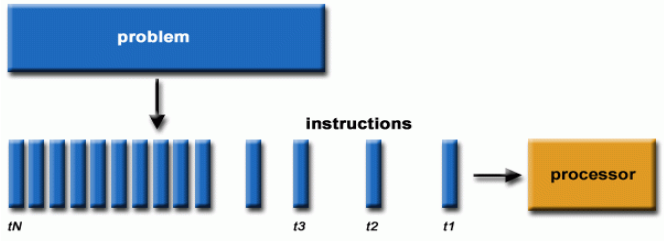
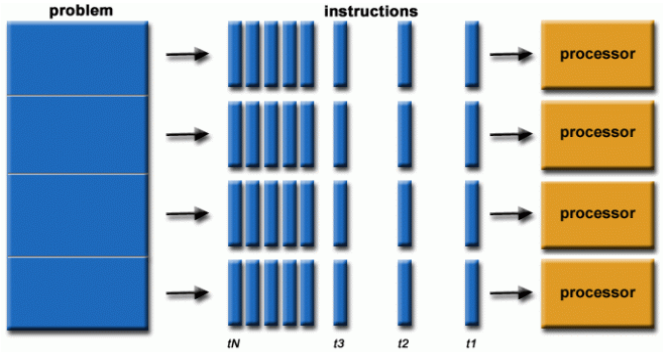
Introductory course on using the Supek supercomputer

Kristijan Dekanić (SRCE)

Parallelization

- **Program** – a set of commands that are executed on the processor
- **Parallelization** – dividing a problem into a series of smaller ones, which can be performed simultaneously and independently
- **Advantages** of parallelization
  - Multiple acceleration
  - Greater efficiency
- **Challenges** of parallelization
  - The complexity of app development
  - Limitations of parallelization – Amdahl's law



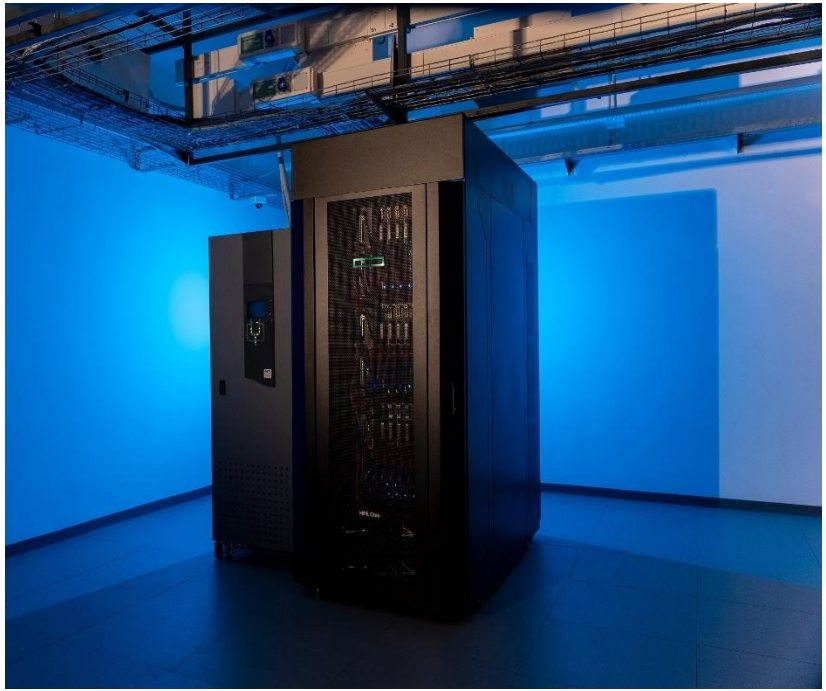


Cluster

- A set of computers (servers, nodes) with fast local connection between that act as a single resource
- Computationally demanding research
- Highly scalable applications with high performance and requirements

Supercomputer (Cluster) Supek

- **8384** processor cores
- **81** graphics processor
- **32 TB** RAM
- **580 TB** fast storage
- **Cray Slingshot** fast local network



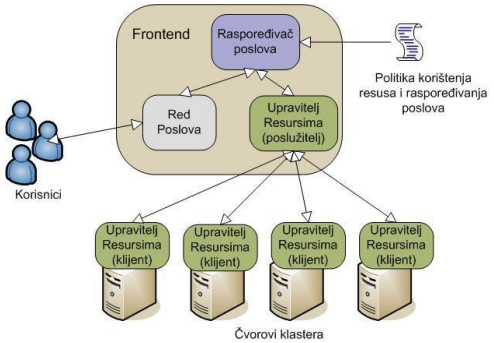
Job scheduler

Job

- Running programs on worker nodes
- Batch jobs do not require user interaction throughout the runtime
- Interactive tasks require user interaction at some point in the execution (e.g. GUI, selection of options, confirmation of actions...)
- Serial jobs are performed on a single processor core
- Parallel jobs are performed on multiple processor cores

Job scheduler

- Distributes jobs to worker nodes
- Records detailed information about job performance
- Monitors load



Job overview
qstat

Queue overview

```
qstat
```

Overview of my jobs

```
qstat -u $USER
```

Job information

```
qstat -fwx <job id>
```

```
Job id                  Name             User              Time Use S Queue
----------------------  ---------------- ----------------  -------- - -----
120513[].x3000c0s25b0* run-14D-66-15.s* user1                    0 B cpu-single
124590.x3000c0s25b0n0   singlegpu_run.p* user2                    0 Q gpu
125639.x3000c0s25b0n0   zh8243_7         user3             2178:51* R cpu
125729.x3000c0s25b0n0   bp_8000-8500     user4             612:36:* R cpu
125808.x3000c0s25b0n0   be_pea           user3             1243:25* R gpu
```

The most common job statuses	
Q	Job is in the Queue
R	Job is Running

The most important information about jobs	
`Job_Name`	Job name
`resources_used.mem`	Memory peak
`resources_used.ncpus`	Amount of processor cores requested
`resources_used.walltime`	Job duration
`job_state`	Job status
`Queue`	The queue in which the job was submitted
`exec_vnode`	The working node on which the job was submitted

Job description

- Jobs are prepared on access nodes, and "described" within the PBS script
  - A PBS script is essentially a shell script, consisting of a series of commands
  - If the interpreter is not specified, bash is given; #!/bin/bash
  - The specificity of the PBS script is the header within which we define the job options, using the key expression #PBS at the beginning of the line)

```
#PBS -q cpu-radionica
#PBS -l select=1:ncpus=16:mem=10gb
#PBS -j oe
#PBS -o output.log

cd ${PBS_O_WORKDIR}

module load "scientific/gromacs/2026.1/gcc-mpich"

gmx mdrun -deffnm nvt
```

Job description

Option	Example argument	Meaning
`-N`	`homo_volans`	Specifying the job name "homo_volans"
`-q`	`gpu`	Specifying the "gpu" job queue
`-l`	`select=2:ngpus=1:ncpus=4:mem=8gb`	Specifying 2 *chunks*, each with 1 GPU, 4 CPU cores, and 8 GiB memory
`-l`	`place=pack`	Placing all chunks on the same node
`-o`	`output.log`	Defining the name/path in which the standard output is saved
`-e`	`error.log`	Defining the name/path in which the standard error is saved
`-j`	`oe`	Linking the standard output and error to the same output file (leaves the error blank)

PBS environment variables

Variable	Description
`PBS_O_WORKDIR`	The directory where the job was submitted, that is, where the qsub command was called
`NCPUS`	Number of requested CPU cores (corresponds to the value from the ncpus option from the PBS script header)
`OMP_NUM_THREADS`	An OpenMP variable that PBS exports to the environment, which is equal to the value of the ncpus option from the header of the PBS script.

Modulefiles

- Setting up an environment for an application or library is achieved by loading modules
- Modules need to be activated every time a job is submitted

A list of all available modules

```
module avail
```

Loading modules

```
module load <ime modula>
```

Search modules

```
module spider <pojam>
```

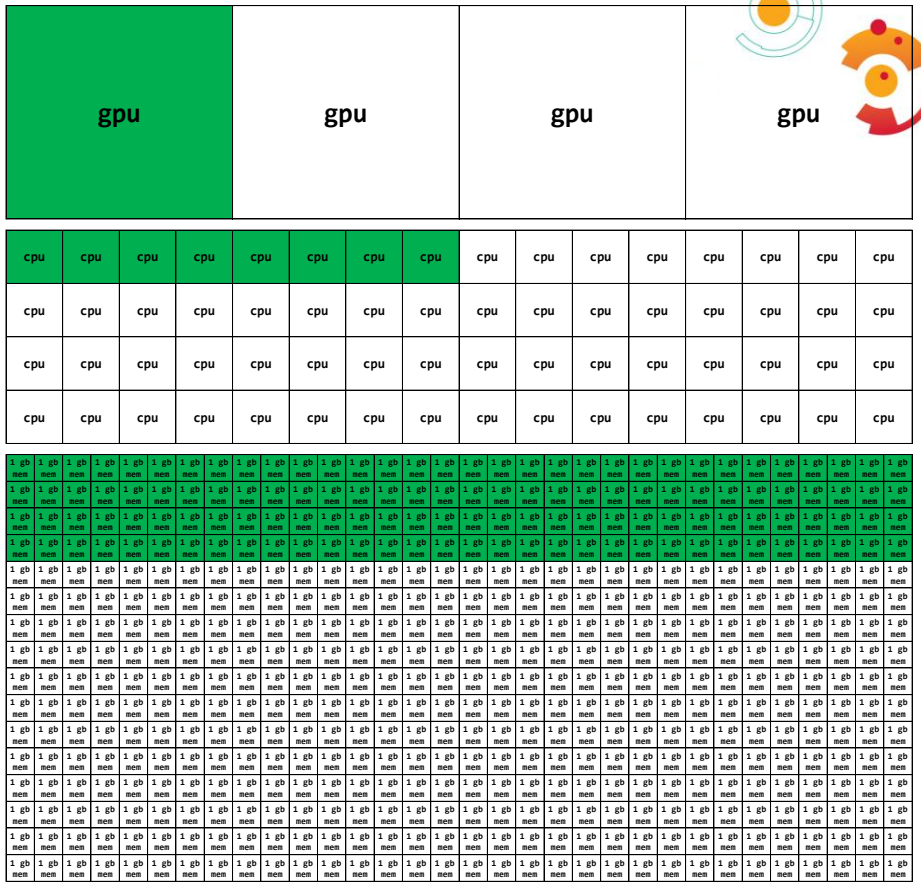
Cleaning (resetting) the environment

```
module purge
```

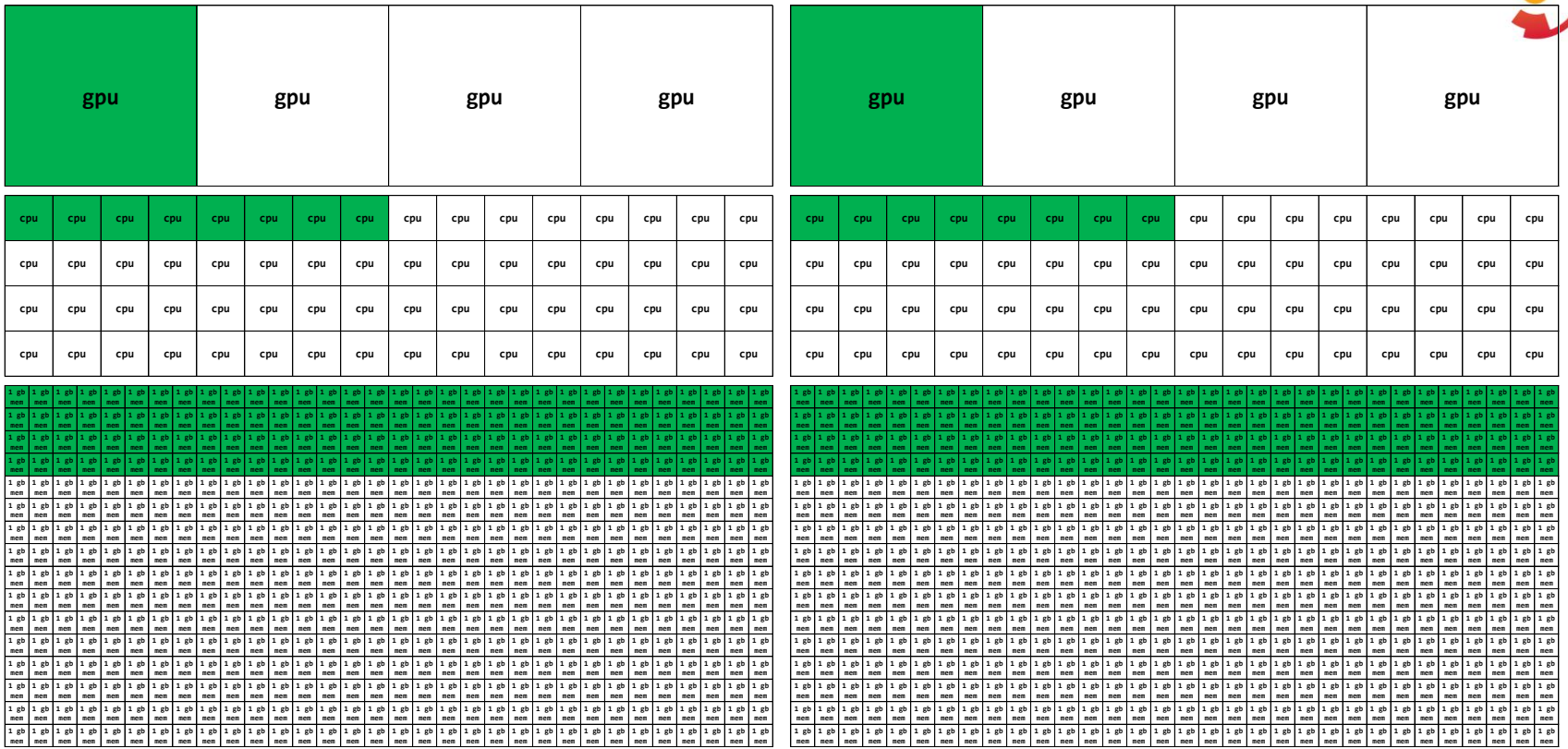
Chunk

- A **chunk** (piece of node) is PBS's abstraction of a set of resources (hardware) that are allocated to a job
  - It is defined by the **select** option
- For each chunk, the required resources are defined in more detail:
  - number of MPI processes, **mpiprocs**
  - number of processor cores, **ncpus**
  - number of graphics processors, **ngpus**
  - the amount of memory, **mem**
- A user can request multiple chunks
- With the place option, the user can choose whether he wants all the chunks
  - place it on the same node (**pack**)
  - scatter them on different nodes (**scatter**)
  - leave to pbs system for distribution (**free**)

```
#PBS -l select=1:ngpus=1:ncpus=8:mem=128gb
```



```
#PBS -l select=2:ngpus=1:ncpus=8:mem=128gb
#PBS -l place=scatter
```



Control of the resources used

- **cgroups** (control groups) are used to control the use of CPU cores and memory.
  - Jobs are limited to the requested amount of CPU cores
  - Jobs are terminated if they try to consume more memory than the amount requested in the job description

```
Cgroup mem limit exceeded: oom-kill:...
```

- Tips
  - Always try to define the amount of memory, otherwise the default memory value will take over
  - Get a "feel" for the memory consumption of your applications – insight into the memory peak of performed jobs is provided by **qstat -fwx** data **resources_used.mem**

Submitting and terminating jobs
qsub & qdel

Submitting a script

```
qsub <pbs script>
```

Terminating a job

```
qdel <job id>
```

Submitting interactive jobs
qsub -I

- Avoid performing interactive jobs on access nodes
  - Applications can be demanding, and often have no way to directly limit the resources used
    - Interactive submission "containerizes" the application within the requested resources (provided by cgroups)
- After submitting an interactive job, PBS assigns you to a work node and a new SSH session opens

Submission of an interactive job

```
qsub -I -q <queue> -l <resource list>
```

Requests an interactive PBS session

Generic examples

~/ccd2026/supek/generic/queue.sh

```
#PBS -q cpu-radionica
#PBS -l select=1:ncpus=1

echo "This is an example with the job queue."
```

- The outputs of PBS scripts (.o and .e files) are stored in the directory where the work is submitted, that is, **PBS_O_WORKDIR**, and are named by the name of the script

Generic examples

~/ccd2026/supek/generic/name.sh

```
#PBS -q cpu-radionica
#PBS -l select=1:ncpus=1
#PBS -N tutorial

echo "This is an example with the job name."
```

- When the job name is defined, the output PBS scripts (.o and .e files) take on its name

Generic examples

~/ccd2026/supek/generic/outputs.sh

```
#PBS -q cpu-radionica
#PBS -l select=1:ncpus=1
#PBS -o out.log
#PBS -e err.log

cd $PBS_O_WORKDIR

cat text.txt
cat another-text.txt
```

- It is possible to define the names of the standard output and the standard error separately
- The script assumes that the starting directory (the relative directory from which all paths within the script are viewed) is the user's HOME directory, and therefore it is recommended to manually move it to PBS_O_WORKDIR

Generic examples

~/ccd2026/supek/generic/join.sh

```
#PBS -q cpu-radionica
#PBS -l select=1:ncpus=1
#PBS -j oe
#PBS -o joined_output.log

cd $PBS_O_WORKDIR

cat text.txt
cat another-text.txt
```

- It is possible to join the standard output and standard error (.o and .e files) into the same file

OpenMP parallelization

- Applications that cannot spread their threads outside of a single computer node
- Such applications work with shared memory, since all threads share physical (RAM) memory
- Regardless of the PBS system, the number of processor cores on which such an application will be placed is determined by the OpenMP variable **OMP_NUM_THREADS**
  - PBS exports OMP_NUM_THREADS variable to the environment according to the **ncpus** option argument

Submitting OpenMP jobs

```
#PBS -l select=1:ncpus=16:mem=10gb
```

- 1 chunk (**select=1**) contains:
  - 16 processor cores (**ncpus=16**)
  - 10 GB of RAM (**mem=10GB**)

- For OpenMP applications, it is desirable that **each OpenMP thread has its own CPU core**

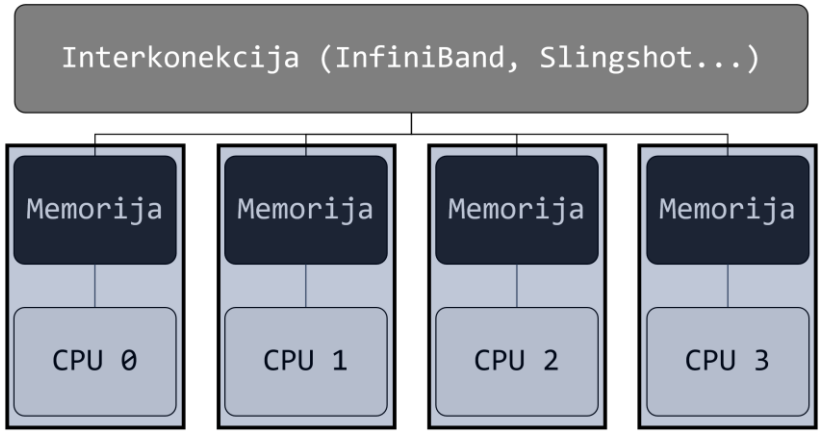
OpenMP job example

~/ccd2026/supek/openmp/run.sh

```
#PBS -q cpu-radionica
#PBS -l select=1:ncpus=16:mem=10gb
#PBS -j oe
#PBS -o output.log

cd ${PBS_O_WORKDIR}

module load "scientific/gromacs/2026.1/gcc-mpich"

gmx mdrun -deffnm nvt
```

MPI parallelization

- Applications that can extend their processes beyond a single computing node
- We say that such applications work with distributed memory, since each process can get its share of physical (RAM) memory
- The advantage is the ability to take up a larger amount of resources – each MPI process can be executed on its own computer node and have access to resources there
- **mpiexec** (or **mpirun**) is a startup command that runs multiple instances (processes) of a program



MPI nodefile (hostfile, machinefile)

- The combination of the values of the select, mpiprocs and place options defines the content of the PBS nodefile (hostfile), which directly determines the execution of MPI applications
- For each line to the PBS_NODEFILE, mpiexec will run one process on that node

```
#PBS -l select=4:mpiprocs=2
#PBS -l place=scatter
```

$PBS_NODEFILE	
node001	*chunk #1*
node001	
node002	*chunk #2*
node002	
node003	*chunk #3*
node003	
node004	*chunk #4*
node004	

```
#PBS -l select=8[:mpiprocs=1]
#PBS -l place=pack
```

$PBS_NODEFILE	
node001	*chunk #1*
node001	*chunk #2*
node001	*chunk #3*
node001	*chunk #4*
node001	*chunk #5*
node001	*chunk #6*
node001	*chunk #7*
node001	*chunk #8*

Submitting MPI jobs

```
#PBS -l select=16:mem=600mb
```

- 16 chunks (**select=16**), and each contains:
  - 1 MPI process (**mpiprocs=1**), **implicitly defined**
  - 1 processor core (**ncpus=1**), **implicitly defined**
  - 600 MB of RAM (**mem=600MB**)

- For MPI applications, it is desirable that each MPI process should have a single CPU core. That is why it is necessary that the values of select×ncpus and select×mpiprocs are the same!

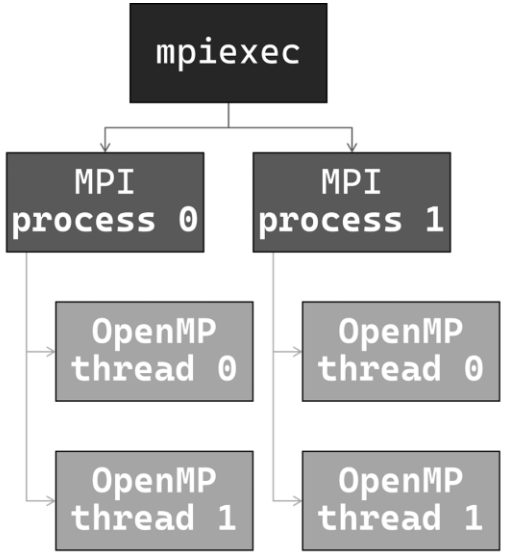
MPI job example

~/ccd2026/supek/mpi/run.sh

```
#PBS -q cpu-radionica
#PBS -l select=16:mem=600mb
#PBS -l place=pack
#PBS -j oe
#PBS -o output.log

cd ${PBS_O_WORKDIR}

module load "scientific/gromacs/2026.1/gcc-mpich"

mpiexec gmx mdrun -deffnm nvt
```

Hybrid (MPI + OpenMP) parallelization

- Applications that use hybrid parallelization can incorporate both types of parallelization (MPI and OpenMP) into their execution
- An application can run *n* MPI processes, with each of these copies being further distributed to OpenMP threads
- It goes without saying that all OpenMP threads (derived from the same MPI process) must be located on the same node, since they share physical (RAM) memory



Submitting hybrid jobs

```
#PBS -l select=2:ncpus=8:mem=5gb
```

- 2 chunks (**select=2**, and each contains:
  - 1 MPI process (**mpiprocs=1**), **implicitly defined**
  - 8 processor cores (**ncpus=8**)
  - 5 GB of RAM (**mem=5gb**)

- For hybrid applications, it is desirable that each OpenMP thread has its own CPU core.

Hybrid job example

~/ccd2026/supek/mpi/run.sh

```
#PBS -q cpu-radionica
#PBS -l select=2:ncpus=8:mem=5gb
#PBS -l place=pack
#PBS -j oe
#PBS -o output.log

cd ${PBS_O_WORKDIR}

module load "scientific/gromacs/2026.1/gcc-mpich"

mpiexec -d ${OMP_NUM_THREADS} --cpu-bind depth gmx mdrun -deffnm nvt
```

Use of GPUs

- In order to gain access to GPUs, it is necessary to define the GPU-having job queue and the number of GPUs with the **ngpus** option
- By defining the ngpus option, a variable **CUDA_VISIBLE_DEVICES** will be exported to the environment that will make the assigned GPU visible to the application
- It is also possible to independently search for processor (CPU) cores

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-6b3045f7-b364-8aea-ff5a-273090d230c6)
GPU 1: NVIDIA A100-SXM4-40GB (UUID: GPU-9f6e7d60-467d-3205-257c-995d93dea7fd)
GPU 2: NVIDIA A100-SXM4-40GB (UUID: GPU-32f785ef-c3b4-c02f-4953-9b0252cc1969)
GPU 3: NVIDIA A100-SXM4-40GB (UUID: GPU-f2cf071a-928d-27eb-9f3d-9d20591d8e76)

+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.60.13    Driver Version: 525.60.13    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  On   | 00000000:03:00.0 Off |                    0 |
| N/A   50C    P0   276W / 400W |  28755MiB / 40960MiB |     90%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM...  On   | 00000000:41:00.0 Off |                    0 |
| N/A   49C    P0   287W / 400W |  16251MiB / 40960MiB |     91%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   38C    P0    51W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM...  On   | 00000000:C1:00.0 Off |                    0 |
| N/A   47C    P0   254W / 400W |  16251MiB / 40960MiB |     92%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    954068      C   /usr/bin/python3                28752MiB |
|    1   N/A  N/A    966601      C   /usr/local/bin/python3.10       16248MiB |
|    3   N/A  N/A    966602      C   /usr/local/bin/python3.10       16248MiB |
+-----------------------------------------------------------------------------+
```

Submitting single-GPU jobs

```
#PBS -l select=1:ngpus=1:ncpus=16:mem=10gb
```

- 1 chunk (**select=1**), and contains:
  - 1 graphics processor (**ngpus=1**)
  - 16 processor cores (**ncpus=16**)
  - 10 GB RAM (**mem=10gb**)

Example of a single-GPU job

~/ccd2026/supek/gpu/run.sh

```
#PBS -q gpu-radionica
#PBS -l select=1:ngpus=1:ncpus=16:mem=10gb
#PBS -j oe
#PBS -o output.log

cd ${PBS_O_WORKDIR}

module load "scientific/gromacs/2026.1/gcc-mpich-cuda"

gpu-gmx-mdrun -deffnm nvt
```

Thank you for your attention

computing@srce.hr