Introduction to CINECA HPC infrastructure and Leonardo

January 30, 2026

Alberto Bocchinfuso
a.bocchinfuso@cineca.it

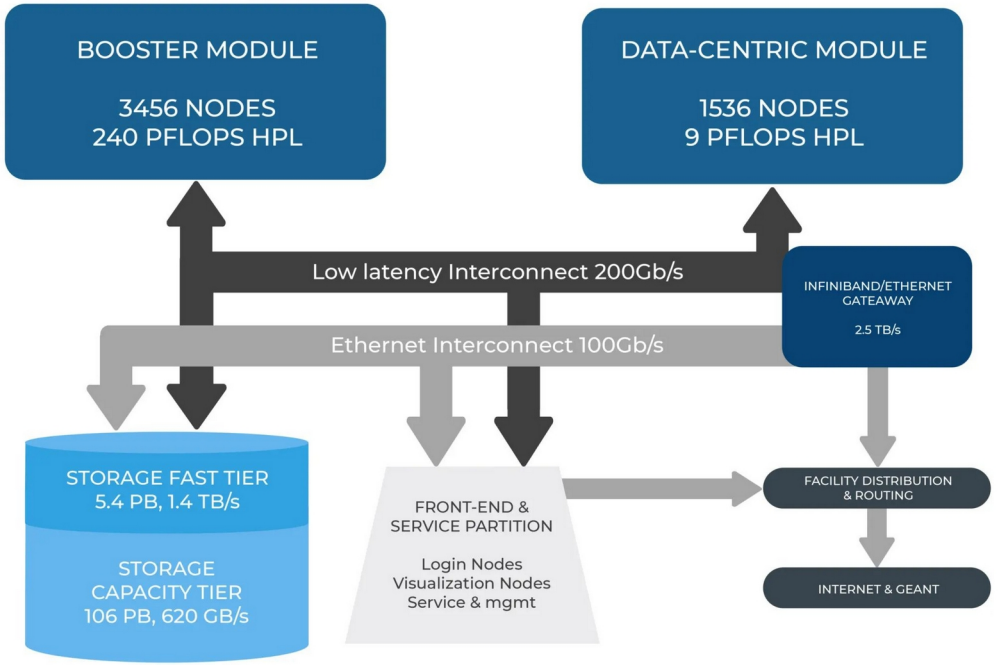
Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

Leonardo infrastructure



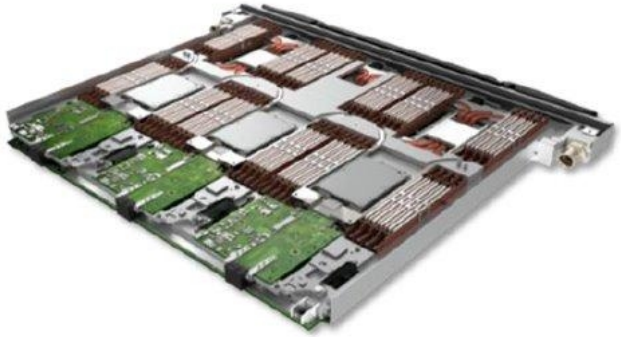
Booster
(GPU) partition

Atos BullSequana X2135 “Da Vinci” blade

- 3456 nodes: **lrdn[0001-3456]**
- Processors: **1 x CPU Intel Xeon 8358,**
  32 cores Intel Ice Lake, 2.6 GHz (one socket)
- Accelerators: **4 x NVidia custom Ampere GPU A100 SXM4 64 GB**
- RAM: 512 (8 x 64) GB DDR4 3200 MHz
- **Diskless**
- Internal network: NVLink 3.0 200 GB/s GPU-to-GPU, PCIe Gen4 GPU-to-CPU, each GPU has direct 100Gb/s connection to the InfiniBand network

Peak performance: about 309 PFlops

Data Centric and General Purpose (CPU-only) partition

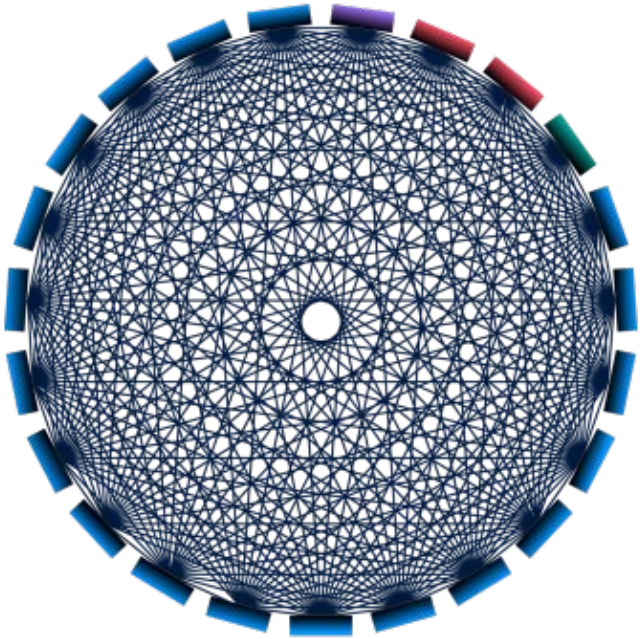


BullSequana X2140 three-node CPU Blade

- 1536 nodes: **lrdn[3457-4992]**
- Processors (dual-socket): **2x CPU Intel *Xeon* 8480+,**
  56 cores Intel *Sapphire Rapids* (**112 cores/node**),
  3.8 GHz (turbo enabled)
- RAM: 512 (16 x 32) GB DDR5 4800 MHz
- **Disk: 1x SSD 3.84 TB M.2 NVMe**
- Internal network: PCIe Gen5,
  1x port HDR100 100Gb/s network interface

Peak performance: about 13 PFlops

Inter-node network topology



Booster Module nodes
I/O cell
Data-Centric cells
Hybrid cell (Booster + Data-Centric nodes)

Dragonfly+ topology

based on **NVidia Mellanox Infiniband HDR, bidirectional bandwidth of 200 Gb/s**

- All nodes are divided into cells
- Non-blocking, two-layer Fat Tree within the cells
- All to all connection between cells

- **Adaptive routing algorithm**: SLURM will take care of the “best”-possible node allocations

Storage

Fast Tier

5.4 PB @ 1.4 TB/s

NVMe storage (SSD disks)

- **HOME, PUBLIC, FAST SCRATCH**

Capacity Tier

106 PB @ read 744 GB/s - write 620 GB/s

HDD disks

- **WORK, LARGE SCRATCH, DRES**

Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

Access a cluster

For this course, you will access Leonardo with temporary training usernames with **password**

$ ssh a08trXXX@login.leonardo.cineca.it

You should have received an email with you usename and password.

If you didn’t receive your credentials, please, ask the instructor.

Access a cluster

$ ssh <username>@login.leonardo.cineca.it

Message of the day

- ➔ Short system description
- ➔ System status
- ➔ “In evidence” messages
- ➔ “Important” messages

  (e.g. scheduled maintenances)

Filesystems

$HOME

- 50 GB per user
- user specific
- permanent
- daily backup (soon)

$PUBLIC

- 50 GB per user, only on Leonardo
- user specific (permissions **755**)
- permanent
- **no** backup

$SCRATCH

- no quota
- user specific
- temporary (data removed after 40 days)
- **no** backup

https://docs.hpc.cineca.it/hpc/hpc_data_storage.html

Filesystems

$HOME

- 50 GB per user
- user specific
- permanent
- daily backup (soon)

$PUBLIC

- 50 GB per user, only on Leonardo
- user specific (permissions **755**)
- permanent
- **no** backup

$SCRATCH

- no quota
- user specific
- temporary (data removed after 40 days)
- **no** backup

$WORK

- quota per account (default 1 TB)
- account specific
- permanent
- **no** backup

$FAST

- $WORK mirror
- **fast I/O**
- only on Leonardo

https://docs.hpc.cineca.it/hpc/hpc_data_storage.html

Filesystems

$HOME
- 50 GB per user
- user specific
- permanent
- daily backup (soon)

$PUBLIC
- 50 GB per user, only on Leonardo
- user specific (permissions **755**)
- permanent
- **no** backup

$SCRATCH
- no quota
- user specific
- temporary (data removed after 40 days)
- **no** backup

$WORK
- quota per account (default 1 TB)
- account specific
- permanent
- **no** backup

$FAST
- $WORK mirror
- **fast I/O**
- only on Leonardo

$TMPDIR
- local on nodes
- job specific

DRES
- long storage on demand
- shared among accounts and platforms (not Leonardo)

All the filesystems are based on **Lustre**

→ Check your areas, disk usage and quota: **$ cindata**

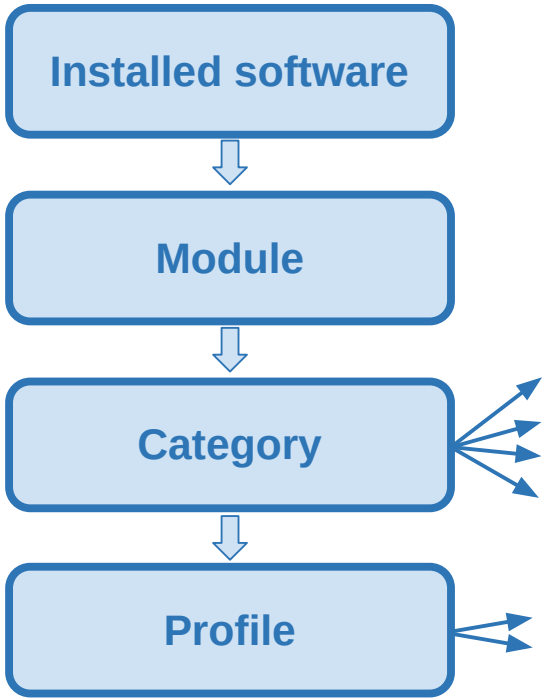
https://docs.hpc.cineca.it/hpc/hpc_data_storage.html

Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

Module environment

Any available software is offered on the clusters in a module environment.
The modules are organized in functional categories and collected in different profiles.



Compilers
Libraries
Tools
Applications

Base is the default profile:
automatically loaded after login,
containing basic modules
for programming activities

Programming (**base**, **advanced**): compilation, profiling, debugging…
Domain (**chem-phys, astro, bioinf…**): production activities

Module environment

$ module av

```
------------------------------------------------------ /leonardo/prod/opt/modulefiles/profiles ------------------------------------------------------
profile/archive  profile/base       profile/chem-phys  profile/geo-inquire  profile/meteo    profile/spoke7
profile/astro    profile/candidate  profile/deeplrn    profile/lifesc       profile/quantum  profile/statistics
```

```
---------------------------------------------- /leonardo/prod/opt/modulefiles/base/libraries ----------------------------------------------
adios/1.13.1--intel-oneapi-mpi--2021.10.0--oneapi--2023.2.0          metis/5.1.0--gcc--12.2.0
adios/1.13.1--openmpi--4.1.6--gcc--12.2.0-cuda-12.1                  metis/5.1.0--oneapi--2023.2.0
blitz/1.0.2--gcc--12.2.0                                             nccl/2.19.1-1--gcc--12.2.0-cuda-12.1
blitz/1.0.2--oneapi--2023.2.0                                        nccl/2.19.3-1--gcc--12.2.0-cuda-12.1
boost/1.83.0--gcc--12.2.0                                            netcdf-c/4.9.2--gcc--12.2.0
boost/1.83.0--intel-oneapi-mpi--2021.10.0--oneapi--2023.2.0-atomic   netcdf-c/4.9.2--intel-oneapi-mpi--2021.10.0--oneapi--2023.2.0
boost/1.83.0--oneapi--2023.2.0                                       netcdf-c/4.9.2--oneapi--2023.2.0
boost/1.83.0--openmpi--4.1.6--gcc--12.2.0                            netcdf-c/4.9.2--openmpi--4.1.6--gcc--12.2.0
boost/1.83.0--openmpi--4.1.6--nvhpc--23.11                           netcdf-c/4.9.2--openmpi--4.1.6--nvhpc--23.11
cfitsio/4.3.0--gcc--12.2.0                                           netcdf-fortran/4.6.1--gcc--12.2.0
```

```
------------------------------------------------ /leonardo/prod/opt/modulefiles/base/tools ------------------------------------------------
anaconda3/2023.09-0  jube/2.4.3  spack/0.21.0-68a
cintools/1.0         maven/3.8.4 spack/DCGP_0.21.0
```

```
---------------------------------------------- /leonardo/prod/opt/modulefiles/base/compilers ----------------------------------------------
cuda/12.1  gcc/12.2.0           intel-oneapi-compilers/2023.2.1     nvhpc/23.11  perl/5.36.0--gcc--8.5.0  python/3.10.8--gcc--8.5.0
cuda/12.3  gcc/12.2.0-cuda-12.1 llvm/14.0.6--gcc--12.2.0-cuda-12.1  nvhpc/24.3   perl/5.38.0--gcc--8.5.0  python/3.11.6--gcc--8.5.0
```

Almost all the modules on Leonardo have been installed with **Spack**, and they report the Spack package name.

Module environment

$ module load profile/astro
$ module av

Loaded profiles
are **added** to the environment

```
------------------------------------------------- /leonardo/prod/opt/modulefiles/profiles -------------------------------------------------
profile/archive  profile/base       profile/chem-phys  profile/geo-inquire  profile/meteo    profile/spoke7
profile/astro    profile/candidate  profile/deeplrn    profile/lifesc       profile/quantum  profile/statistics

------------------------------------------------ /leonardo/prod/opt/modulefiles/astro/libraries ------------------------------------------------
cfitsio/4.3.0--gcc--12.2.0
```

$ module show <module_name>/<version> → Print information about the module, such as dependencies, paths

$ module help <module_name>/<version> → Print the help of the software, its brief description and examples of the use

Module environment

$ modmap -m <module_name> → Detect all profiles, categories and modules available (e.g. different releases)

$ module load <profile>

$ module load <module_name>/<version> → **all the dependencies are automatically loaded; we recommend to specify the module version!**

$ module list → List all the profiles and modules loaded so far

Important!

You will find **modules compiled to support GPUs and modules suitable only for CPUs**.
You can check the compiler in the full name of the module, where the version is specified (e.g. gromacs/2022.3--intel-oneapi-mpi--2021.10.0--oneapi–2023.2.0). Remind that modules compiled with gcc, nvhpc, cuda should be used only on the Booster partition, while modules compiled with intel, intel-oneapi are suitable for running on the DGCP partition.

Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

Programming environment

Compilers and MPI libraries are available as modules in profile/base.
Use the ones suitable for the architecture!

Check with commands
modmap -m,
module av,
module show,
module help,
and **man**

Compilers

- **GCC** (GNU compilers: gcc, g++, gfortran)
- **NVHPC** (ex hpc-sdk, ex PGI + CUDA → NVIDIA compilers: nvc, nvc++, nvcc, nvfortran)
- **CUDA**
- **INTEL ONEAPI** (Intel compilers: icc, icpc, ifort…) → **no** Nvidia GPU support (Leonardo DCGP)

MPI libraries

- **OpenMPI** (GNU/NVHPC compilers)
- **Intel OneAPI MPI** (Intel compilers) → **no** CUDA-aware (Leonardo DCGP)

Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

Login and compute nodes

CINECA HPC clusters are shared among many users, so **a responsible use is crucial!**

Login nodes

- Interactive runs on login nodes are strongly discouraged and should be limited to short test runs
  → ***10 minutes cpu-time limit***
- Avoid running large and parallel applications on login nodes
- ***No GPUs on login nodes***

Compute nodes

- Long production jobs should be submitted on compute nodes using the **scheduler** → **SLURM**
- Jobs can be submitted in two main ways: via **batch mode** and via **interactive mode**
- **Nodes shared**, but the allocated resources (cores, gpus, memory) are assigned in an exclusive way

Submit jobs with SLURM

Batch mode

- Write a batch script
- Launch the batch script
  $ sbatch [options] start.sh
- The job is queued and scheduled

```
#!/bin/bash

#SLURM directives

variables environment

execution line
```

Submit jobs with SLURM

```
#!/bin/bash

#SBATCH --job-name=myjob
#SBATCH --output=job.out
#SBATCH --error=job.err
#SBATCH --time=00:30:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --mem=350GB
#SBATCH --partition=boost_usr_prod
#SBATCH --qos=boost_qos_dbg
#SBATCH --account=tra26_castiel2
#SBATCH --mail-type=ALL
#SBATCH --mail-user=user@email.com

module load <module_name>

srun ./myprogram
```

```
#!/bin/bash

#SLURM directives

variables environment

execution line
```

Submit jobs with SLURM

#SBATCH --job-name=myname or **-J myname**
Defines the name of your job (default=slurm-<JobID>)

#SBATCH --output=job.out or **-o job.out**
Specifies the file where the standard output is directed (default=slurm-<JobID>.out)

#SBATCH --error=job.err or **-e job.err**
Specifies the file where the standard error is directed (default=slurm-<JobID>.err)

#SBATCH --mail-type =ALL
Specifies mail notification. A mail will be sent to you when something happens to your job, according to the keywords you specified (NONE, BEGIN, END, FAIL, ALL)

#SBATCH --mail-user=user@email.com
Specifies the mail address for the keyword above

Submit jobs with SLURM

Specify the resources needed for the simulation:

#SBATCH --nodes=1 or **-N 1** number of compute nodes

#SBATCH --ntasks-per-node=4 number of MPI tasks per node

#SBATCH --cpus-per-task=8 number of cores per task

#SBATCH --gres=gpu:4 number of GPUs per node (max 4 on Leonardo Booster)

#SBATCH --mem=300000MB memory allocated for each node
(default=15400MB per task, max=494000 MB on Leonardo Booster)

#SBATCH --time=00:30:00 or **-t 00:30:00** maximum duration of the job
(the maximum time allowed depends on the partition/qos used)

The lower the resources or walltime you require the lower the time your job will wait before starting. Always require the amount of resources you really need, not to waste time queuing.

Submit jobs with SLURM (Booster)

#SBATCH --partition=boost_usr_prod or **-p boost_usr_prod**
Specifies the “partition”, a.k.a. the specific set of nodes among which your job can search for resources.

#SBATCH --qos=boost_qos_dbg or **-q boost_qos_dbg**
The Quality of Service is used to modify the limits of a partition and its priority, or to access selected partitions.

#SBATCH --reservation=<res_name>
Used to access to selected nodes of the partition reserved to particular accounts.
The reservation lets you bypass the regular queue and your job to start immediately.

Partition	QOS	#Cores/#GPU per job	Walltime	Max Nodes/cores/GPUs/user
boost_usr_prod	boost_qos_dbg	2 nodes	00:30:00	2 nodes / 64 cores / 8 GPUs

Submit jobs with SLURM (DCGP)

#SBATCH --partition=dcgp_usr_prod or **-p dcgp_usr_prod**
Specifies the “partition”, a.k.a. the specific set of nodes among which your job can search for resources.

#SBATCH --qos=dcgp_qos_dbg or **-q dcgp_qos_dbg**
The Quality of Service is used to modify the limits of a partition and its priority, or to access selected partitions.

#SBATCH --reservation=<res_name>
Used to access to selected nodes of the partition reserved to particular accounts.
The reservation lets you bypass the regular queue and your job to start immediately.

Partition	QOS	#Cores/#GPU per job	Walltime	Max Nodes/cores/GPUs/user
dcgp_usr_prod	dcgp_qos_dbg	2 nodes	00:30:00	2 nodes / 224 cores per user acc 512 nodes per prj. account

Submit jobs with SLURM

#SBATCH –account=tra26_castiel2 or **-A tra26_castiel2**

Specifies the account with a **budget** of core-hours available to run jobs.
WARNING: use account on **DCGP**

The computational hours are not assigned to users, but to projects and are shared among the users of the same project. Such projects are called **accounts** and are a different concept from your username.

You can check the status of all the accounts you are associated at, on the Booster/DCGP partition:

$ saldo -b

```
abocchin@login05:~
[abocchin@login05 ~]$ saldo -ba tra25_sycl
PI: nshukla1
COLLABORATORS: a08trc13 a08trc25 a08trc08 a08trc02 a08trc07 a08trc20 a08trc19 a08trc14 a08trc10 a08trc01 a08trc23 a08trc04 a08trc11 a08tr
c09 a08trc17 a08trc12 a08trc03 a08trc06 a08trc05 a08trc18 a08trc24 a08trc15 a08trc22 menayat1 a08trc16 msafari1 a08trc21
--------------------------------------------------------------------------------------------------------------------------------------
account               start         end         total        localCluster   totConsumed     totConsumed     monthTotal     monthConsumed
                                                (local h)   Consumed(local h)     (local h)               %      (local h)         (local h)
--------------------------------------------------------------------------------------------------------------------------------------
tra25_sycl          20250514    20250525          2000                  0             0             0.0          2727                 0
[abocchin@login05 ~]$
```

Accounting

Each node → max resources you can request per node, on Leonardo Booster

- 32 cores (cpus) → **ntasks-per-node * cpus-per-tasks ≤ 32**
- 4 GPUs
- 494000 MB (481 GB)
- 112 cores, no GPU and same memory → **DCGP**

⇨ The **accounting** considers

- the requested number of CPUs
- the requested number of GPUs
- the requested memory

and calculates the **number of equivalent cores** → it takes the **maximum** among

- N cpus
- N GPUs * 8 (= N GPUs * cores-per-node / GPUs-per-node)
- Memory / Memory-per-core (= Memory * cores-per-node / memory-per-node)

Submit jobs with SLURM

module load profile/<profile_name>
module load <module_name>/<version>
export …

srun ./myprogram
or
mpirun ./myprogram
Your parallel executable is launched on the compute nodes, with all the tasks requested via resource allocation.

In order to use mpirun, openmpi/intelmpi has to be loaded inside the job script:
module load openmpi/<version> (recommended on Leonardo Booster)
or
module load intelmpi/<version> (recommended on Leonardo DCGP)

Reminder!

Be sure to load the same version of the compiler that you used to compile your code!

Submit jobs with SLURM

Interactive mode

- Ask for the needed resources with the same **SLURM directives** with srun or salloc
- The job is queued and scheduled but, when executed, the standard input, output, and error streams are connected to the **terminal session** from which srun or salloc were launched
- **Run your application from that prompt**
- Exit from the terminal session: **$ exit**

Non MPI programs (one process using one or more GPUs)

$ srun -N 1 --ntasks-per-node=8 --cpus-per-task=4 --gres=gpu:4 -t 01:00:00 -p boost_usr_prod -q boost_qos_dbg -A tra26_castiel2 --pty /bin/bash

The session starts on the compute node: **[username@lrdn0053 ~]$**

Also **MPI programs** (using one or more GPUs)

$ salloc -N 1 --ntasks-per-node=8 --cpus-per-task=4 --gres=gpu:4 -t 01:00:00 -p boost_usr_prod -q boost_qos_dbg -A tra26_castiel2

A new session starts on the login node: **[username@login02 ~]$**

Monitor your jobs with SLURM

$ squeue -u <username> or **$ squeue --me**
Shows the list of all your scheduled jobs, along with their status (pending, running, closing, …)
Also, shows you the **job ID** required for other SLURM commands.

$ scontrol show job <job_id>
Provides a long list of informations for the job requested.
In particular, if your job isn't running yet, you'll be notified about the reason it has not started yet and, if it is scheduled with top **priority**, you will get an estimated start time.

$ scancel <job_id>
Removes the job (queued or running) from the scheduled job list by killing it.

$ sinfo (e.g. **$ sinfo -o "%10D %a %20F %P"**)
Provides information about SLURM nodes and partitions

$ sacct <options> <job_id> (e.g. **$ sacct -Bj <job_id>**)
Displays accounting data for all jobs and job steps in the SLURM job accounting log or Slurm database.

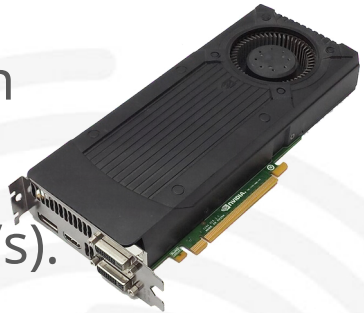
Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- BONUS – Introduction to GPUs
- Final remarks

GPU Architectures

This section of the slideshow is courtesy of Sergio Orlandini

Graphics Processing Unit aka GPU



Graphics Processing Unit (**GPU**) is a device equipped with

- **highly parallel microprocessor** (thousands of cores)
- private memory with very **high bandwidth** (~900 GB/s).

GPU highly parallel structure makes them more efficient than CPUs for embarrassingly parallel algorithms.

Born in '90 as a response to the growing demand for high definition **3D rendering** graphic applications (gaming, animations, etc)

The increasing popularity of 3D-accelerated games caused a rapid growth of computational capabilities of modern GPUs.

Parallel Intensive Computation

GPUs are designed to render complex 3D scenes composed of **millions of data** points/vertex at high frame rates (60-120 FPS)

The rendering process requires a set of transformations based on linear algebra operations and (mostly local) filters

- the same set of operations are applied on each data point of the scene
- each operation is independent of each other
- all operations are performed in parallel using a huge number of threads which process all data independently

Parallelism of Single Program Multiple Data (SPMD)

```
// typical loop over each point with the same set of operation
for each point in collection_of_points:
    output_data = trasformations_on_point(point, input_data)
```

- If the set of transformations can be applied **independently on each point**, the output result is independent on the order of point computations
- If transformations are independent, we can speed up the elaboration using **parallel work**:
    - apply the same transformation (Single Program) …
    - … to each point (Multiple Data)
    - … using multiple threads concurrently

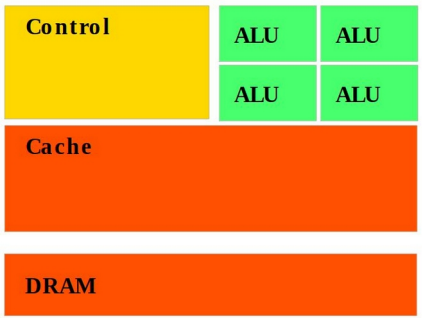
CPU vs GPU

A Central Processing Unit (**CPU**) is a latency-optimized general purpose processor designed to handle a wide range of tasks sequentially, while a Graphics Processing Unit (**GPU**) is a throughput-optimized specialized processor designed for high-end parallel computing.

- **Massive Parallel Computing**: extensive calculations with similar operations
- **High Data Throughput**: thousands of cores performing the same operation on multiple data items in parallel
- **High Computing Throughput**: high-performance computing power

CPU vs GPU

CPU	GPU
• **Low** compute density • Few **heavy**-weight cores 8 to 32 cores • Suitable for **Task** parallelism • Low latency • Large caches • Explicit thread management • Optimized for **serial** tasks	• **High** compute density • Many **light**-weight cores 5000+ cores • Suitable for **Data** parallelism • High throughput • High Memory Bandwidth • Threads are managed by hardware • Optimized for **parallel** tasks



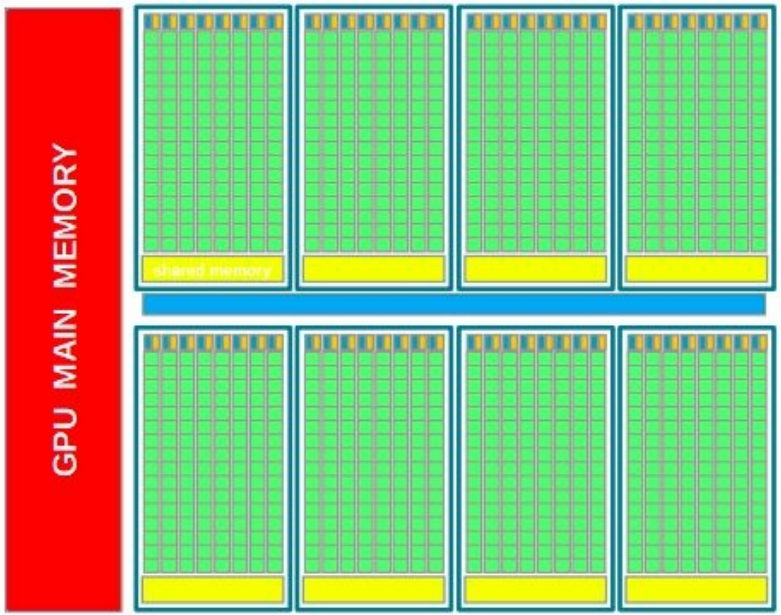
CPU

GPU

GPU Architecture Scheme

A typical GPU architecture consists of:

- **Main Global Memory**
  - Medium size (16-40 GB)
  - Very high bandwidth (800-1200 GB/s)
- **Streaming Processors (SM)**
  - Grouping independent cores and control units
  - Number of SM depends on GPU architecture
  - ~16-32 up to ~100 SM on modern GPUs
- Each SM unit has:
  - Many cores (> 100 cores)
  - Lots of registers (32K-64K)
  - Instruction scheduler dispatchers
  - Shared memory with fast access to data
  - Several caches



Host/Device Data Movements

- CPU and GPU are separated devices with **separate memory** space addresses.
- GPU is seen as an auxiliary **coprocessor**.
- Data must be moved from Host to Device memory in order to be processed on the GPU.
- When data has been processed on the GPU, they are transferred back to Host.
- The **data movement bottleneck**:
  - Data movement is often the bottleneck of many GPU applications.
  - The bus transfer can be quite slow with respect to the GPU throughput capacity.
  - Sometimes data transfer can take more than the data initialization/computation on GPU.

Outline

- CINECA HPC infrastructure → Leonardo Booster and DCGP
- Access HPC resources and filesystems
- Software environment
- Programming environment
- Production environment → Submit and monitor your jobs with SLURM
- Final remarks

Final remarks

- ★ **2FA method** is mandatory on CINECA HPC systems (not for training usernames).

- ★ **Login nodes** should only be used for installation (connection to external network!), compilation, and small tests. **No** GPUs on login nodes!

- ★ Consider to use **Leonardo Booster for your applications on GPUs** and **Leonardo DCGP for applications only on CPUs.**

- ★ Recommended **compilers** are gcc and Nvidia compilers (CUDA, nvhpc) for Leonardo Booster, and gcc and Intel for Leonardo DCGP.
  Check the **options** required to enable OpenACC/OpenMP parallelization, GPU support…

- ★ Rely on the already available **software stack**, tested and optimized for the cluster architecture, and on **Spack** for autonomously installing additional software.

https://docs.hpc.cineca.it/

Write to superc@cineca.it in case of need!