VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

ACCESSING AND USING IT4I CLUSTERS | KAROLINA

Ondrej Meca

IT4Innovations

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

IT4I CLUSTERS

TOP500 CERTIFICATE

The List.

Karolina, GPU partition - Apollo 6500, AMD EPYC 7452 32C 2.35GHz, NVIDIA A100 SXM4 40 GB, Infiniband HDR200

IT4Innovations National Supercomputing Center, VSB-Technical University of Ostrava, Czechia

is ranked

No. 69

among the World's TOP500 Supercomputers

with 6.05 PFlop/s Linpack Performance

in the 57th TOP500 List published at the ISC Virtual 2021

Conference on June 28, 2021.

Congratulations from the TOP500 Editors

Erich Strohmaier, NERSC/Berkeley Lab
Jack Dongarra, University of Tennessee
Horst Simon, NERSC/Berkeley Lab
Martin Meuer, Prometeus

The GREEN 500 CERTIFICATE

Karolina, GPU partition - Apollo 6500, AMD EPYC 7452 32C 2.35GHz, NVIDIA A100 SXM4 40 GB, Infiniband HDR200

IT4Innovations National Supercomputing Center, VSB-Technical University of Ostrava, Czechia

is ranked

No. 15

among the World's TOP500 Supercomputers

with 20.346 GFlops/watts Performance

in the Green500 List published at the ISC Virtual 2021

Conference on June 28, 2021.

Congratulations from the Green500 Editors

Wu-chun Feng, Virginia Tech
Kirk Cameron, Virginia Tech

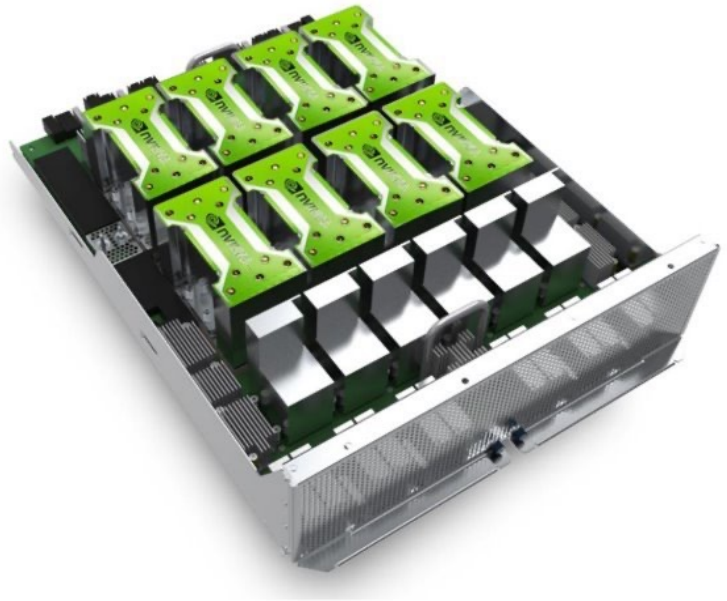
KAROLINA CLUSTER

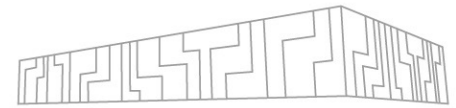
Universal partition: 720 compute nodes

- **2x 64-core** AMD EPYC 7H12 @ 2.6 GHz
- **256 GB** of memory
- 346 GB/s memory bandwidth, 5.3 Tflop/s per node
- 3.8 Pflop/s peak total
- 100 Gb/s NIC (infiniband HDR100)

GPU-accelerated partition: 72 compute nodes

- **2x 64-core** AMD EPYC 7763 @ 2.45 GHz
- **1024 GB** of memory
- **8x NVIDIA A100** SXM4 40GB
- 12.4 TB/s memory bandwidth, 156 Tflop/s per node
- Total 11.1 Pflop/s peak
- 4x 200 Gb/s NIC





How to connect to Karolina

ACCESSING KAROLINA

SSH client

Command line interface
- connect via ssh protocol

SSH server

Connect from your computer to Karolina
- SSH server on Karolina
- SSH client on your computer
- like remote desktop, but command-line interface only

SSH
- connect and do work

SCP
- copy files between Karolina and your computer

ACCESSING KAROLINA

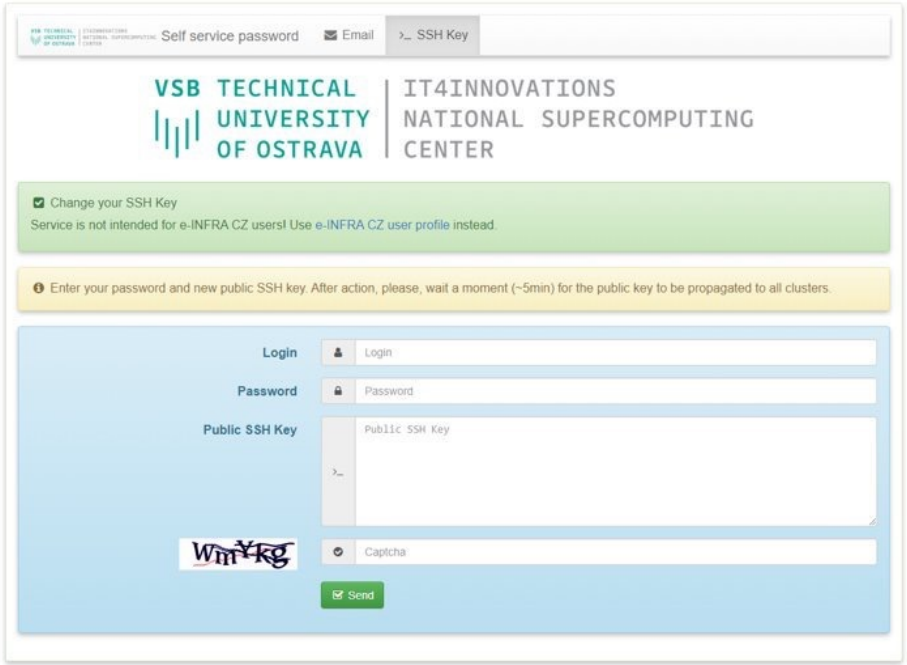
SSH keys for authentication
- Private-public key pair
- Password auth. is disabled on Karolina

Examine the .ssh directory
- **/home/<username>/.ssh**
- **C:/Users/<username>/.ssh**
- Create the directory if it does not exist

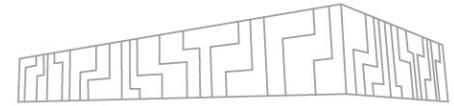
Are there **id_rsa** and **id_rsa.pub** files?
- This is the private and public key

No there aren't / Yes there are, but I want to generate new keys
- Open command line / terminal / powershell
- Run ssh-keygen
- Follow the instructions

1. Upload your public ssh key
2. https://extranet.it4i.cz/ssp
3. Choose SSH Key option in the top menu
4. Use the login and password you received
5. Paste the contents of your public ssh key
   - `~/.ssh/id_rsa.pub`



ACCESSING KAROLINA



DO NOT CTRL+C, CTRL-V

SOME CHARACTERS CAN BE INCORRECTLY COPIED

Command line:

- all Linux systems (incl. MacOS)
- newer Windows versions

Connect

- `$ ssh -i ~/.ssh/id_rsa username@karolina.it4i.cz`

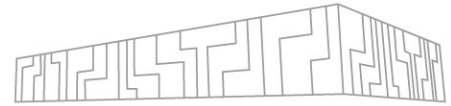
Copy

- `$ scp -i ~/.ssh/id_rsa local_file username@karolina.it4i.cz:path/on/karolina`

PuTTY, WinSCP

- SSH and SCP clients for Windows
- https://docs.it4i.cz/general/accessing-the-clusters/shell-access-and-data-transfer/putty/

ACCESSING KAROLINA



DO NOT CTRL+C, CTRL-V

SOME CHARACTERS CAN BE INCORRECTLY COPIED

Command line:

- all Linux systems (incl. MacOS)
- newer Windows versions

Connect

- `$ ssh karolina`

Copy

- `$ scp local_file karolina:path/on/karolina`

`~/.ssh/config`

```
host karolina
    HostName karolina.it4i.cz
    IdentityFile ~/.ssh/id_rsa
    User username
```

PuTTY, WinSCP

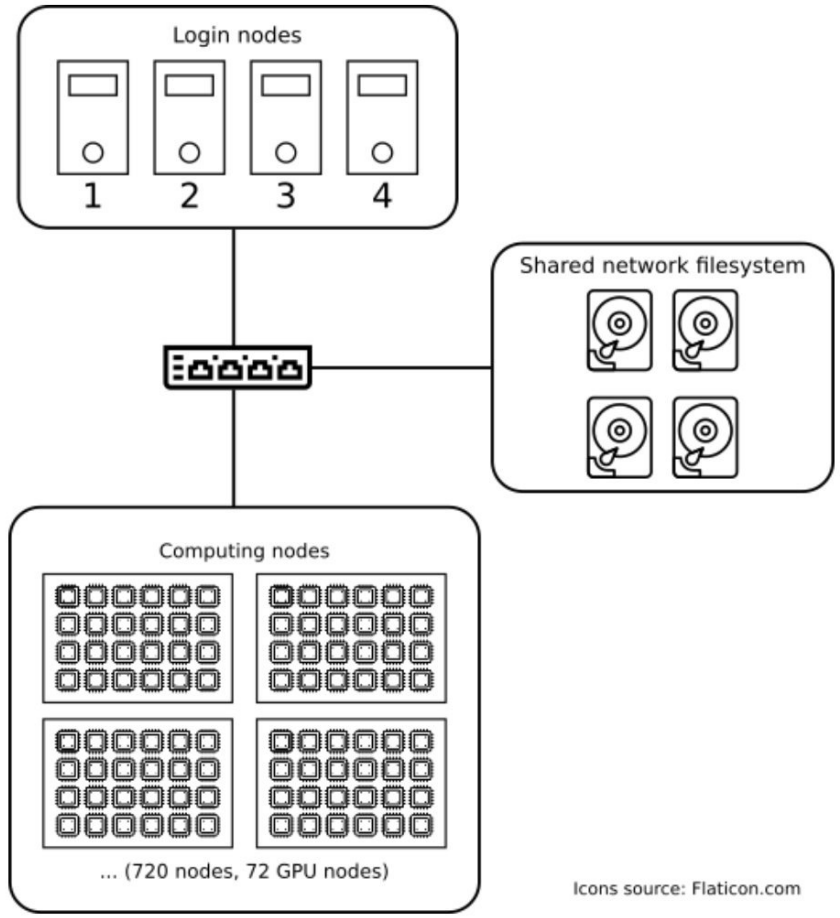
- SSH and SCP clients for Windows
- https://docs.it4i.cz/general/accessing-the-clusters/shell-access-and-data-transfer/putty/

KAROLINA

Login nodes
- program preparation
- job submission

Compute nodes (720 CPU nodes, 72 GPU nodes)
- job execution

Shared filesystem
- code
- job inputs and outputs
- shared between login and compute nodes



KAROLINA FILESYSTEM

HOME workspace (NFS)

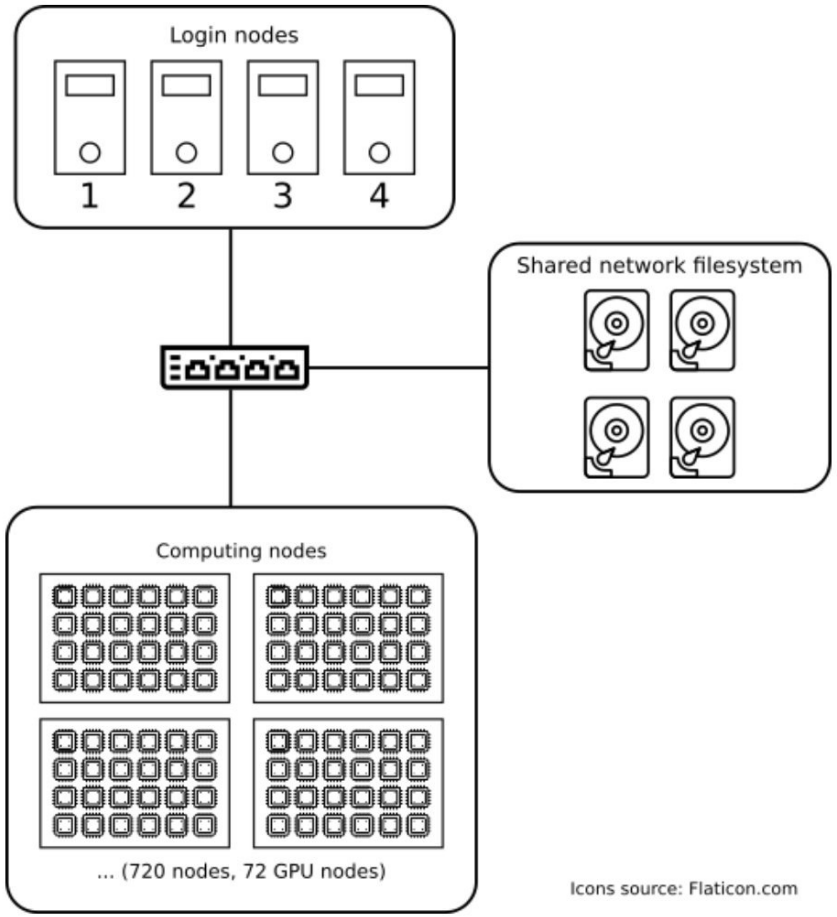
- Located at ~ (your home directory)
- Limited size (~25 GiB), quite slow (2-3 GiB/s)
- Use for config files, build artifacts, source code repositories

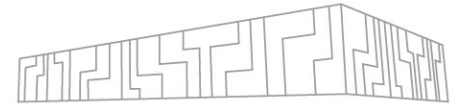
PROJECT workspace (GPFS)

- Very large (~15 PiB), rather slow (40 GiB/s)
- Each project has its own directory (deleted after project ends)
- Central storage for all project data, use for important data
- `$ it4i-get-project-dir <project-id>`

SCRATCH workspace (Lustre)

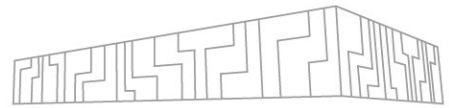
- Located at /scratch/project/<project-id>, no backup
- Large (~20 TiB), very fast (1 TiB/s)
- Use for reading job inputs and writing job results
- Copy results to HOME or PROJECT after the job ends
- **Files are deleted after 90 days of inactivity!**





How to work on Karolina

MODULES



Each IT4I cluster has its own set of pre-installed modules available for immediate use

Module
- is a set of binaries, libraries, header files, …
- has a set of modules that it depends on
- might have several available versions (Python/2.7.9 vs Python/3.6.1)
- might have a specific toolchain (GCC vs Intel toolchain)

To use a module, you must load it
- loading a module modifies environment variables (PATH, LIBRARY_PATH, LD_LIBRARY_PATH)
- this enables executing module binaries and linking to module libraries

Lmod is used to load modules
You can also create your own modules or ask support to install new modules for you
Modules are defined using EasyBuild

If you find a module that is not working, contact support

MODULES

Useful hints

- Always load specific versions of modules to avoid surprises
  - `ml GCC/6.3.0` (OK)
  - `ml GCC` (avoid loading of default module)

- Module load order matters (because of conflicting dependencies)
  - ml A B might produce different results than ml B A

- Filtering modules
  - `ml spider <package>`
  - `ml` command also provides tab completion

- `ml` command is case sensitive
- match module toolchains (GCC vs Intel)
- do not forget to load the correct modules in your job script!

```
# show available modules
$ ml av

# load a module with its dependencies
$ module load Python/3.6.8

# list loaded modules
$ module list
Currently loaded modules:
1) GCC/6.3.0 2) Python/3.6.8
$ python --version
Python 3.6.8

# unload all loaded modules
$ ml purge
$ python --version
Python 2.7.5
```

COMPUTATIONAL PROJECT

Choose the correct computational project for your experiment

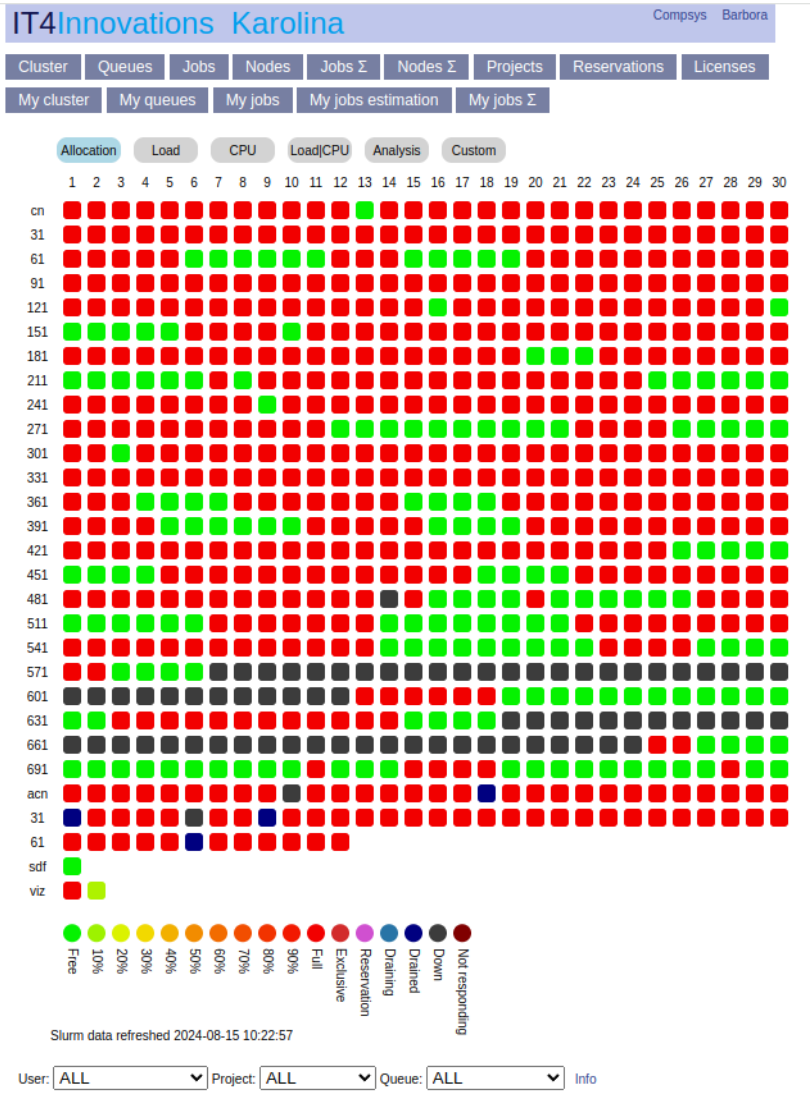
Check status of the cluster

- https://extranet.it4i.cz/rsweb/karolina

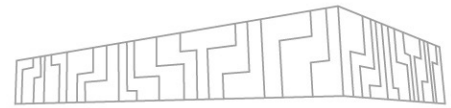
Check how much core hours are left in the project

- https://scs.it4i.cz/
- `$ it4ifree`

```
Projects I am participating in
==============================
PID         Resource type    Days left       Total      Used     By me      Free
----------  ---------------  -----------  ---------  --------  --------  --------
DD-24-74    Karolina CPU     16              1000        30         0       970
            Karolina GPU     16               200        12         0       188
Legend
======
N/A    =    No one used this resource yet
Legacy Normalized core hours are in NCH
Everything else is in Node Hours
```



AVAILABLE QUEUES



Each IT4I cluster is shared by many users

To perform a computation (a job), you must go through a queue

- We use a queuing system called Slurm (https://slurm.schedmd.com/documentation.html)

There are several queues with different properties

- **qcpu_exp**, **qgpu_exp** (quick experiments, do not charge for use, up to 2 nodes and 1-hour jobs)
- **qcpu** (common computations, up to 720 nodes and 2-day jobs)
- **qgpu** (common computations, up to 72 nodes and 2-day jobs)

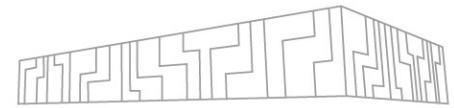
- **qcpu_long** (long-running computations, up to 200 nodes and 6-days jobs)
- **qfat** (fat node – 768 cores, 24 TiB RAM)

- You can find the complete queue list here: https://docs.it4i.cz/general/karolina-partitions/

To access most queues, you will need to specify a computational project that you are a part of

- Computational resources that you spend are deducted from the used project
- You can use the **qcpu_free** and **qgpu_free** up to 150% of the project resources

SLURM PARAMETERS



You can submit jobs on the cluster in two modes

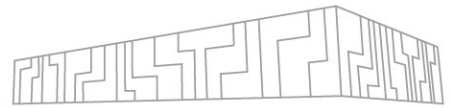
- batch mode (**sbatch**)
  - you specify a script which is executed once you get to the front of a queue
- interactive mode (**salloc**)
  - your terminal will be connected to the first computing node in the job via SSH

You can have multiple jobs in the queue at once (both waiting and executing)
Be careful with walltime to avoid wasting project resources!

You must give basic parameters to define a job:

- `-N, --nodes` number of nodes
- `-n, --ntasks` number of tasks (MPI processes)
- `-c, --cpus-per-task` number of threads per MPI process
- `-p, --partition` requested queue
- `-t, --time` walltime for your job
- `-A, --account` account number (e.g., DD-24-74)
- `-J, --job-name` name of your job
- `-G, --gpus` number of required GPUs

SUBMIT YOUR JOB



You can submit jobs on the cluster in two modes

- batch mode (**sbatch**)
  - you specify a script which is executed once you get to the front of a queue
- interactive mode (**salloc**)
  - your terminal will be connected to the first computing node in the job via SSH

```
$ salloc -A DD-24-74 -p qcpu_exp -N 2 -n 256 -t 00:05:00
salloc: Granted job allocation 1493151
salloc: Waiting for resource configuration
salloc: Nodes cn[160-161] are ready for job
$ ml OpenMPI/4.1.4-GCC-11.3.0
$ srun hostname | sort | uniq -c
    128 cn160.karolina.it4i.cz
    128 cn161.karolina.it4i.cz
```

Run interactive job using 2 nodes (160,161)

- 128 cores per node (256 cores total)

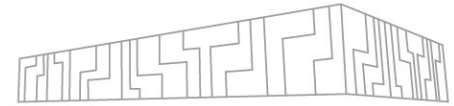
```
#!/usr/bin/bash
#SBATCH --job-name TEST
#SBATCH --account DD-24-74
#SBATCH --partition qcpu_exp
#SBATCH --nodes 2
#SBATCH --ntasks-per-node 128
#SBATCH --time 00:05:00

ml purge
ml OpenMPI/4.1.4-GCC-11.3.0

srun hostname | sort | uniq -c
```

sbatch script.sh
… wait for completion …
cat slurm-<JOB-ID>.out

SLURM UTILITIES



Starting jobs:

- `salloc -A PROJECT-ID -p qcpu -N 4 -n 128 –c 4 -t 2:00:00`
- `sbatch –A PROJECT-ID script.sh`

Start the job (**use `srun` instead of `mpirun`!**):

- `srun –n 128 ./app`

Get info about queued jobs:

- `squeue --me`

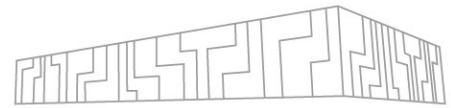
Job canceling:

- `scancel JOBID`

Informations about nodes and partitions:

- `sinfo`

JOB EXECUTION



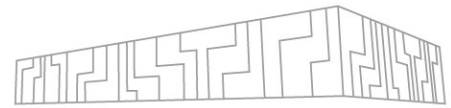
Once the job gets to the front of the queue

1. Slurm will allocate the specified number of nodes
2. The specified script will be executed
    - on the first allocated node
    - in submit directory
    - with all loaded modules before submitting (hence, purge modules in your script)
3. Once your script finishes, the job will also end
4. stdout and stderr of your script will be written to a file on the shared filesystem
    - slurm-<JOB-ID>.out
    - they will be stored in the directory where you submit the job
        - You can override this location with -o and –e

Useful environment variables available during a job

- SLURM_JOB_ID – job id of the execution job
- SLURM_JOB_NUM_NODES – number of nodes allocated to the job
- SLURM_JOB_NODELIST – nodes allocated to the job

MONITORING JOB STATUS



Once your job starts running, you can observe its status in several ways

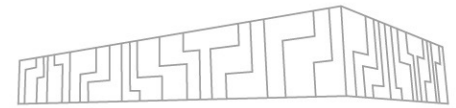
`$ squeue --me`

- Displays status of my queues, elapsed time, allocated computing nodes
- You can connect to the individual computing nodes via SSH to inspect them

```
[mec059@login2.karolina 01_hello]$ sbatch run.slurm
Submitted batch job 1143932
[mec059@login2.karolina 01_hello]$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1143920  qcpu_exp  zphpc01   mec059 CD       0:03      1 cn553
           1143922  qcpu_exp  zphpc01   mec059  F       0:04      1 (NonZeroExitCode)
           1143932  qcpu_exp  zphpc01   mec059  R       0:04      1 cn147
[mec059@login2.karolina 01_hello]$ ssh cn147
[mec059@login2.karolina 01_hello]$ htop
```

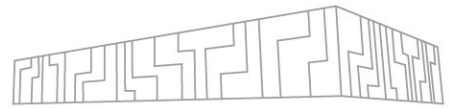
When something goes wrong you can delete jobs (both running and enqueued)

- **`$ scancel <JOBID>`**



How to run GUI applications

ACCESSING KAROLINA



1. **VNC** - server on a Karolina login node + client on laptop
   - How to? https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/
   - Recommended client https://www.realvnc.com/en/connect/download/viewer/
2. **OOD** - Open OnDemand GUI via web browser, **IT4I VPN required**
   - How to? https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/ood/
   - Connection link https://ood-karolina.it4i.cz/
3. **X11** - Log in via terminal with X-Window system enabled
   - How to? https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/x-window-system/
   - Usually worse UX for GUI apps due to network latency

ACCESSING KAROLINA

DO NOT CTRL+C, CTRL-V

SOME CHARACTERS CAN BE INCORRECTLY COPIED

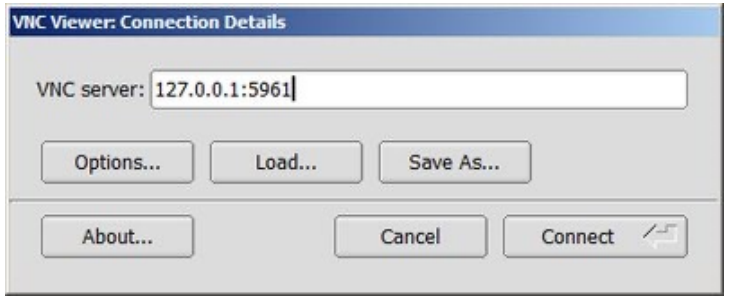
GUI applications via VNC

- https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/

1. Connect to a login node
   - ssh, putty
2. Set VNC password
   - `$ vncpasswd`
3. Check available ports
   - `$ ps aux | grep Xvnc | sed -rn 's/(\s) .*Xvnc (\:[0-9]+) .*/\1 \2/p'`
4. Start VNC server on an available port
   - `$ vncserver :61 -geometry 1600x900 -depth 16`
5. Open the tunnel on your laptop
   - `$ ssh -TN -f username@login2.karolina.it4i.cz -L 5961:localhost:5961`
6. Start VNC viewer on your laptop
   - e.g., TigerVNC

the same login where you started the server

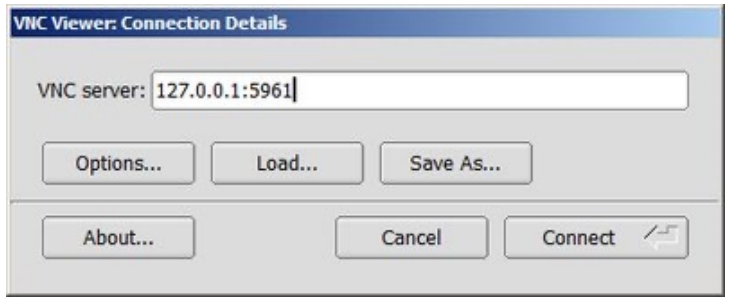
port number: 5900 + 61

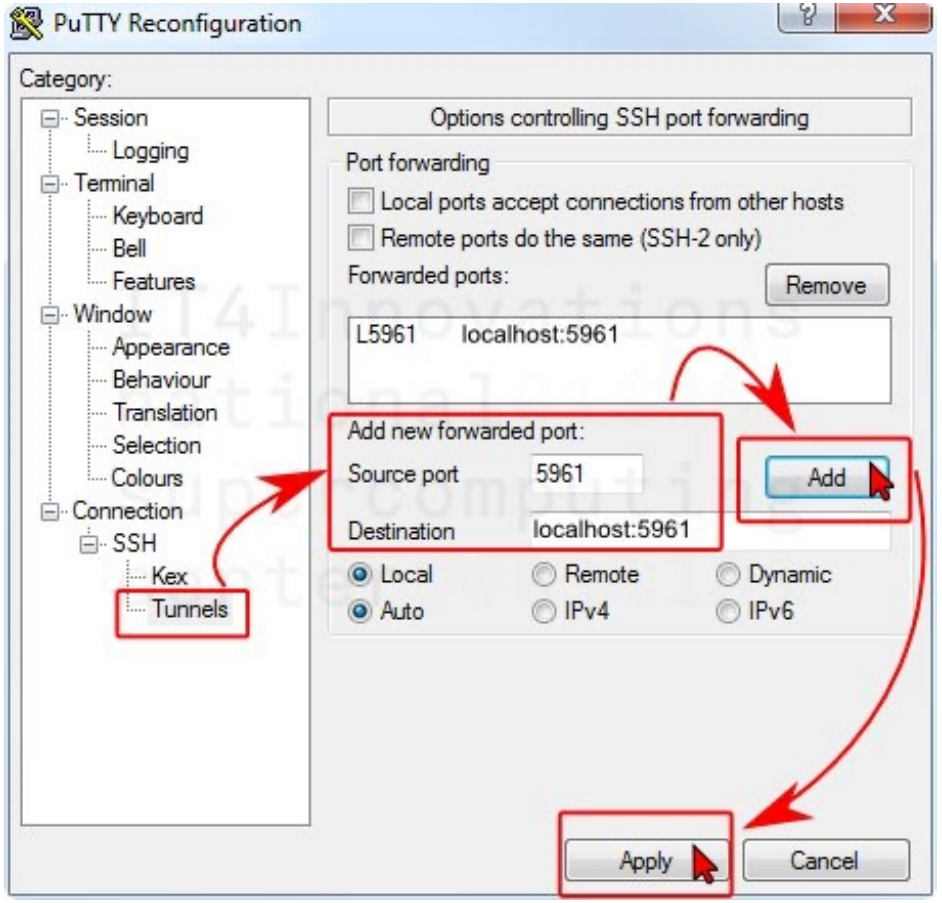


ACCESSING KAROLINA

PuTTY, WinSCP

- Use PuTTy to create the tunnel
- add port forwarding to VNC server





ACCESSING KAROLINA

LOGIN

COMPUTE NODE

Connect to a login node
ssh, putty

Check ports

```
$ netstat -natp | grep 5961
$ vncserver --list
```

e.g., on login 1

```
$ vncserver :61 -geometry 1600x900 -depth 16
```

Open the tunnel
ssh -TN -f username@login1.karolina.it4i.cz -L **5961**:localhost:**5961**

Start VNC client with port **5961**

do not forget to kill the server

```
$ vncserver -kill :61
```

DO NOT CTRL+C, CTRL-V

SOME CHARACTERS CAN BE INCORRECTLY COPIED

ACCESSING KAROLINA

LOGIN

COMPUTE NODE

Connect to a login node
ssh, putty

Check ports
```
$ netstat -natp | grep 5961
$ vncserver --list
```

e.g., on login 1
```
$ vncserver :61 -geometry 1600x900 -depth 16
```

Open the tunnel
ssh -TN -f username@login1.karolina.it4i.cz -L
5961:localhost:**5961**

Start VNC client with port **5961**

e.g., in TigerVNC Viewer
in terminal 1:
```
$ salloc --x11 -pqcpu -ADD-24-74 –N1
```

in terminal 2:
```
$ ssh -X cn123
```

start GUI application

DO NOT CTRL+C, CTRL-V

SOME CHARACTERS CAN BE INCORRECTLY COPIED

ACCESSING KAROLINA

LOGIN

COMPUTE NODE

Connect to a login node
ssh, putty

e.g., on login 1

```
$ ml Anaconda3
$ ml jupyter-lab
```

check port, where run the server, e.g., **8888**
copy http address http://localhost:**8888**/lab?token=**....**

Open the tunnel
ssh -TN -f username@login1.karolina.it4i.cz -L
8888:localhost:**8888**

Start Jupyter-lab in the browser:
http://localhost:**8888**/lab?token=**....**

DO NOT CTRL+C, CTRL-V

SOME CHARACTERS CAN BE INCORRECTLY COPIED

ACCESSING KAROLINA

LOGIN

COMPUTE NODE

Connect to a login node
ssh, putty

e.g., on **login 1**

```
$ salloc -pqcpu -ADD-24-74 –N 1
```

e.g., on **cn123**

```
$ ml Anaconda3
$ jupyter-lab
```

copy http address and **port**

check availability of a port
netstat -natp | grep **8888**

tunnel from login to node cn123

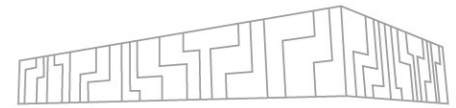
```
ssh -TN -f cn123 -L 8888:localhost:8888
```

Open the tunnel
ssh -TN -f username@**login1**.karolina.it4i.cz -L **8888**:localhost:**8888**

Start Jupyter-lab in the browser:
http://localhost:**8888**/lab?token=**....**

DO NOT CTRL+C, CTRL-V

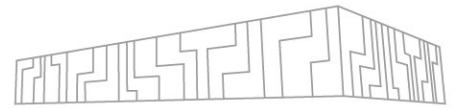
SOME CHARACTERS CAN BE INCORRECTLY COPIED



How to run your job efficiently

(mapping, pinning)

INTERACTIVE JOB EXECUTION



Threaded application on a single node (OpenMP):

- `$ salloc -N 1 –n 1 ...`
- `$ OMP_NUM_THREADS=128 srun –n 1 ./app`

Pure MPI application on several nodes:

- `$ salloc -N 4 –n 512 ...`
- `$ srun –n 512 ./app`

Hybrid application on several nodes (MPI+OpenMP):

- `$ salloc -N 4 –n 512 ...`
- `$ OMP_NUM_THREADS=4 srun –n 128 ./app`

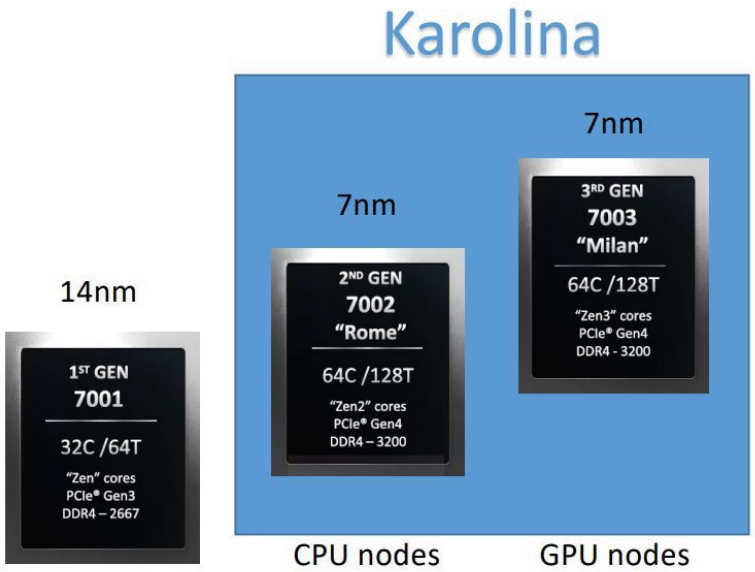
INTERACTIVE JOB EXECUTION

Pure MPI application:

- `$ salloc -N 1 –n 128 ...`
- `$ srun –n 128 ./app`

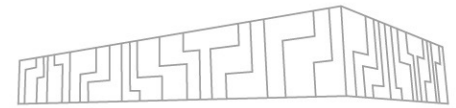
MPI+OpenMP application:

- `$ salloc -N 1 –n 128 ...`
- `$ OMP_NUM_THREADS=4 srun –n 32 ./app`



CATEGORY	EPYC 7002 (Rome)	EPYC 7003 (Milan)
Socket	SP3	SP3
Core / Process	Zen2 / 7nm	Zen3 / 7nm
Max Core Count / Threads	64 / 128	64 / 128
L3 Cache Size	256 MB	256 MB
CCX Arch	4 Cores + 16MB	8 Cores + 32MB
Memory	8 Ch DDR4-3200, NVDIMM-N	8 Ch DDR4-3200, NVDIMM-N
PCIe Tech & Lane Count	PCIe Gen4, 128L/Socket	PCIe Gen4, 128L/Socket
Security	SME, SEV	SME, SEV, SNP
Chipset	NA	NA
Power	120W - 280W	120W - 280W

JOB EXECUTION



Pure MPI application:

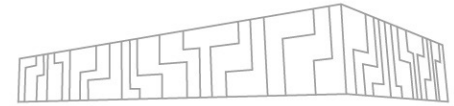
- `$ salloc -N 1 –n 128 ...`
- `$ srun –n 128 ./app`

MPI+OpenMP application:

- `$ salloc -N 1 –n 128 ...`
- `$ OMP_NUM_THREADS=4 srun –n 32 ./app`

Is it the best possible setting?

JOB EXECUTION



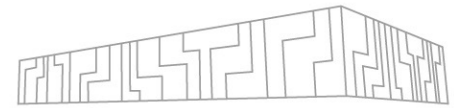
I have a simple application:

- Karolina supercomputer
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- `salloc -p qcpu_exp -N 1`
- `export OMP_NUM_THREADS=1`

- `srun –n   8 ./app 39.66s`
- `srun –n  16 ./app 17.46s`
- `srun –n  32 ./app 11.87s`
- `srun –n  64 ./app  7.31s`
- `srun –n 128 ./app  5.56s`

JOB EXECUTION



I have a simple application:

- Karolina supercomputer
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- `salloc -p qcpu_exp -N 1`
- `export OMP_NUM_THREADS=1`

- `srun -n   8 ./app 39.66s`
- `srun -n  16 ./app 17.46s`
- `srun -n  32 ./app 11.87s`
- `srun -n  64 ./app  7.31s`
- `srun -n 128 ./app  5.56s`

What Is it the best possible setting?

Why app does not scale well?

MAPPING, PINNING

Mapping:

- specifies how the software components are mapped to a given hardware

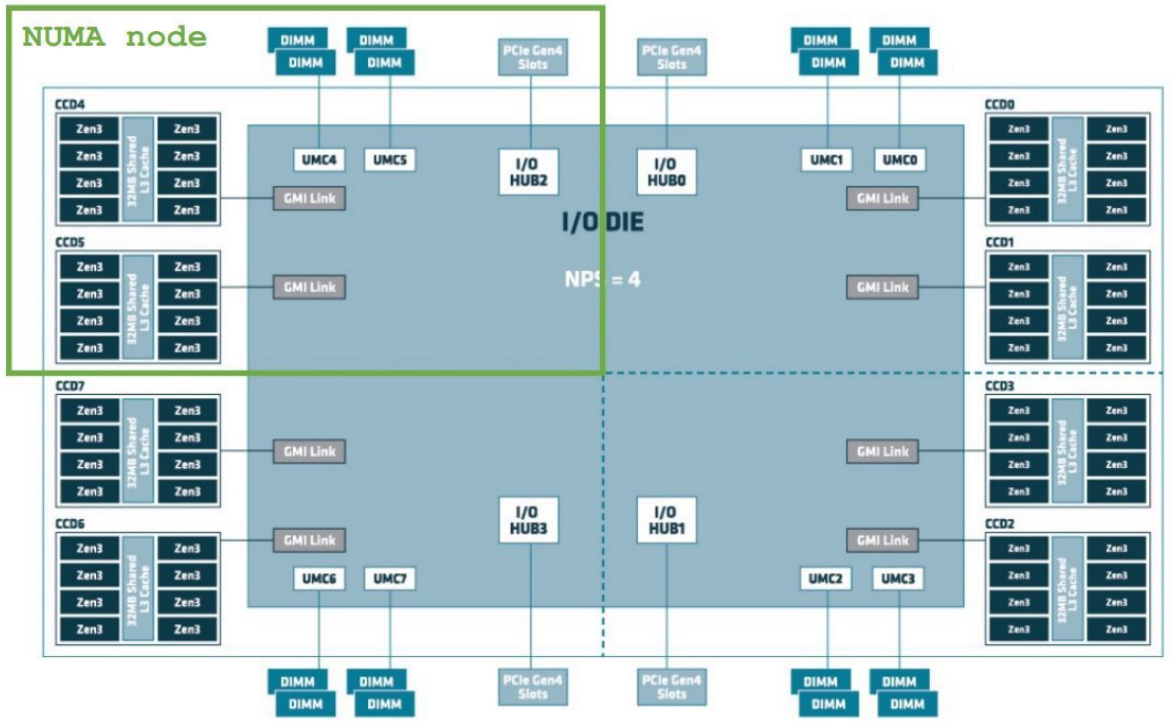
Pinning, binding:

- deny migration of threads and processes to another resources

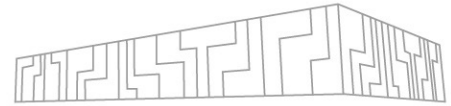
	0	1	2	3	4	5	6	7
0	10	12	12	12	32	32	32	32
1	12	10	12	12	32	32	32	32
2	12	12	10	12	32	32	32	32
3	12	12	12	10	32	32	32	32
4	32	32	32	32	10	12	12	12
5	32	32	32	32	12	10	12	12
6	32	32	32	32	12	12	10	12
7	32	32	32	32	12	12	12	10

```
numactl -H
| node 0 cpus: 0   -  15
| node 1 cpus: 16  -  31
| node 2 cpus: 32  -  47
| node 3 cpus: 48  -  63
| node 4 cpus: 64  -  79
| node 5 cpus: 80  -  95
| node 6 cpus: 96  - 111
| node 7 cpus: 112 - 127
| node 0-7 size: 128 GB
```

	0	1	2	3
0	10	12	12	12
1	12	10	12	12
2	12	12	10	12
3	12	12	12	10



MAPPING, PINNING



Intel-MPI (environment variables)

- KMP_AFFINITY
- I_MPI_PIN_DOMAIN
- https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/interoperability-with-openmp-api.html

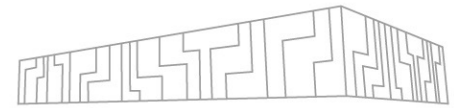
OpenMPI (arguments of mpirun)

- --bind-to <hwthread, core, socket, numa, ...>
- --map-by <hwthread, core, socket, numa, ...>
- --report-bindings
- https://www.open-mpi.org/doc/v4.0/man1/mpirun.1.php

Slurm:

- --cpu-bind={sockets,ldoms,cores}
- -c, --cpus-per-task=<ncpus>

MAPPING, PINNING



I have a simple application:

- Karolina supercomputer
- compile with OpenMPI: `mpic++ -fopenmp -O3 -march=native app.cpp -o app`
- test with different number of MPI processors up to 128

- `salloc -p qcpu_exp -N 1`
- `export OMP_NUM_THREADS=1`

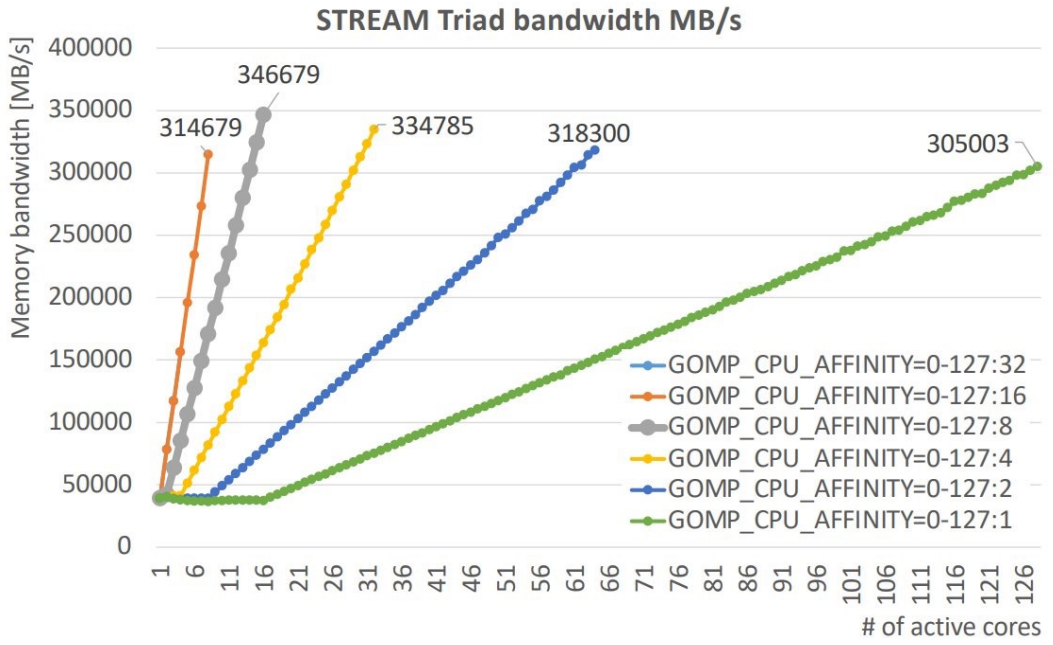
- `srun -n   8 ./app 39.66s` `srun -n   8 -c 16 ./app 7.11s`
- `srun -n  16 ./app 17.46s` `srun -n  16 -c  8 ./app 5.21s`
- `srun -n  32 ./app 11.87s` `srun -n  32 -c  4 ./app` **`5.08s (9% better)`**
- `srun -n  64 ./app  7.31s` `srun -n  64 -c  2 ./app 5.31s`
- `srun -n 128 ./app` **`5.56s`** `srun -n 128 -c  1 ./app 5.56s`

MAPPING, PINNING

Memory bound application
- Number of MPI processes / thread equal to memory channels
- Correct pinning to NUMA domains (sockets, chiplets)

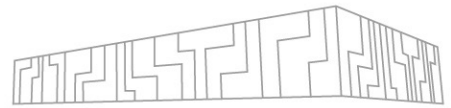
Compute bound application
- As many MPI processes / threads as possible

Your application?
- Test performance for different number of cores per node:
  - 16, 32, 64, 128 cores per node
- Test different mapping / pinning options
  - close, spread



GOMP_CPU_AFFINITY	0-127:32	0-127:16	0-127:8	0-127:4	0-127:2	0-127:1
# of active CPU cores	4	8	16	32	64	128
Max bandwidth [GB/s]	153,1	307,3	338,6	326,9	310,8	297,9
Efficiency	45,2%	90,8%	100,0%	96,6%	91,8%	88,0%

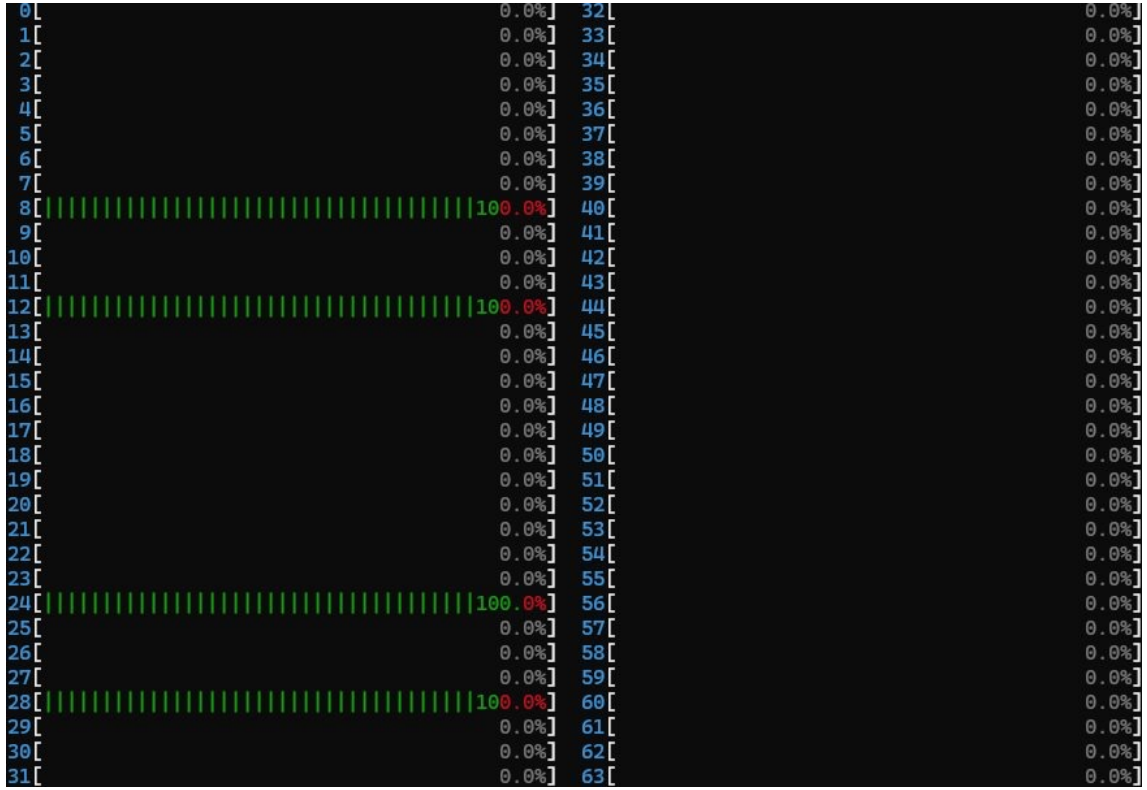
MAPPING, PINNING



More advanced binding (e.g., run on cpus as close to GPUs as possible)

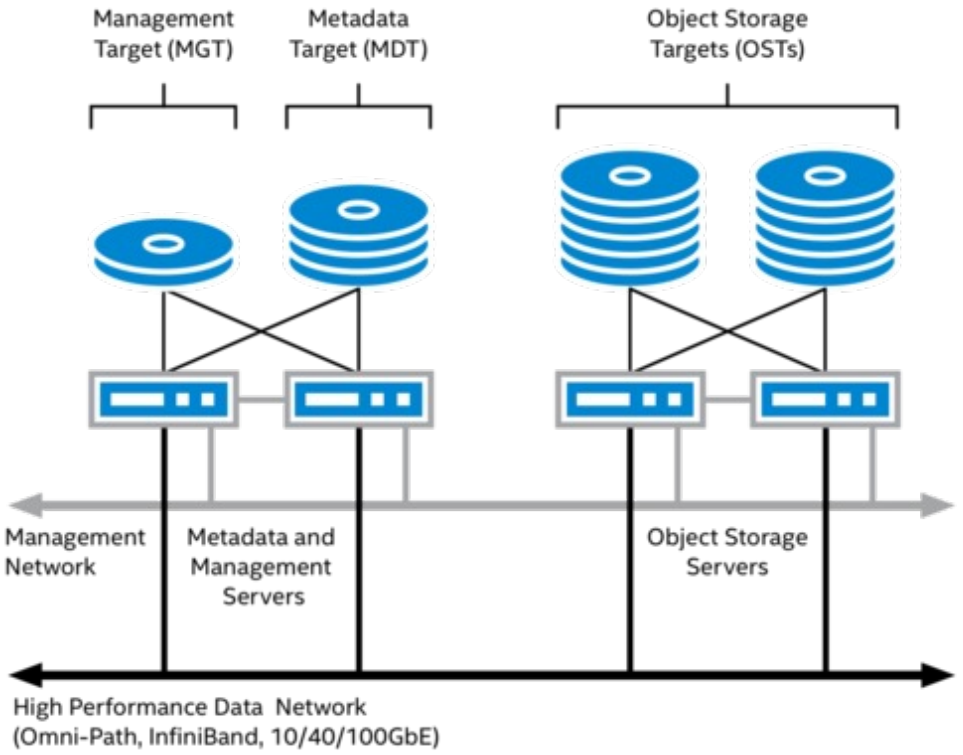
- GPUs are connected to NUMA domains 1, 3, 5, 7 (each of those NUMA domains have 2 GPUs)

```
srun -n 4 --cpu-bind=mask_cpu:0x100,0x1000,0x1000000,0x10000000 ./app
```



PARALLEL FILESYSTEM

SCRATCH filesystem	
Mountpoint	/scratch
Capacity	1361 TB
Throughput	730.9 GB/s write, 1198.3 GB/s read
Default stripe size	1 MB
Default stripe count	1
Protocol	Lustre



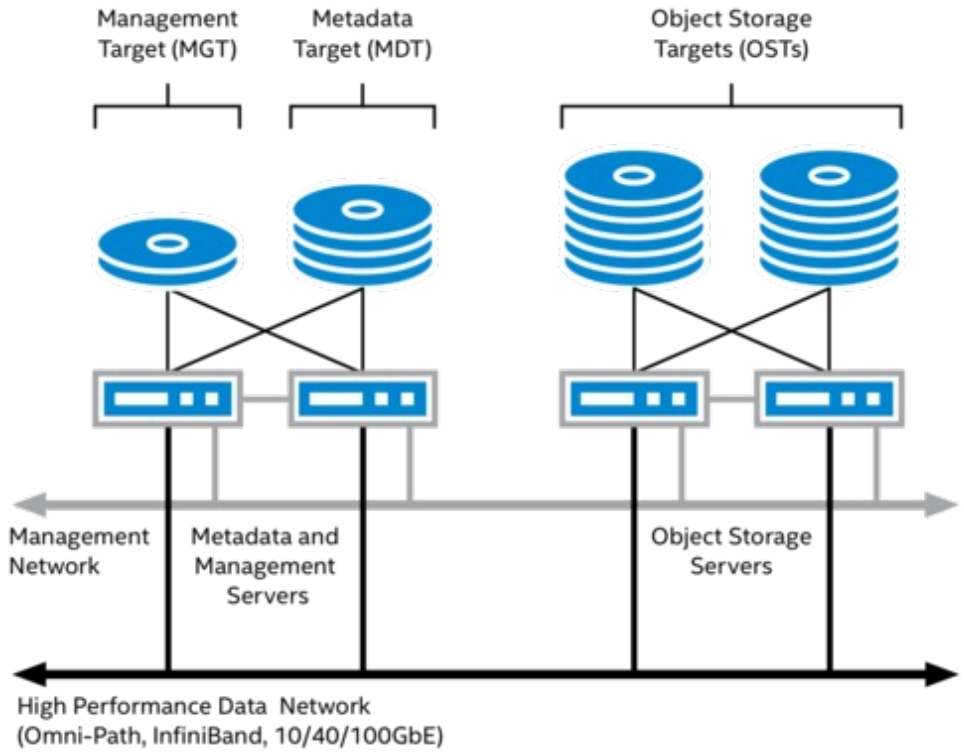
PARALLEL FILESYSTEM

Lustre filesystem

- Metadata server for storing information about a file

- A file is divided into multiple chunks (stripes)
- Each chunk can be stored on different object storage target (disk)
- Round-robin distribution

- Assure coherency when multiple clients access the same file



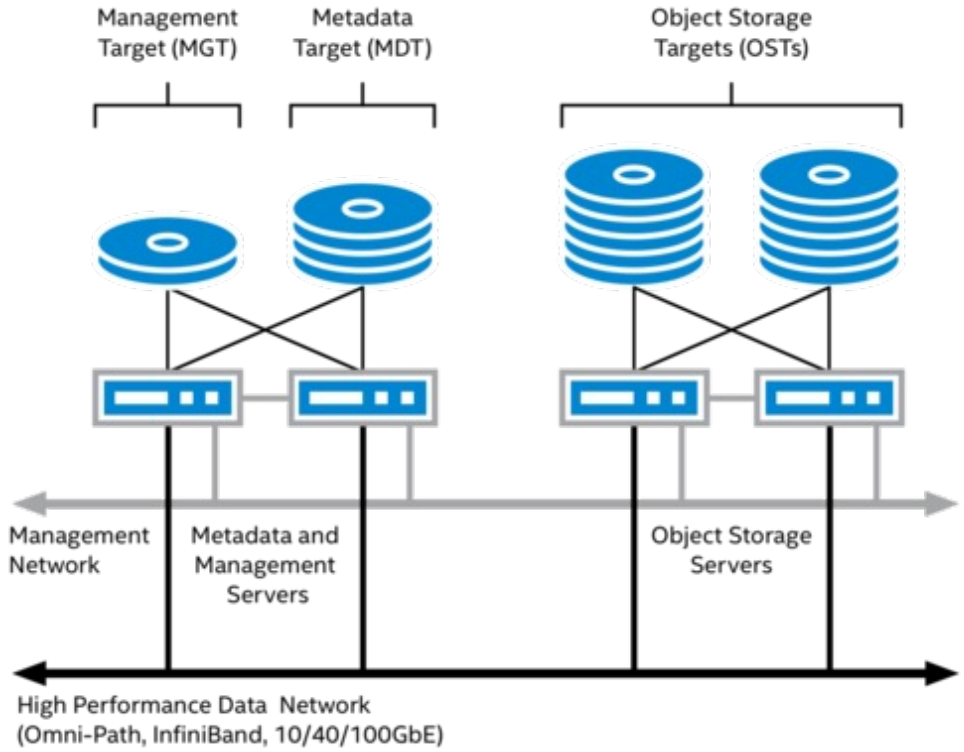
PARALLEL FILESYSTEM

Lustre filesystem settings

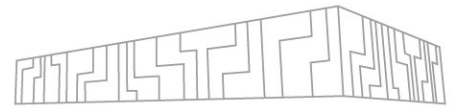
- stripe_size
  - the size of the chunk in bytes
  - the size must be an even multiple of 65,536 bytes
  - default is 1MB

- stripe_count
  - the number of OSTs to stripe across
  - default is 1
  - specify -1 to use all OSTs in the filesystem

- stripe_offset
  - index of the OST where the first stripe is to be placed
  - default is -1 which results in random selection
  - using a non-default value is NOT recommended



PARALLEL FILESYSTEM



Get stripe settings

- `$ lfs getstripe dir|filename`

Set stripe settings

- `$ lfs setstripe -s stripe_size -c stripe_count dir|filename`

Set stripping according to your application needs

- Performance for large files improve when the stripe_count is set to a larger value
- Large files should use stripe counts of greater than 1
- A rule of thumb is to use a stripe count approximately equal to the number of gigabytes in the file.

- Make the stripe count be an integral factor of the number of processes performing the write in parallel
- It achieves load balance among the OSTs

- https://doc.lustre.org/lustre_manual.xhtml#managingstripingfreespace

Ondrej Meca
ondrej.meca@vsb.cz

IT4Innovations National Supercomputing Center
VSB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research, Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS