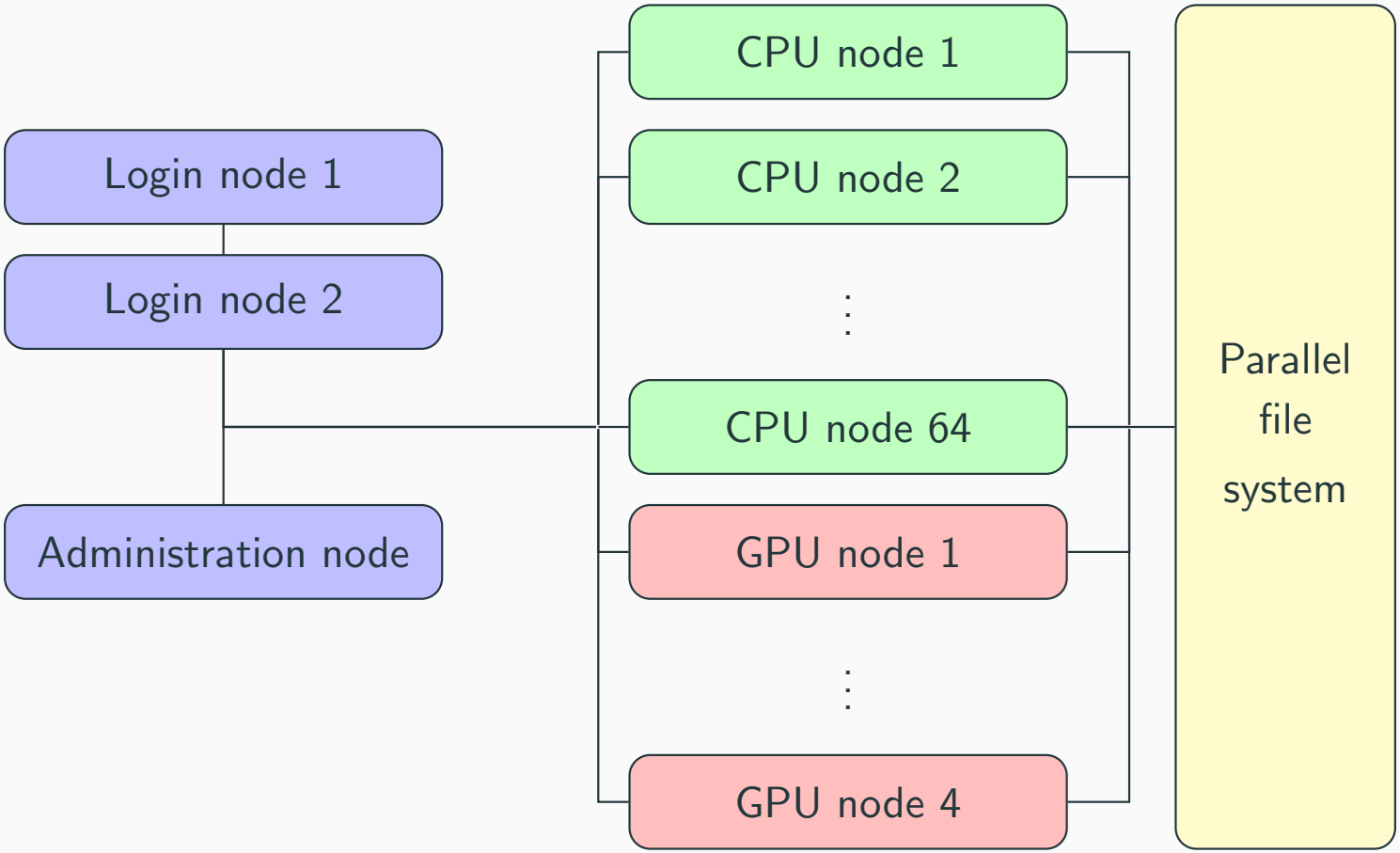
- Each user account must be associated with a project.
- The call for projects is permanent, the submission of applications is possible throughout the year.
- Only a permanent member can submit a project (project coordinator).
- Each project coordinator can add a user to the project.
- The only prerequisite for creating an account is that the user is registered in XAJAM (Ecole Polytechnique directory).

Hardware configuration

- one login node and one administration node
- 68 compute CPU nodes :
  - 2 processors Intel Xeon Gold 6230 (2.10GHz, 20 cores)
  - 192 Gb RAM (eight nodes with 384 Gb)
  - $\rightarrow$ 2720 compute CPU cores
- 5 GPU nodes
  - 2 nodes with Nvidia V100 GPU (NVlink)
  - 3 nodes with Nvidia A100 GPU (NVlink)
  - $\rightarrow$ 16 Nvidia GPU
- a Mellanox InfiniBand HDR (High Data Rate) interconnection network 200Gb/s (100Gb/s per CPU or GPU node)
- a parallel file system BeeGFS : 385 TB of usable space
- theoretical peak performance : 360 Tflops

Cluster architecture

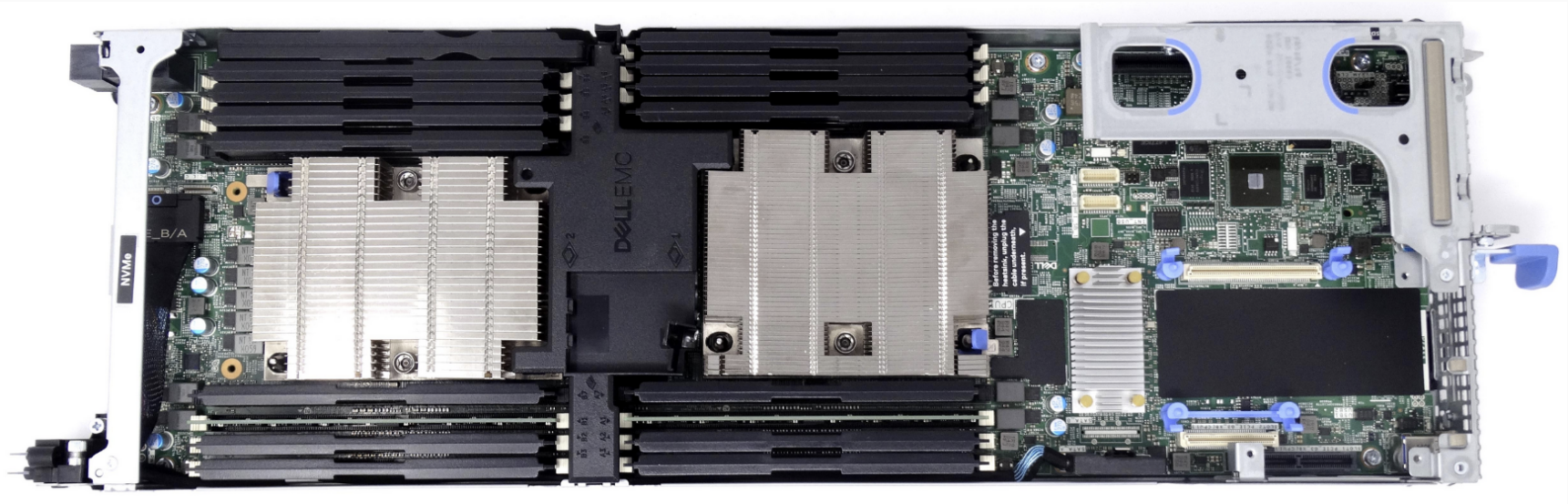
- Cholesky is parallel computer system comprising an integrated collection of independent nodes interconnected with a high-bandwidth network.
- All the nodes access a parallel file system (BeeGFS).



Cholesky

Cholesky on two racks

CPU node



Cholesky CPU node

Connection

You can connect to cholesky, using SSH with the following command :

```
$ ssh firstname.lastname@cholesky.mesocentre.idcs.polytechnique.fr
Last login: Fri Jan 14 17:57:30 2022 from europe.polytechnique.fr
#################################################################
#                        I.D.C.S                                #
#################################################################
#            !!! Acces autorise uniquement !!!                  #
# Deconnectez vous immediatement si vous n'etes pas autorise    #
#          toute action sera tracee et enregistree              #
# -------------------------------------------------------------#
#              !!! Authorized access only !!!                   #
#             Disconnect immediately otherwise                  #
#        All action will be monitored and recorded              #
#################################################################

#############################################################################
#  Vous etes connecte a une ressource administree par le mesocentre IDCS    #
# ------------------------------------------------------------------------- #
#         email : idcs.meso.support@polytechnique.fr                        #
# documentation : http://meso-ipp.gitlab.labos.polytechnique.fr/user_doc/   #
#############################################################################
firstname.lastname@cholesky:~$
```

Connection

- SSH key authentification is the only method to access Cholesky
- Use your lab SSH gateway or the school VPN (GlobalProtect)
- For Windows users, you can install git-bash (a tool from Git for Windows) or MobaXterm

Important

- After logging in, you are connected to the login node, from which you can edit files, compile your code and submit jobs.
- Editors available : vim, nano (you can also use remote ssh extension of vscode)
- Do not use the login node to run job.

SSH configure file

The command can be simplify from

`$ ssh firstname.lastname@cholesky.mesocentre.idcs.polytechnique.fr`

to

`$ ssh cholesky`

by writing on your laptop in `$HOME/.ssh/config`:

```
host <gateway>
   User firstname.lastname
   Hostname <gateway.polytechnique.fr>
   PubkeyAuthentication yes
   ForwardAgent yes
host cholesky
   Hostname cholesky.mesocentre.idcs.polytechnique.fr
   User firstname.lastname
   PubkeyAuthentication yes
   ForwardAgent yes
   ProxyCommand ssh -W %h:%p <gateway>
# or
#   ProxyJump <gateway>
```

Shared storage

You have access to two separate directories shared for all the nodes.

The home directory

- to store your source files and light input datas
- per user quotas of 10 Gb
- the environment variable $HOME contains the path of the home directory

The workdir directory

- to store your binary files and results of job
- per user quotas of 200 Gb
- the variable $WORKDIR contains the path of the workdir directory

```
$ cd $WORKDIR/
$ pwd
/mnt/beegfs/workdir/firstname.lastname
```

Shared storage

The project directory

- request by the project manager
- per project quotas of 200 Gb
- path : `/mnt/beegfs/project/project_name`

```
$ cd /mnt/beegfs/project/project_formation
$ ls
$ formation_cholesky_part01.pdf  mxv  seq_hello_world
```

Shared storage : quota command

```
$ cholesky_quota -h
Usage: /usr/local/sbin/cholesky_quota [OPTIONS] [USER]

Get USER's quota informations on Cholesky HPC cluster.

Options:

    [-m | --home]                 Get HOME quota informations.
    [-w | --work]                 Get WORKDIR quota informations.
    [-p | --project[=<PROJECT>]]  Get all project(s) or specified <PROJECT> quota(s) informations.
    [-a | --all]                  Get all quotas informations.
    [-c | --check]                Check and display file permission error(s).
    [-h | --help]                 Prints help.

$ cholesky_quota -a
Quota Name  Directory                                 Used   Hard    Use%    Error(s)
----------  ---------                                 ----   ----    ----    --------
HOME        /mnt/beegfs/home/IDCS/firstname.lastname  3GiB   10GiB   30.11%  No
WORKDIR     /mnt/beegfs/workdir/firstname.lastname    61GiB  200GiB  30.37%  No
bench       /mnt/beegfs/project/bench                 30MiB  100GiB  0.03%   No
formation   /mnt/beegfs/project/formation             0B     200GiB  0.00%   No

Quota error(s): please check file permissions given by -c option.

The quotas are refreshed every 10 minutes. All the information is not in real time
and may not reflect your real storage occupation.
```

Quota: beware to the group owner

- The quotas are computed from the size of files belonging to groups:
  - home: firstname.lastname
  - workdir: work_firstname.lastname
- When moving files or directories from the `$HOME` to the `$WORKDIR`, the group is preserved.
- The files or directories are still counted in the `$HOME` quota (10 Go).
- To change it:

```
user@mycomputer:~$ mv $HOME/my_dir $WORKDIR/
user@mycomputer:~$ ls -lh $WORKDIR
drwxrwxr-x 2 aurelien.canou aurelien.canou    1 29 janv. 11:46 my_dir
user@mycomputer:~$ chgrp -R work_$USER $WORKDIR/my_dir
drwxrwxr-x 2 aurelien.canou work_aurelien.canou    1 29 janv. 11:46 my_dir
```

File transfer

You can transfer files from the cluster or to the cluster, you can use the shell command `scp` from your computer.

```
user@mycomputer:~$ ls
file01.txt  mydir
user@mycomputer:~$ scp file01.txt
↪  firstname.lastname@cholesky-login.mesocentre.idcs.polytechnique.fr:
↪  /mnt/beegfs/home/firstname.lastname
Enter passphrase for key '/home/username/.ssh/id_rsa':
file01.txt                                    100%  132KB 132.3KB/s   00:00
user@mycomputer:~$ ls mydir
file02.txt  file03.txt
user@mycomputer:~$ scp -r mydir
↪  firstname.lastname@cholesky-login.mesocentre.idcs.polytechnique.fr:
↪  /mnt/beegfs/workdir/firstname.lastname
file02.txt                                    100%  132KB 112.6KB/s   00:00
file03.txt                                    100%  132KB 117.3KB/s   00:00
```

Exercise

1. connect to cholesky
2. create in your `$HOME` directory a file with the command
   `touch tmp.txt`
3. copy the file `tmp.txt` in your `$WORKDIR` directory
4. check your quota
5. copy from `/mnt/beegfs/project/formation` the file
   `cholesky_training_part01.pdf` on your machine
6. copy a file from your machine on your working directory on cholesky

Job scheduling and execution: SLURM

Table of contents

Job scheduling system : SLURM

- *SLURM* (Simple Linux Utility for Resource Management) is an open source cluster management and job scheduling system .
- *SLURM* is used in small and large Linux clusters (e.g. occigen@cines, jean-zay@idris, ...).
- All the information on: `https://slurm.schedmd.com`

SLURM - CPU partitions

Partition	Max. nb of cpus per job	Max. Walltime
cpu_seq	1	100 h
cpu_shared	40	100 h
cpu_dist	400	24 h
cpu_test	80	30 m

- *cpu_seq*: for sequential (i.e. one core) jobs.
- *cpu_shared*: for shared memory jobs (multithreading, OpenMP, etc...).
- *cpu_dist*: for distributed memory jobs (MPI) ; use of 2 nodes minimum.
- *cpu_test*: 'floating' queue for debugging jobs ; use of 6 nodes maximum.

SLURM - GPU partitions

Partition	Max. nb of gpus per user	Max. Walltime
gpu	8	24 h
gpu_v100	8	24 h
gpu_a100	8	24 h
gpu_a100_80g	8	24 h

The partition *gpu* encompasses:

- *gpu_v100*: NVIDIA V100 GPUs with 32 GB of GRAM
- *gpu_a100*: NVIDIA A100 GPUs with 40 GB of GRAM
- *gpu_a100_80g*: NVIDIA A100 GPUs with 80 GB of GRAM

Show partitions : sinfo

Show information about the cluster's node

```
# show information of all partitions
$ sinfo

# show information of a specific partition
$ sinfo --partition=cpu_dist

# use of format to list specific fields
$ sinfo --format="%20P %.5a %.15l %.8D %.12T"

# show informations of all nodes
$ sinfo -Nl
```

Note: you can create aliases for these commands:

```
$ alias sinfo_long='sinfo -Nl'
$ sinfo_long
```

Job scheduling system : SLURM

- In order to submit jobs on the cluster, you must describe the resources (cores, time) used to the *SLURM* task scheduler.
- *SLURM* will execute jobs on remote compute node(s) as soon as the resources described will be available.
- There are 2 ways to run jobs on Cholesky :
  - using a *SLURM* script
  - using a interactive *SLURM* job (not discussed here)

Launching a job: SLURM script

- **NEVER** launch a job on the login node !
- ... except the compilation
- but submit a job with the command : `sbatch myscript.sh`
- *myscript.sh* is a *bash* script with *SLURM* compatible directives
- *SLURM* compatible directives are called SBATCH directives
- *myscript.sh* tells *SLURM* which ressources to allocate to run the job
- the content of *myscript.sh* varies according to the type of job (sequential, OpenMP, MPI, etc...)

Launching a job : example

A typical `sbatch` compatible script:

```
#! /bin/bash

#SBATCH --job-name=job00     # name of the job
#SBATCH --nodes=1            # number of nodes
#SBATCH --ntasks=1           # number of tasks (e.g. MPI processes)
#SBATCH --ntasks-per-node=1  # number of tasks per node
#SBATCH --cpus-per-task=1    # number of cpus per task (e.g. OpenMP threads)
#SBATCH --time=00:10:00      # maximum execution time (format: HH:mm:ss)
#SBATCH --output=job.%j.out  # name of the standard output file
#SBATCH --error=job.%j.err   # name of the error output file
#SBATCH --partition=cpu_seq  # partition to use
#SBATCH --account=formation  # account

# Job preparation
...

# launch the program
./my_program
```

```
# To submit the job:
$ sbatch myscript.sh
Submitted batch job 579005
```

squeue

Show the jobs that are scheduled

```
# standard formatting
$ squeue

# list job for USER only
$ squeue -u $USER

# list job for ACCOUNT only
$ squeue --account=formation

# special formatting of jobs for all users
squeue --format "%.10i %.10P %.20j %.20u %.10T %.13M %.13l %.6D %.4C"

# special formatting of jobs for USER only
squeue -u $USER --format "%.10i %.10P %.20j %.20u %.10T %.13M %.13l %.6D %.4C"
```

Note: you can create aliases for these commands:

```
$ alias squeue_u='squeue -u $USER'
$ squeue_u
```

Practice - first_sbatch.sh

- Submit the job `first_batch.sh` with the command `sbatch`.
- run the `squeue` command, locate your job and try to understand the various fields
- use `squeue_u` instead
- read and link with the SBATCH directives what is written in the output *.out
- read the error file
- modify the script to make it run on several nodes with several tasks per node.

Software environment

Table of contents

Lmod : a environment module system

- Several software components (compiler, libraries, software, etc.) are available on the Cholesky cluster.
- Lmod allows to load these software components into your user's environment.
- Lmod allows to use hierarchical modules what that means is that loading a module makes more modules available.
- There are 3 distinct categories of modules, Core Modules, Compiler Modules, and Compiler/MPI Modules :
    - Core modules are modules built against the system compiler
    - Compiler modules are built against a specific compiler
    - Compiler/MPI modules are built against a specific compiler/MPI combination.

Available modules

The `module avail` command reports modules that can be loaded directly on your environment :

```
$ module purge
$ module avail

------------------------ /mnt/beegfs/softs/opt/modulefiles/core -------------------------
   Eigen/3.3.9          git/2.31.1                  python/3.9.2
   R/4.0.5              gnuplot/5.4.2               ruby/3.0.2
   abaqus/2021          htop/3.0.5                  singularity/3.4.1
   anaconda3/2020.11    intel_compiler/19.1.3.304   tcsh-csh/6.22.04
   clang/13.0.0         julia/1.6.1                 texlive/2021
   cmake/3.19.7         mathematica/12.3.1          triqs/3.0.0
   cuda/10.2            matlab/2021a                w2dynamics/1.1.1
   gcc/10.2.0           nvhpc/21.3

Utilisez "module spider" pour trouver tous les modules possibles.
Utilisez "module keyword key1 key2 ..." pour chercher tous les modules possibles qui
correspondent à l'une des clés (key1, key2).
```

Available modules

The `module avail` command reports modules that can be loaded directly on your environment :

```
$ module avail

-----------------------/mnt/beegfs/softs/opt/modulefiles/core --------------------------
   Eigen/3.3.9              gcc/10.2.0                          mambaforge/22.11.1-4
   R/4.0.5                  gcc/13.2.0                   (D)    mathematica/12.3.1
   abaqus/2021              git/2.31.1                          matlab/2021a
   abaqus/2023       (D)    gnuplot/5.4.2                       nvhpc/21.3
   anaconda3/2020.11        gurobi/9.5.2                        ruby/3.0.2
   anaconda3/2023.09 (D)    htop/3.0.5                          singularity/3.4.1
   clang/13.0.0             intel_compiler/oneapi_2023.0.0      tcsh-csh/6.22.04
   cmake/3.19.7             intel_compiler/oneapi_2023.1.0      texlive/2021
   cmake/3.26.3             intel_compiler/oneapi_2023.2.0      triqs/3.0.0
   cmake/3.29.3      (D)    intel_compiler/19.1.3.304    (D)    valgrind/3.23.0
   cuda/12.2                julia/1.10.4                 (D)    vaspkit/1.4.0      (D)
   cuda/12.4         (D)    lua/5.4.4                           w2dynamics/1.1.1

  Où:
   D:  Default Module

Utilisez "module spider" pour trouver tous les modules possibles.
Utilisez "module keyword key1 key2 ..." pour chercher tous les modules possibles
qui correspondent à l'une des clés (key1, key2).
```

Available modules

The `module spider` command reports all the modules that can be loaded on your environment.

1. `module spider` : Report all modules
2. `module spider <name>`: Report all the versions for the modules that match `<name>`
3. `module spider <name/version>` : Report detailed information on a particular module version

```
# live demo
```

Load modules

The `module load` or `ml` command adds one or more modules to your current session.

```
$ module load gcc/13.2.0
```

Note : after loading a compiler, if you load a compiler, the command `module load` reports modules which are built against this compiler

```
# live demo
```

List, unload, purge modules

The `module list` or `ml` command lists the modules which are currently loaded in your environment.

```
$ module list
Currently Loaded Modules:
  1) gcc/13.2.0   2) fftw/3.3.10
```

The `module unload` command unloads a software component.

```
$ module unload fftw/3.3.10
$ module list
Currently Loaded Modules:
  1) gcc/13.2.0
```

The `module purge` command unloads all loaded software components.

```
$ module purge
$ module list
No modules loaded
```

Swap modules

The `module swap` command allows to switch from a compiler and/or MPI library to another compiler and/or MPI library

```
$ module list

Currently Loaded Modules:
  1) gcc/10.2.0   2) fftw/3.3.9

$ module swap gcc/10.2.0 intel_compiler/19.1.3.304

Dû à un changement dans la variable MODULEPATH, les modules suivants ont été rechargés :
  1) fftw/3.3.9

$ module list

Currently Loaded Modules:
  1) intel_compiler/19.1.3.304   2) fftw/3.3.9
```

Exercise

1. how many version of fftw library are installed on cholesky
2. load the version built against `gcc/13.2.0`
3. switch to the version built against `intel_compiler/oneapi_2023.2.0`
4. launch a job which loads `gcc/13.2.0` and lists the modules loaded

Conda Environment for python

- A conda environment is a Python tool for dependency management and project isolation (link to the documentation).
- It allows to install packages on a host on which you do not have admin privileges.
- On cholesky, you can create environments with conda by loading anaconda module.

```
$ module load anaconda3/2023.09
```

Create environment

To create an environment named `myenv` with a specific version of Python

```
$ conda create -n myenv python=3.8
```

When conda asks you to proceed, type `y` :

```
$ Proceed ([y]/n)?
```

To activate the environnment named `myenv` :

```
$ source activate myenv
(myenv) $
```

- Activating a conda environment modifies the PATH and shell variables to point to the specific isolated Python set-up you created.
- The command prompt will change to indicate which conda environemnt you are currently.

Manage python packages into environment

To install python packages :

```
(myenv) $ conda install numpy mpmath
```

To search a python package :

```
(myenv) $ conda search scipy
```

To install a specific python package :

```
(myenv) $ conda install scipy=1.7.3
```

To list the environnment named `myenv` :

```
(myenv) $ conda list
```

Deactivate and remove an environment

To deactivate an environment :

```
(myenv) $ conda deactivate
(base) $
```

- Conda removes the path name for the currently active environment from your system command.

To delete an environmennt

```
(base) $ conda remove -n myenv --all
```

Compilation

Table of contents

Compilation: why and what for ?

- The source code of a program is written in languages that are intelligible to programmers.
- In order to be executed by the the central processing unit (CPU), the source code must be translated into a machine code for the CPU to execute.
- For interpreted langages like Python or Matlab, this translation is done in real time during execution by an interpreter.
- For compiled langages like Fortran, C, C++, this translation is done by a compiler launched by the programmer in two step :
    - compiling step : the source code files is compiled into object code files ;
    - linking step : the object code files are combined to make an executable program (this step typically involves adding in any libraries that are required).

Example : hello world

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello world !\n");

    return 0;
}
```

GNU compiler suite :

```
$ module load gcc/10.2.0
$ gcc -c hello_world.c -o hello_world.o # Compilation stage
$ gcc hello_world.o -o hello_world      # Linking stage
```

Intel compiler suite :

```
$ module swap gcc/10.2.0 intel_compiler/19.1.3.304
$ icc -c hello_world.c -o hello_world.o # Compilation stage
$ icc hello_world.o -o hello_world      # Linking stage
```

Example with an example library (BLAS MKL)

Product matrix vector

```c
#include <stdio.h>
#include <cblas.h>

int main(int argc, char **argv)
{
    ...
    cblas_dgemv(CblasRowMajor, CblasNoTrans, n, n, 1., a, n, x, 1, 0., y, 1);
    ...
    return 0;
}
```

It is necessary to specify the header files and libraries path and the libraries :

```
$ module load intel_compiler/19.1.3.304
$ export MKL_ROOT=/mnt/beegfs/softs/intel/compilers_and_libraries_2020.4.304/linux/mkl/
$ icc -c mxv_blas.c -o mxv_blas.o -I${MKLROOT}/include
$ icc mxv_blas.o -o mxv_blas -L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread
↪  -lmkl_core -liomp5 -lpthread -lm -ldl
```

Example with an external library (BLAS MKL)

If module intel mkl is loaded, it is unnecessary to specify the header files and libraries path:

```
$ module  intel_mkl/2020.0.4
$ icc -c mxv_blas.c -o mxv_blas.o
$ icc mxv_blas.o -o mxv_blas  -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5
↪  -lpthread -lm -ldl
```

Compilation using a simple Makefile

Make is a tool that builds executable programs from source code by reading files called Makefile which specify set of instructions to build the target program.

```
$ cat Makefile
CC = gcc
LD = gcc

hello_world: hello_world.o
	$(LD) $^ -o $@

hello_world.o: hello_world.c
	$(CC) $(FLAGS) -c $< -o $@

clean:
	rm -f *.o hello_world

$ make
gcc  -c hello_world.c -o hello_world.o
gcc hello_world.o -o hello_world
```

Compilation using a simple Makefile with external library

```
$ cat Makefile
CC = icc
LD = icc

FLAGS = -O3

MKLROOT=/mnt/beegfs/softmake/intel/compilers_and_libraries_2020.4.304/linux/mkl
MKL_INC_DIR = ${MKLROOT}/include
MKL_LIBS =  -L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5
↪  -lpthread -lm -ldl

all: mxv_blas

mxv_blas: mxv_blas.o
        $(LD) $^ -o $@ $(MKL_LIBS)
mxv_blas.o: mxv_blas.c
        $(CC) $(FLAGS) -I$(MKL_INC_DIR) -c $< -o $@

clean:
        rm -f *.o mxv_blas

$ make
icc -O3 -I/mnt/beegfs/softs/intel/compilers_and_libraries_2020.4.304/linux/mkl/include   -c
↪  mxv_blas.c -o mxv_blas.o
icc mxv_blas.o -o mxv_blas
↪  -L/mnt/beegfs/softs/intel/compilers_and_libraries_2020.4.304/linux/mkl/lib/intel64
↪  -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm -ldl
```

Compilation using a more complex Makefile

```
$ cat Makefile
# Choose the compiler (CC) and linker (LD)
ifeq ($(compiler),intel)
 CC = icc
 LD = icc
else
 CC = gcc
 LD = gcc
endif

FLAGS = -O2

all: hello_world

hello_world: hello_world.o
	$(LD) $^ -o $@

hello_world.o: hello_world.c
	$(CC) $(FLAGS) -c $< -o $@

clean:
	rm -f *.o hello_world
```

Compilation using a more complex Makefile

Using GNU compiler suite :

```
$ module load gcc/10.2.0 # load the module
$ make
$ make compiler=intel
gcc -O2 -c hello_world.c -o hello_world.o
gcc hello_world.o -o hello_world
```

Using Intel compiler suite :

```
$ module load intel_compiler/19.1.3.304 # load the module
$ make compiler=intel # explicitly specify the compiler to be intel
icc -O2 -c hello_world.c -o hello_world.o
icc hello_world.o -o hello_world
```

CMAKE

CMake is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method (see `https://cmake.org/`)

Compilation Options

- The compilation options depend on the compiler suite and on the langage (Fortran,C).
- Straightforward optimization for all compilers:
  - `-O0` = no optimization
  - `-O1` = some optimisations are performed which allow moderate acceleration
  - `-O2` = level of optimisation which includes vectorisation, inlining, loop unrolling
  - `-O3` = optimisations are more aggressive than in -O2, with prefetching, loop transformations
- Note that the activation of the full optimization may induce no replicable results. A careful checking is therefore necessary by comparing the results with -O0, -O2 and -O3.
- For further informations, the users can read:
  - `http://www.idris.fr/eng/jean-zay/cpu/jean-zay-cpu-comp_options-eng.html`
  - the GNU and Intel manuals : `man gcc` and `man icc`

Exercise: hello world

1. copy the directory `/mnt/beegfs/project/formation/hello_world` into your `$WORKDIR`.
2. load the module gcc/13.2.0
3. compile the program with gcc using the Makefile
4. submit the script job.sh via *sbatch*
5. view the result
6. swap module gcc by module icc
7. compile the program with icc using the Makefile
8. modify the script job.sh and submit script

Vectorization and memory hierarchy

Table of contents

Performances des processeurs

- Une des principales caractéristiques du processeur est la **fréquence d'horloge** qui détermine la durée d'un cycle.
- Un processeur exécute chaque instruction en un ou plusieurs cycles donc plus la fréquence augmente plus le processeur exécute d'instructions par seconde.
- L'augmentation de la performance des processeurs a reposé, en partie, sur la croissance de la fréquence.
- Depuis quelques années, sa croissance est de plus en plus limitée. Actuellement, la fréquence stagne autour de 3 GHz (soit un temps de cycle de 0.33 ns).
- Principales limites à sa croissance : la consommation électrique et la dissipation thermique du processeur.

$$Consommation\ électrique \propto Fréquence^3$$

- Conséquences : les constructeurs ne cessent d'optimiser l'architecture générale du processeur afin d'augmenter ses performances.

Unités vectoriels

- Les processeurs équipés d'unités vectorielles permettent de réaliser une même opération sur plusieurs données simultanément.

- Exemple d'une addition de vecteur :

  On remplace les 4 instructions :

$$\begin{array}{|c|} \hline x_1 \\ \hline x_2 \\ \hline x_3 \\ \hline x_4 \\ \hline \end{array} + \begin{array}{|c|} \hline y_1 \\ \hline y_2 \\ \hline y_3 \\ \hline y_4 \\ \hline \end{array} = \begin{array}{|c|} \hline x_1 + y_1 \\ \hline x_2 + y_2 \\ \hline x_3 + y_3 \\ \hline x_4 + y_4 \\ \hline \end{array}$$

  par une seule instruction :

$$\begin{array}{|c|} \hline x_1 \\ \hline x_2 \\ \hline x_3 \\ \hline x_4 \\ \hline \end{array} + \begin{array}{|c|} \hline y_1 \\ \hline y_2 \\ \hline y_3 \\ \hline y_4 \\ \hline \end{array} = \begin{array}{|c|} \hline x_1 + y_1 \\ \hline x_2 + y_2 \\ \hline x_3 + y_3 \\ \hline x_4 + y_4 \\ \hline \end{array}$$

  les 4 additions sont traités simultanément

- Le nombre d'éléments qui peuvent être traités simultanément dépend de la taille des registres vectoriels
- Les processeurs les plus récents sont équipés de registres de taille 512 bits (AVX512) qui permet de traiter 8 réels représentés sur 64 bits (double precision).

Vectorisation : mise en œuvre explicite

Exemple avec le jeu d'instruction AVX512 pour des unités vectorielles de taille 512 bits :

sans vectorisation :

```c
#include <stdio.h>

int main(void)
{
  int const n =10000;
  double a[n], b[n], c[n];

  for (int i=0; i<n; i++)
  {
    a[i] = i;
    b[i] = i;
  }

  for (int i=0; i<n; i++)
  {
      c[i] = a[i] + b[i];
  }

  return 0;
}
```

avec vectorisation :

```c
#include <stdio.h>
#include <emmintrin.h>

int main(void)
{
  int const n =10000;
  double a[n], b[n], c[n];
  __m512d* mm_c = (__m512d*) c;
  __m512d* mm_b = (__m512d*) b;
  __m512d* mm_a = (__m512d*) a;

  for (int i=0; i<n; i++)
  {
    a[i] = i;
    b[i] = i;
  }

  int nvec = n/8;
  for (int i=0; i<nvec; i++)
  {
      *mm_c = _mm512_add_pd(*mm_a,*mm_b);
      mm_c++;
      mm_b++;
      mm_a++;
  }

  return 0;
}
```

Vectorisation : mise en œuvre à la compilation

- On ne modifie pas le code original.
- On compile avec une option activant la vectorisation.

```c
#include <stdio.h>
int main(void)
{
int const n = 1024;
double a[n], b[n], c[n];

for (int i=0; i<n; i++)
{
  a[i] = i;
  b[i] = i;
}

printf("Adding vectors of size : %d\n", n);
for (int i=0; i<n; i++)
{
  c[i] = a[i] + b[i];
}
for (int i=0; i<n; i++) printf("%f  ", c[i]);
return 0;
}
```

```
> gcc -O2 -ftree-vectorize -fopt-info-vec -mavx512f add_vec.c
add_vec.c:15:3: optimized: loop vectorized using 64 byte vectors
add_vec.c:8:3: optimized: loop vectorized using 64 byte vectors
```

Vectorisation : mise en œuvre avec OpenMP

- On indique au compilateur qu'il est sans danger de vectoriser avec des directives.
- On active ces directives SIMD avec une option du compilateur (-fopenmp-simd pour gcc).

```c
#include <stdio.h>
int main(void)
{
  int const n =10000;
  double a[n], b[n], c[n];
  int i;

  for (i=0; i<n; i++)
  {
    a[i] = i;
    b[i] = i;
  }

  printf("Add a and b\n");
  #pragma omp simd
  for (i=0; i<n; i++) c[i] = a[i] + b[i];

  for (i=0; i<n; i++) printf("%f", c[i]);

  return 0;
}
```

```
> gcc -fopenmp-simd vec.c -o vec
```

Vectorisation : exemple du produit matrice vecteur

```
...
  for (int i=0; i<n; ++i)
  {
    y[i] = 0.;
    for (int j=0; j<n; ++j)
    {
      y[i] += a[i][j] * x[j];
    }
  }
...
```

```
> gcc -Ofast -march=native -fno-tree-vectorize mxv.c -o mxv
> ./mxv 40000
Matrix vector product (40000)
Elapsed time to compute matrix vector product : 2.303715
> gcc -Ofast -march=native mxv.c -o mxv
> ./mxv 40000
Matrix vector product (40000)
Elapsed time to compute matrix vector product : 1.040397
```

- Exemple réalisé avec le compilateur gcc 13.2 sur un processeur de type Intel Xeon Gold 6230 de 40 cores @ 2.10 GHz (Cascade lake)
- Dans cet exemple, la vectorisation apporte un gain de plus de 50%.

Exercise : product matrix vector

1. copy the directory `/mnt/beegfs/project/formation/vectorization` into your `$WORKDIR`.
2. load gcc/13.2.0 module
3. compile the program using the script compil.sh
4. detect vectorized loops
5. launch the job by using the batch script (`job.sh`)
6. what are the benefits of vectorisation ?

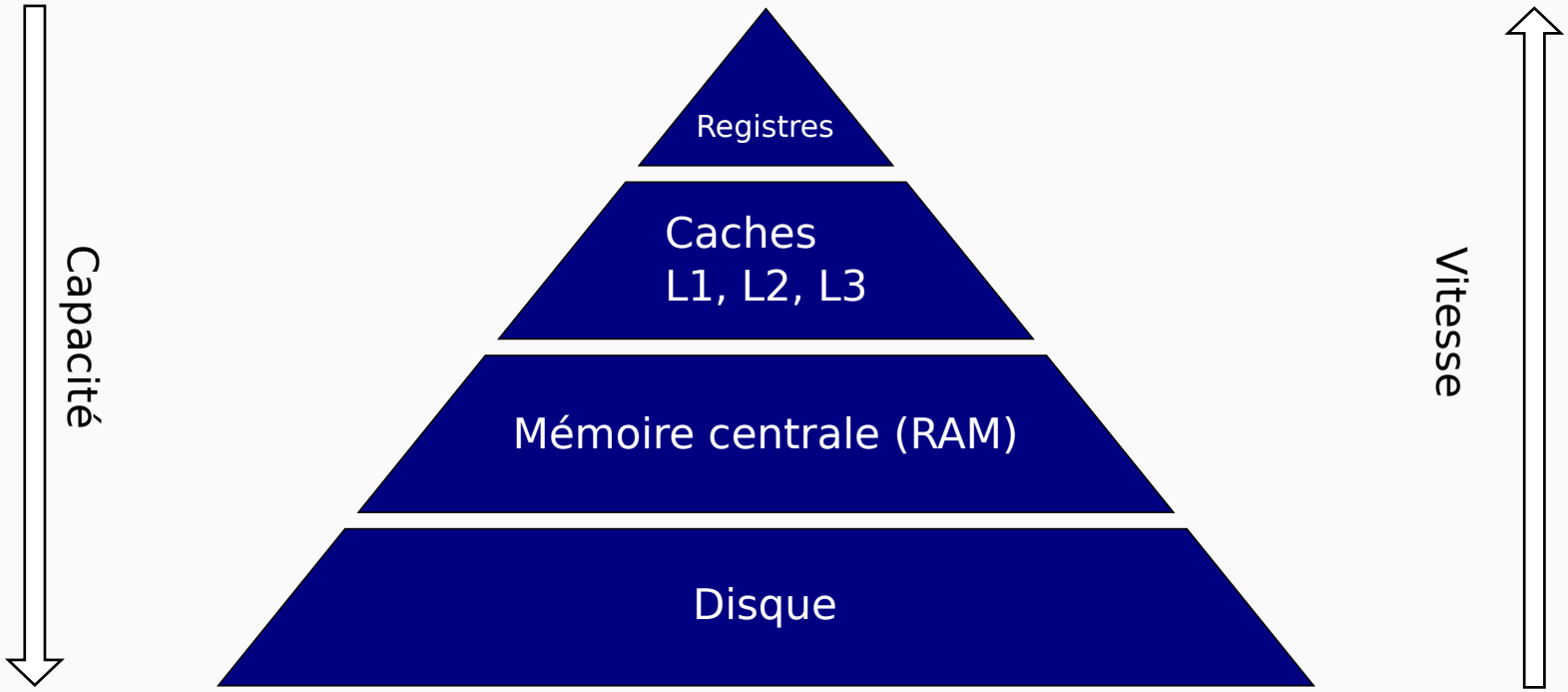
La mémoire

- La mémoire désigne les lieux de stockage des données.
- C'est depuis la mémoire que le processeur accède aux instructions du programme à exécuter ainsi qu'aux données nécessaires à son exécution.
- Il existe plusieurs niveaux de mémoire en fonction de son temps d'accès.

Hiérarchie mémoire

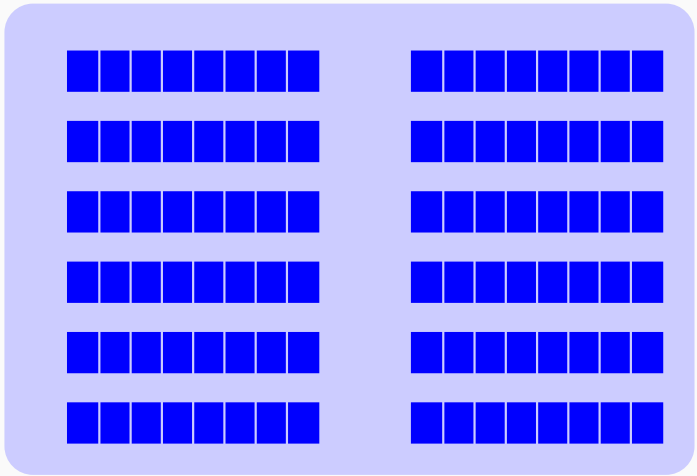
- Registres :
  - Mémoires intégrées au processeur
  - Accès rapide : en général, 1 cycle
- Caches :
  - Mémoires intermédiaires entre le processeur et la mémoire vive
  - Les caches sont hiérarchisés (L1, L2 et L3)
  - Accès : quelques cycles pour le cache L1 à quelques dizaines de cyles pour le cache L3
  - Taille : d'une dizaine de *Kio* à la dizaine de *Mio*
- Mémoire vive (RAM) :
  - Mémoire centrale d'une machine
  - Accès en quelques dizaines de cyles
  - Taille : de la dizaine à quelques centaines de *Gio*
- Disques
  - Accès en plusieurs millions de cycles.
  - Taille : plusieurs *Tio*.

Hiérarchie mémoire

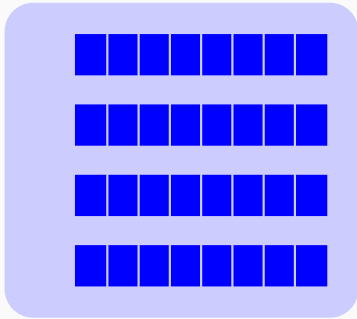


Caches

- Le cache est divisé en lignes (ou blocs) de mots.



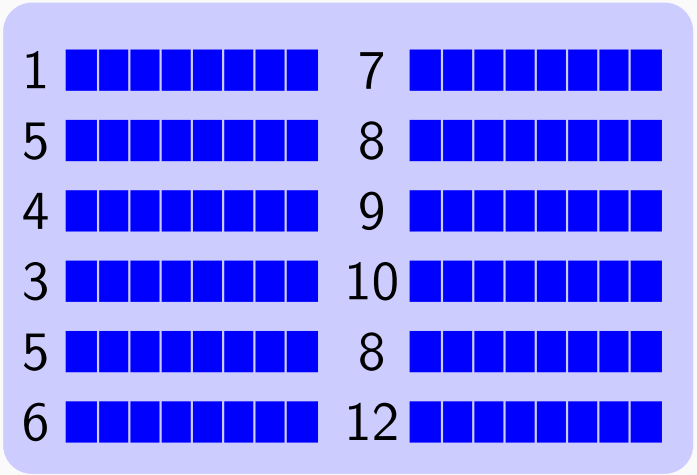
Mémoire centrale



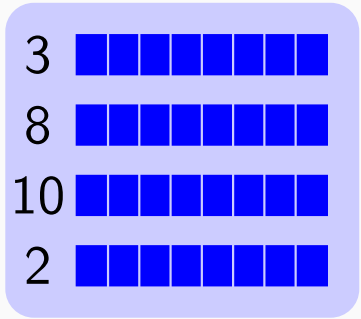
Cache

Caches

- Les lignes présentes dans le cache ne sont que des copies temporaires des lignes de la mémoire centrale.



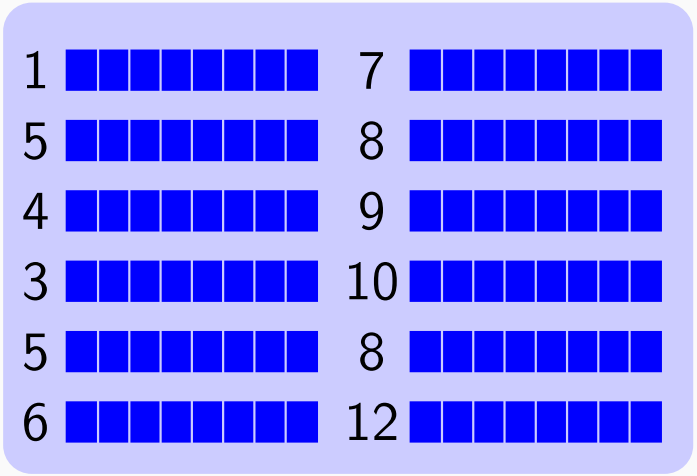
Mémoire centrale



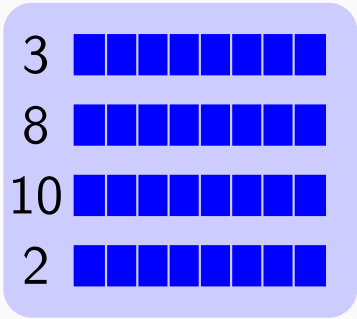
Cache

Caches

- Lorsque le processeur doit accéder à une donnée :



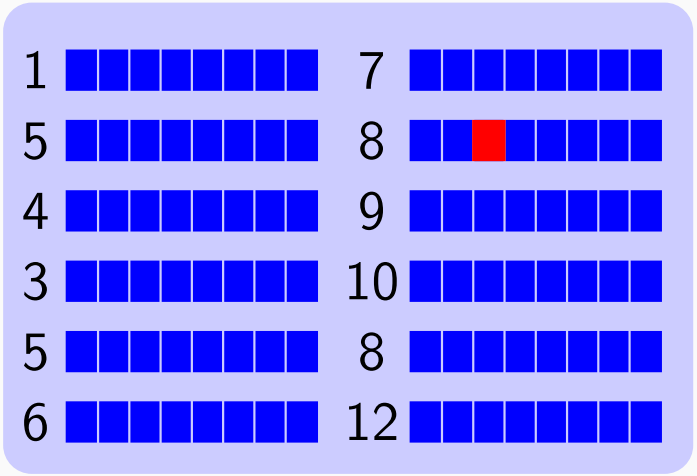
Mémoire centrale



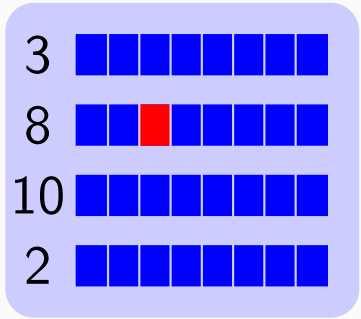
Cache

Caches

- Lorsque le processeur doit accéder à une donnée :
  - soit la donnée se trouve dans le cache (”*cache hit*”) : le transfert de la donnée vers les registres se réalise immédiatement.



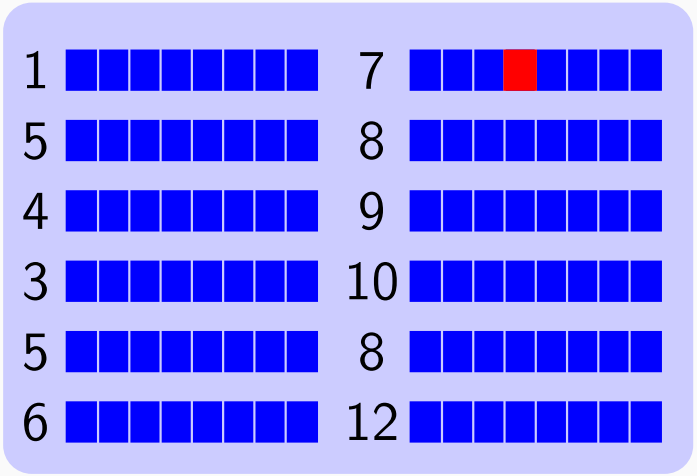
Mémoire centrale



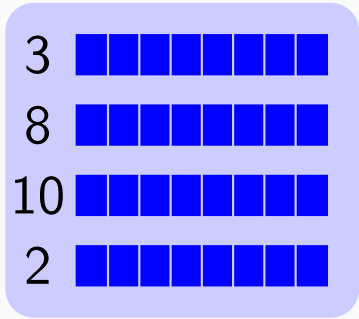
Cache

Caches

- Lorsque le processeur doit accéder à une donnée :
  - soit la donnée se trouve dans le cache (" *cache hit* ") : le transfert de la donnée vers les registres se réalise immédiatement.
  - soit la donnée ne se trouve pas dans le cache (" *cache miss* "): une nouvelle ligne est chargée dans le cache depuis la mémoire centrale.
    Il faut libérer de la place dans le cache et donc renvoyer en mémoire une des lignes, de préférence celle dont la durée d'inactivité est la plus longue.
    Le processeur est en attente pendant toute la durée de chargement de la ligne.



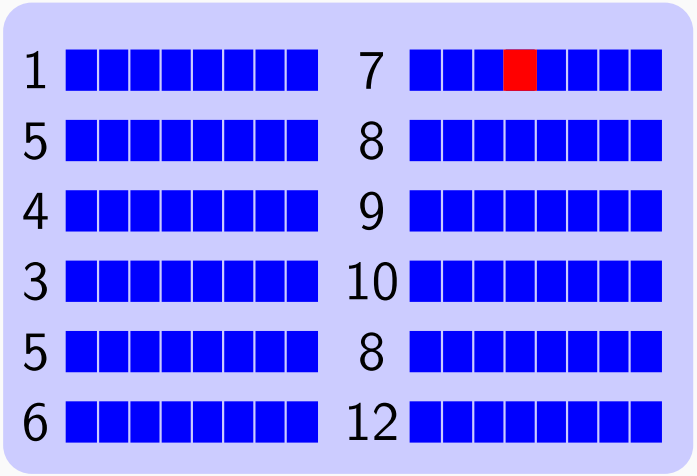
Mémoire centrale



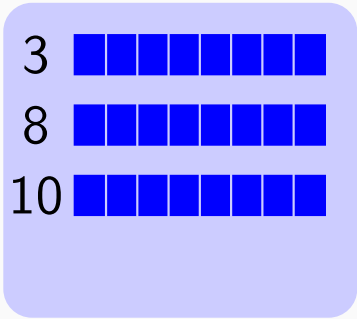
Cache

Caches

- Lorsque le processeur doit accéder à une donnée :
  - soit la donnée se trouve dans le cache (" *cache hit* ") : le transfert de la donnée vers les registres se réalise immédiatement.
  - soit la donnée ne se trouve pas dans le cache (" *cache miss* "): une nouvelle ligne est chargée dans le cache depuis la mémoire centrale.
    Il faut libérer de la place dans le cache et donc renvoyer en mémoire une des lignes, de préférence celle dont la durée d'inactivité est la plus longue.
    Le processeur est en attente pendant toute la durée de chargement de la ligne.



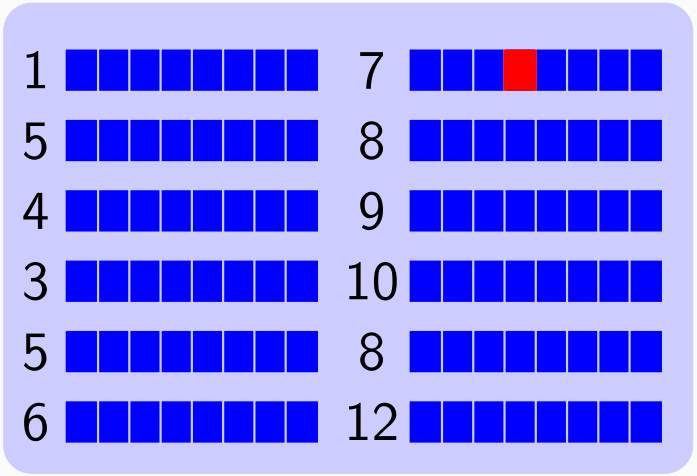
Mémoire centrale



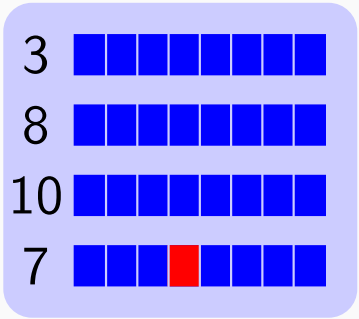
Cache

Caches

- Lorsque le processeur doit accéder à une donnée :
  - soit la donnée se trouve dans le cache (" *cache hit* ") : le transfert de la donnée vers les registres se réalise immédiatement.
  - soit la donnée ne se trouve pas dans le cache (" *cache miss* "): une nouvelle ligne est chargée dans le cache depuis la mémoire centrale.
    Il faut libérer de la place dans le cache et donc renvoyer en mémoire une des lignes, de préférence celle dont la durée d'inactivité est la plus longue.
    Le processeur est en attente pendant toute la durée de chargement de la ligne.



Mémoire centrale



Cache

Localité

- La localité consiste à améliorer le taux de succès du cache (éviter les "cache miss") en utilisant les principes de localités spatiale et temporelle.
- **Localité temporelle** : lorsqu'un programme accède à une donnée, il est intéressant d'y accèder à nouveau lors des instructions suivantes.
- **Localité spatiale** lorsqu'un programme accède à une donnée, il est intéressant d'accèder aux données voisines lors des instructions suivantes.
- Ces principes sont utilisés par :
  - les compilateurs ;
  - les programmeurs (*cf.* exemples suivants).

Localité : produit matrice-vecteur

- Les éléments d'un tableau C de dimension 2 sont rangés en mémoire de manière contigu en faisant varier d'abord le dernier indice.
- Par exemple, les éléments du tableau `a[2][3]` sont ordonnées de la façon suivante : `a[0][0]`, `a[0][1]`, `a[0][2]`, `a[1][0]`, `a[1][1]`, `a[1][2]`.

C: accès optimal

```
for (i=0; i<n; ++i)
{
  y[i] = 0.;
  for (j=0; j<n; ++j)
  {
      y[i] += a[i][j] * x[j];
  }
}
```

C: accès non optimal

```
for (i=0; i<n; ++i) y[i] = 0.;
for (j=0; j<n; ++j)
{
  for (i=0; i<n; ++i)
  {
      y[i] += a[i][j] * x[j];
  }
}
```

```
> icc -O0 mxv.c -o mxv
> ./mxv 20000
Matrix vector product (20000)
Elapsed time to compute matrix vector product (optimal) : 1.977292
Elapsed time to compute matrix vector product (not optimal) : 11.355297
```

Localité : produit matrice-vecteur

- Le loop tiling partitionne l'espace d'itération d'une boucle en morceaux ou blocs plus petits, de manière à garantir que les données utilisées dans la boucle restent dans la mémoire cache.

Exemple utilisant 2x2 blocs :

```
for (ib = 0; ib < n; ib += 2) {
  y[ib] = 0;
  y[ib + 1] = 0;
  for (jb = 0; jb < n; jb += 2) {
    for (i = ib; i < min(ib + 2, n); i++) {
      for (j = jb; j < min(jb + 2, n); j++) {
        y[i] += a[i][j] * x[j];
      }
    }
  }
}
```

- Difficile de déterminer le nombre de blocs permettant d'optimiser le cache.

Localité : produit matrice-vecteur

- La libraire mkl d'Intel est une version optimisée de la libraire BLAS (Basic Linear Algebra Subprograms).
- Elle permet d'obtenir les meilleures performances possibles pour des opérations d'algèbre linéaire sur les processeurs Intel notamment en optimisant les accès à la mémoire.

```
for (i=0; i<n; ++i)
{
  y[i] = 0.;
  for (j=0; j<n; ++j)
  {
      y[i] += a[i][j] * x[j];
  }
}

...

cblas_dgemv(CblasRowMajor, CblasNoTrans, n, n, 1., a, n, x, 1, 0., y, 1);
```

```
$ icc mxv.c -o mxv_mkl -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
$ ./mxv_mkl 80000
Matrix vector product (60000)
Elapsed time to compute matrix vector product : 4.131499
Elapsed time to compute matrix vector product with mkl : 2.549388
```

Exercise : product matrix vector

1. copy the directory `/mnt/beegfs/project/formation/memory` into your `$WORKDIR`.
2. load
3. compile the program using the Makefile
4. launch the job by using the batch script (`job.sh`)
5. compare the results

Floating-point numbers

Table of contents

Nombre à virgule flottante

- Les nombres à virgule flottante ou flottants permettent de représenter les nombres réels de manière approchée avec une quantité d'information fixe.
- Un flottant est caractérisé par :
  - son signe;
  - son exporsant;
  - sa mantisse.
- Exemple de flottants en base 10 :

$$0.02345 = +0.2345 \times 10^{-1}$$

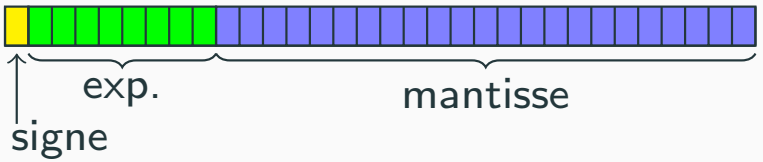
  - signe : $+$;
  - exposant : -1;
  - sa mantisse $= 2345$.

Norme IEEE754 - Nombre à virgule flottante en base 2

- La norme IEEE 754 de 1985 spécifie deux formats de nombres à virgule flottante (et deux formats étendus optionnels) en base 2

Norme IEEE754 - Nombre à virgule flottante en base 2

- La norme IEEE 754 de 1985 spécifie deux formats de nombres à virgule flottante (et deux formats étendus optionnels) en base 2
- Exemple : les flottants stockés sur 32 bits ("simple précision") :



  - le signe est codé sur 1 bit;
  - l'exposant est codé sur 8 bits;
  - la mantisse est codée sur 23 bits.
  - ils permettent de représenter les réels r tels que :

$$1.2 \times 10^{-38} \leq |r| \leq 3.4 \times 10^{+38}$$

avec 7 à 8 chiffres significatifs

Norme IEEE754 - Nombre à virgule flottante en base 2

- La norme IEEE 754 de 1985 spécifie deux formats de nombres à virgule flottante (et deux formats étendus optionnels) en base 2
- Exemple : les flottants stockés sur 64 bits ("double précision") :

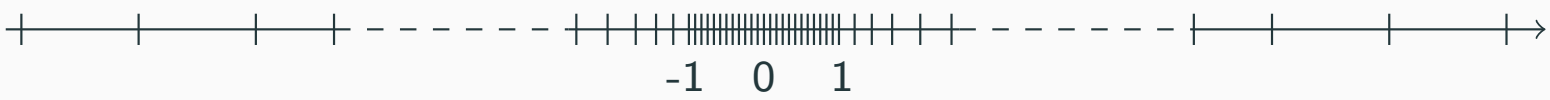
  - le signe est codé sur 1 bit;
  - l'exposant est codé sur 11 bits;
  - la mantisse est codée sur 52 bits.
  - Les flottants 64 bits permettent de représenter les réels r tels que :

$$2.2 \times 10^{-308} \leq |r| \leq 1.8 \times 10^{+308}$$

avec 15 chiffres significatifs

Quelques propriétés des nombres à virgule flottante

- L'ensemble des flottants décrivent seulement un sous-ensemble fini des nombres réels.
- Les éléments de cet ensemble ne sont pas équirépartis sur l'axe des rééls :



- La plupart des réels sont donc arrondis
- Ex: $0,1$ n'est pas représentable exactement avec un flottant sur 64 bits car 0.1 admet une représentation périodique en base 2 et nécessite une infinité de décimales pour avoir une représentation exacte.

Quelques propriétés des nombres à virgule flottante

- Conséquence : dans les ensembles de nombres flottants, l'addition n'est pas associative !
- Exemple en python :

```
a = 1000000.
b = 999999.5
c = 0.1

res1 = (a - b) + c
res2 = (a + c) - b

print(f"(a - b) + c = {res1}")
print(f"(a + c) - b = {res2}")
```

```
(a - b) + c = 0.6
(a + c) - b = 0.5999999999767169
```

- L'erreur d'arrondi de l'opération $1000000 + 0.1$ est plus important que celle de $0.5 + 0.1$ car la distance entre des flottants est plus grande autour de 1000000 que celle autour de 0.5.
- Pour limiter les erreurs d'arrondi, il est préférable de réaliser des opérations entre flottants de même ordre de grandeur.

Erreur d'arrondi

- On considère la somme :

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{10^6}$$

- puis la même somme évaluée dans l'ordre inverse :

$$\frac{1}{10^6} + \cdots + \frac{1}{3} + \frac{1}{2} + 1$$

```
s1 = 0.
for i in range(1,10_000_001): s1 += 1/i
print(f"s1 = {s1}")

s2 = 0.
for i in range(10_000_000, 0, -1): s2 += 1/i
print(f"s2 = {s2}")
```

Erreur d'arrondi

- On considère la somme :

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{10^6}$$

- puis la même somme évaluée dans l'ordre inverse :

$$\frac{1}{10^6} + \cdots + \frac{1}{3} + \frac{1}{2} + 1$$

```
s1 = 0.
for i in range(1,10_000_001): s1 += 1/i
print(f"s1 = {s1}")

s2 = 0.
for i in range(10_000_000, 0, -1): s2 += 1/i
print(f"s2 = {s2}")
```

```
s1 = 16.695311365857272
s2 = 16.695311365859965
```

Erreur d'arrondi

- On considère la somme :

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{10^6}$$

- puis la même somme évaluée dans l'ordre inverse :

$$\frac{1}{10^6} + \cdots + \frac{1}{3} + \frac{1}{2} + 1$$

```
s1 = 0.
for i in range(1,10_000_001): s1 += 1/i
print(f"s1 = {s1}")

s2 = 0.
for i in range(10_000_000, 0, -1): s2 += 1/i
print(f"s2 = {s2}")
```

```
s1 = 16.695311365857272
s2 = 16.695311365859965
```

- C'est le deuxième algorithme qui donne le résultat le plus précis car c'est celui qui permet de réaliser le plus d'opérations entre des nombres de même ordre de grandeur (notion de stabilité d'un algorithme).

Erreur d'arrondi

- On considère la suite

$$\begin{cases} u_0 = U_0 \\ u_{n+1} = au_n - b \end{cases}$$

- Lorsque $U_0 = 0.1$, $a = 11$ et $b = 1$, la suite constante u_n est une suite constante $u_n = 0.1$ pour laquelle le source python est :

```
x = 0.1
for i in range(50):
    x = 11*x - 1;
    print(f"Etape {i+1:2d} : x = {x}")
```

- Lorsque $U_0 = 0.25$, $a = 5$ et $b = 1$, la suite constante u_n est une suite constante $u_n = 0.25$ pour laquelle le source python est :

```
x = 0.25
for i in range(50):
    x = 5*x - 1;
    print(f"Etape {i+1:2d} : x = {x}")
```

Erreur d'arrondi

- Exécution pour $U_0 = 0.1$, $a = 11$ et $b = 1$ (sachant que 0.1 n'est pas représentable en virgule flottante en base 2 sur 64 bits) :

```
Etape  1 : x = 0.10000000000000009
Etape  2 : x = 0.10000000000000098
Etape  3 : x = 0.10000000000001075
Etape  4 : x = 0.10000000000011822
Etape  5 : x = 0.10000000000130038
Etape  6 : x = 0.1000000000143042
Etape  7 : x = 0.10000000015734622
Etape  8 : x = 0.10000000173080847
Etape  9 : x = 0.10000001903889322
Etape 10 : x = 0.10000020942782539
Etape 11 : x = 0.10000230370607932
Etape 12 : x = 0.10002534076687253
Etape 13 : x = 0.10027874843559781
Etape 14 : x = 0.1030662327915759
Etape 15 : x = 0.13372856070733485
Etape 16 : x = 0.4710141677806834
Etape 17 : x = 4.181155845587517
Etape 18 : x = 44.992714301462684
Etape 19 : x = 493.9198573160895
Etape 20 : x = 5432.118430476985
```

```
Etape 21 : x = 59752.302735246834
Etape 22 : x = 657274.3300877152
Etape 23 : x = 7230016.630964867
Etape 24 : x = 79530181.94061354
Etape 25 : x = 874832000.346749
Etape 26 : x = 9623152002.814238
Etape 27 : x = 105854672029.95662
Etape 28 : x = 1164401392328.5227
Etape 29 : x = 12808415315612.75
Etape 30 : x = 140892568471739.25
Etape 31 : x = 1549818253189130.8
Etape 32 : x = 1.7048000785080436e+16
Etape 33 : x = 1.875280086358848e+17
Etape 34 : x = 2.0628080949947328e+18
Etape 35 : x = 2.269088904494206e+19
Etape 36 : x = 2.4959977949436268e+20
Etape 37 : x = 2.745597574437989e+21
Etape 38 : x = 3.0201573318817883e+22
Etape 39 : x = 3.322173065069967e+23
Etape 40 : x = 3.6543903715769637e+24
```

- Problème mal conditionné : une petite perturbation de la donnée initiale implique une modification du résultat grande devant la taille de la perturbation.

Erreur d'arrondi

- Exécution pour $U_0 = 0.25$, $a = 5$ et $b = 1$ (sachant que 0.25 est représentable exactement par un nombre à virgule flottante en base 2 sur 64 bits) :

```
Etape  1 : x = 0.25
Etape  2 : x = 0.25
Etape  3 : x = 0.25
Etape  4 : x = 0.25
Etape  5 : x = 0.25
Etape  6 : x = 0.25
Etape  7 : x = 0.25
Etape  8 : x = 0.25
Etape  9 : x = 0.25
Etape 10 : x = 0.25
Etape 11 : x = 0.25
Etape 12 : x = 0.25
Etape 13 : x = 0.25
Etape 14 : x = 0.25
Etape 15 : x = 0.25
Etape 16 : x = 0.25
Etape 17 : x = 0.25
Etape 18 : x = 0.25
Etape 19 : x = 0.25
Etape 20 : x = 0.25
```

```
Etape 21 : x = 0.25
Etape 22 : x = 0.25
Etape 23 : x = 0.25
Etape 24 : x = 0.25
Etape 25 : x = 0.25
Etape 26 : x = 0.25
Etape 27 : x = 0.25
Etape 28 : x = 0.25
Etape 29 : x = 0.25
Etape 30 : x = 0.25
Etape 31 : x = 0.25
Etape 32 : x = 0.25
Etape 33 : x = 0.25
Etape 34 : x = 0.25
Etape 35 : x = 0.25
Etape 36 : x = 0.25
Etape 37 : x = 0.25
Etape 38 : x = 0.25
Etape 39 : x = 0.25
Etape 40 : x = 0.25
```

- Pas de perturbation de la donnée initiale, l'algorithme donne le résultat attendu.

Erreur d'arrondi

- On considère l'évaluation de la fonction : $f(x, y) = 9x^4 - y^4 + 2y^2$ pour $x = 40545$ et $y = 70226$ réprésentés par des nombres à virgule flottante sur 32 bits, sur 64 bits et sur 128 bits.

```
from mpmath import mp
def f(x,y):
    return 9*x**4 - y**4 + 2*y**2

# flottants sur 32 bits
mp.prec = 24
x = mp.mpf('40545')
y = mp.mpf('70226')
print(f"Evaluation de la fonction pour des flottants sur 32 bits  : f(x,y) = {f(x,y)}")
# flottants sur 64 bits
mp.prec = 53
x = mp.mpf('40545')
y = mp.mpf('70226')
print(f"Evaluation de la fonction pour des flottants sur 64 bits  : f(x,y) = {f(x,y)}")
# flottants sur 128 bits
mp.prec = 118
x = mp.mpf('40545')
y = mp.mpf('70226')
print(f"Evaluation de la fonction pour des flottants sur 128 bits : f(x,y) = {f(x,y)}")
```

```
Evaluation de la fonction pour des flottants sur 32 bits  : f(x,y) = 9.86338e+9
Evaluation de la fonction pour des flottants sur 64 bits  : f(x,y) = 1160.0
Evaluation de la fonction pour des flottants sur 128 bits : f(x,y) = 1.0
```

Erreur d'arrondi

Eléments d'explication :

- pour $x = 40545$, $9x^4 = 24321576859234655625$
- pour $y = 70226$, $y^4 = 24321576869098037776$
- la taille des mantisses des flottants sur 32 et 64 bits ne permet pas de réprésenter exactement ces entiers
- la source des erreurs étant la différence $9x^4 - y^4$, il faudrait obtenir ce terme en évitant de calculer $9x^4$ et y^4

Erreur d'arrondi

- On peut améliorer l'algorithme en factorisant $9x^4 - y^4$
- On calcule alors $f(x) = (3x^2 - y^2)(3x^2 + y^2) + 2y^2$
- Cet algorithme fonctionne pour les flottants sur 64 bits

```
def f(x,y):
    return (3*x**2 - y**2)*(3*x**2 + y**2) + 2*y**2

# flottants sur 32 bits
mp.prec = 24
x = mp.mpf('40545')
y = mp.mpf('70226')
print(f"Evaluation de la fonction pour des flottants sur 32 bits  : f(x,y) = {f(x,y)}")

# flottants sur 64 bits
mp.prec = 53
x = mp.mpf('40545')
y = mp.mpf('70226')
print(f"Evaluation de la fonction pour des flottants sur 64 bits  : f(x,y) = {f(x,y)}")
```

```
Evaluation de la fonction pour des flottants sur 32 bits  : f(x,y) = 5.05992e+12
Evaluation de la fonction pour des flottants sur 64 bits  : f(x,y) = 1.0
```

Références concernant les flottants

- *What every scientist should know about floating-point arithmetic*, David Goldberg, ACM Computing Surveys, Vol 23, No 1, March 1991.
- *Handbook of Floating-Point Arithmetic*, Muller et al., Birkhäuser Boston, 2010.

Exercise: float

1. create a conda environment
2. install module mpmath in this environment
3. test examples