An Introduction to HPC

High Performance Computing & Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

UNIVERSITÉ DU LUXEMBOURG

Getting started with HPC

Basic components and architecture of HPC systems

Outline

- The main components of an HPC system
  - Login nodes and scheduler
  - Compute nodes
  - Interconnect
  - Storage
  - Accelerators
- The scheduler and cluster resources
  - The scheduler
  - Accelerators and special resource types
  - Process and thread placement
  - Job stages and job dependencies
- Software distribution
  - Modules
  - Containers
  - Environments
- Partnerships and specialized systems

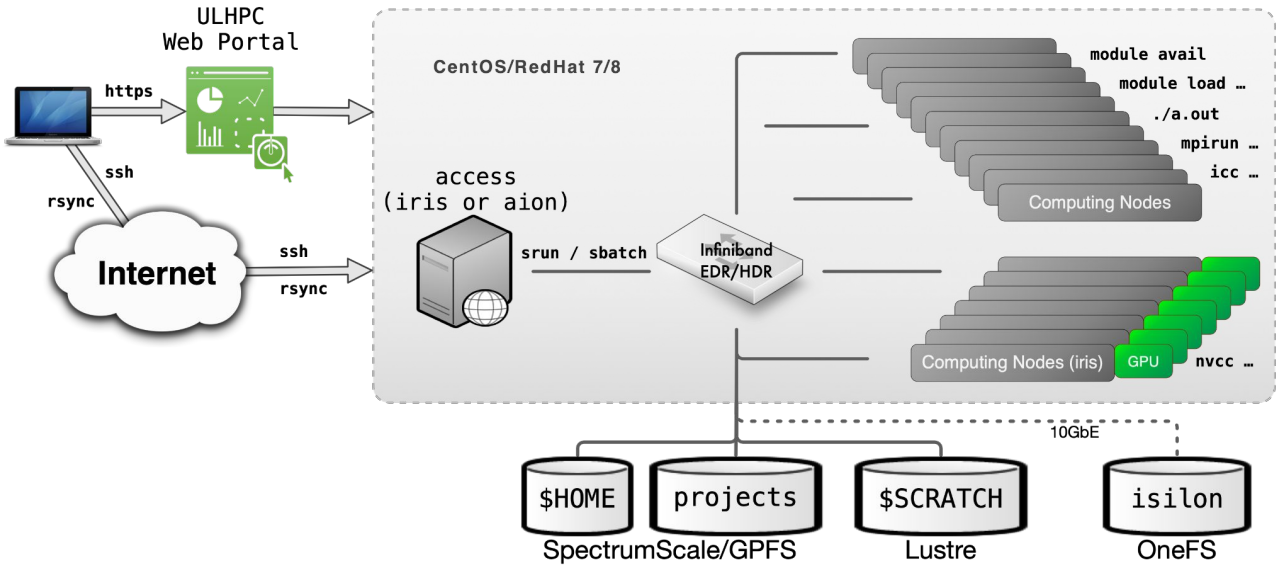
The main clusters

HPC platform of the University of Luxembourg

UNIVERSITÉ DU LUXEMBOURG

HPC clusters at the University of Luxembourg

A typical tier-2 facility:



The main clusters

HPC platform of the University of Luxembourg

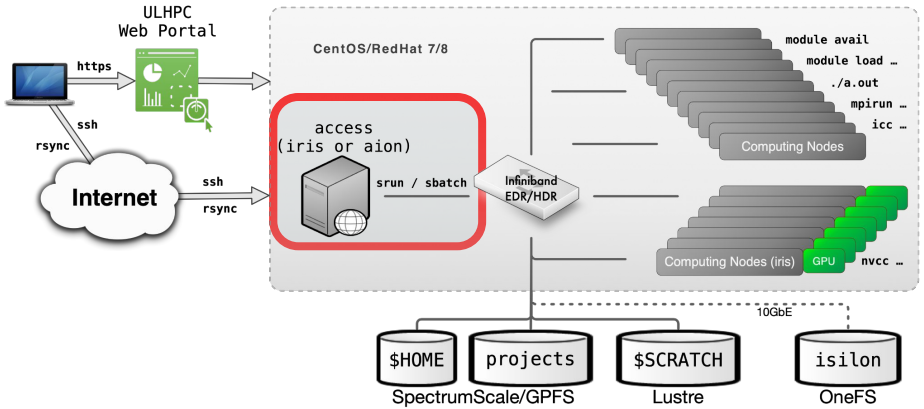
HPC clusters at the University of Luxembourg

Access nodes and the scheduler

- 2 access nodes visible to the outside network
- Slurm scheduler used in UL HPC

HPC clusters designed around resource allocation:

- Allocate resources: `salloc`/`sbatch`
- Use resources: `srun`
- Finite duration resource allocation



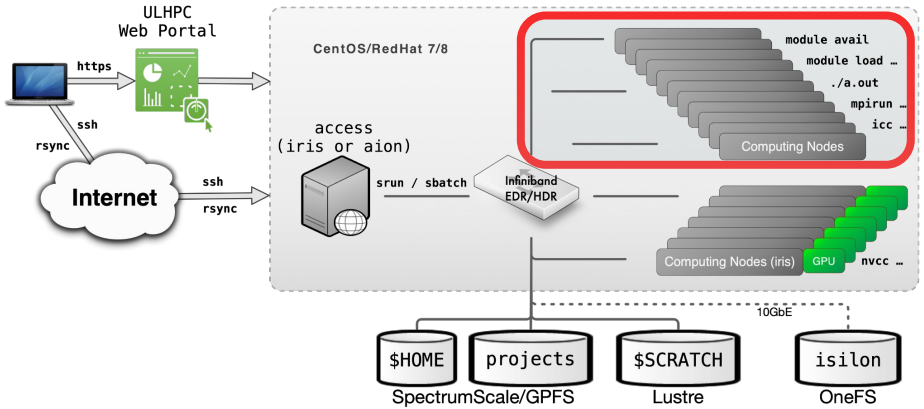
HPC clusters at the University of Luxembourg

Aion

- 354 compute nodes (2 CPUs AMD Epyc ROME 7H12)
- 128 cores and 256GB of memory per node
- 45312 compute cores in total

Used mainly in Physics and Engineering simulations:

- CFD and mechanics (FEA and DEM)
- Molecular Dynamics
- Quantum dynamics (Quantum Monte Carlo)



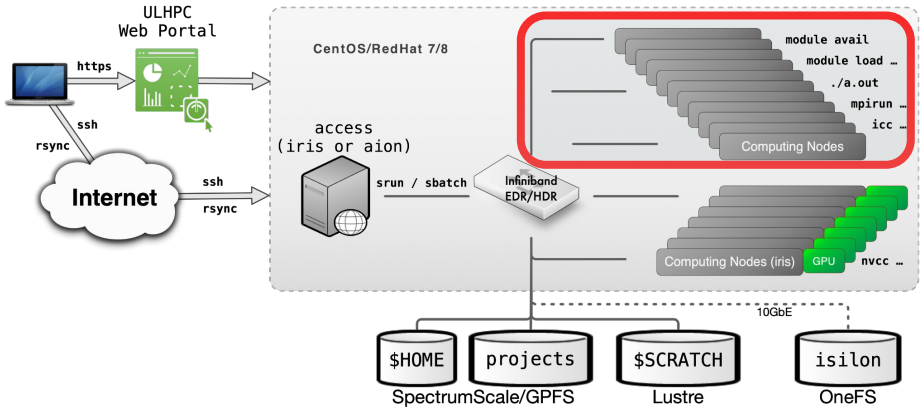
HPC clusters at the University of Luxembourg

Iris CPU

- Old production cluster
- 130 compute nodes (2 CPUs 2 Xeon E5-2680v4)
- 28 cores and 128GB of memory per node
- 3640 compute cores in total

Used mainly for interactive jobs

- Debugging
- Jupyter and R notebooks
- Data visualization
- Graphical desktop environments



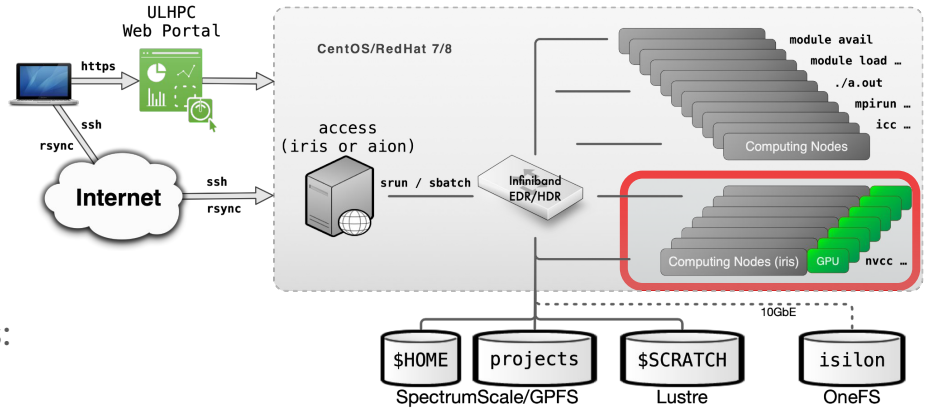
HPC clusters at the University of Luxembourg

Iris GPU and Large Memory

- 24 GPU nodes:
  - 4 V100-SXM2 per node
  - 32GB VRAM per GPU card
  - 768GB RAM per node
- 4 Large memory nodes:
  - 4 Xeon Platinum 8180M 28cores
  - 3072GB RAM per node

Used mainly for simulation and machine learning jobs:

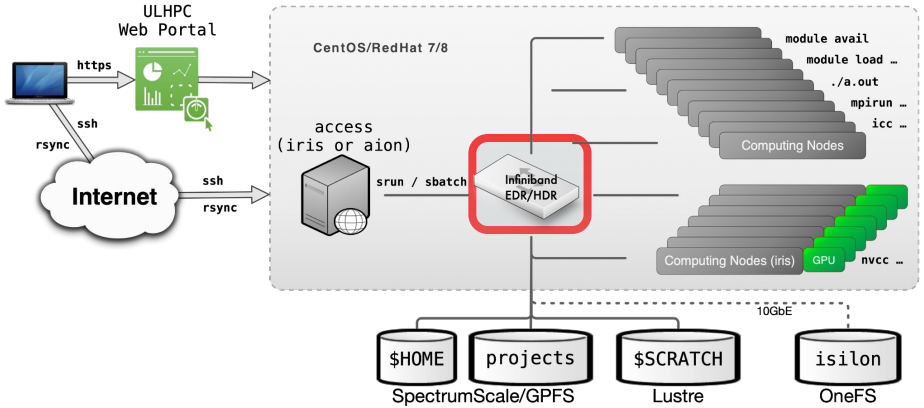
- Molecular Dynamics
- DNNs



HPC clusters at the University of Luxembourg

Interconnect

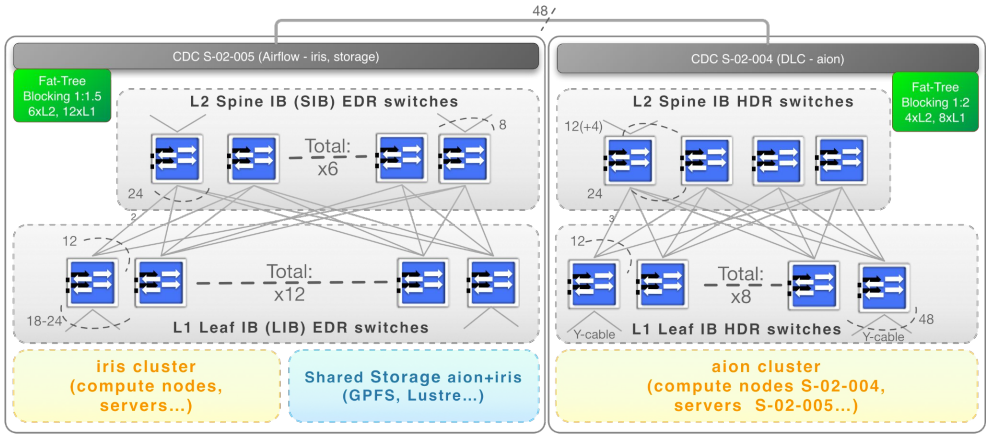
- InfiniBand interconnect
- Ethernet Network



HPC clusters at the University of Luxembourg

InfiniBand interconnect

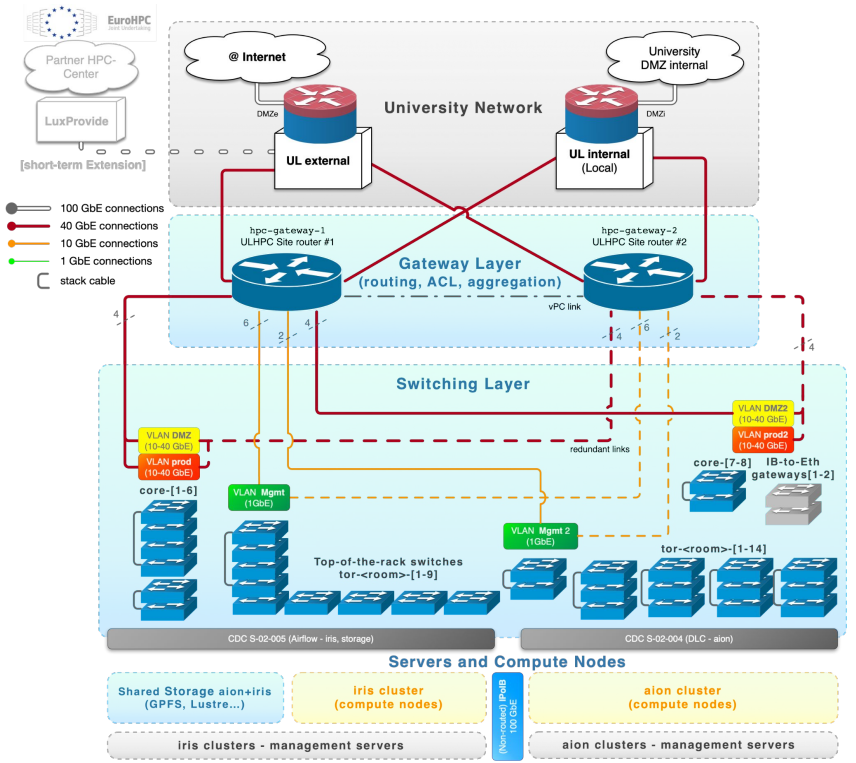
- single high-bandwidth and low-latency network
- Connects Iris and Aion nodes
- Fat-Tree topology
- Provides fast access to clustered storage



HPC clusters at the University of Luxembourg

Ethernet Network

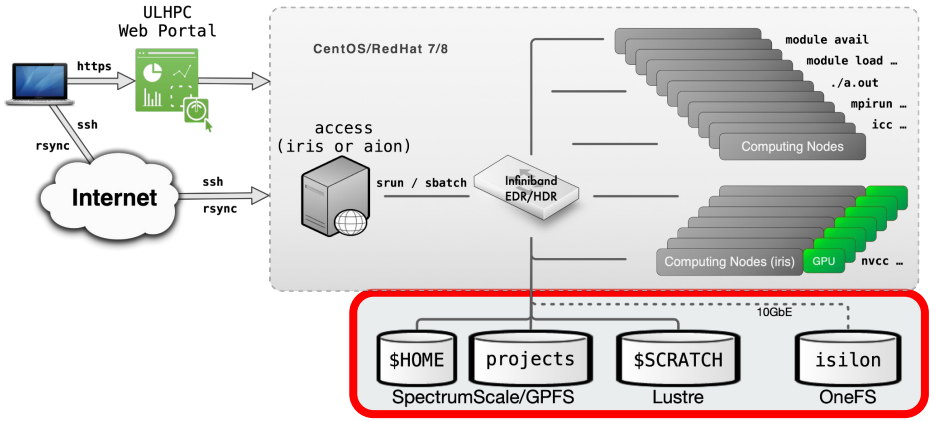
- Access to the internet for all nodes
- Access to high capacity term storage
- Optimized for data transfers



HPC clusters at the University of Luxembourg

Storage

Mount point	File system	Backup	Access
HOME	GPFS (cache)	No	Infiniband
PROJECTHOME	GPFS	No	Infiniband
SCRATCH	Lustre	No	Infiniband
/tmp	ext4	Job based lifetime	local
/mnt/isilon	OneFS	Yes	Ethernet



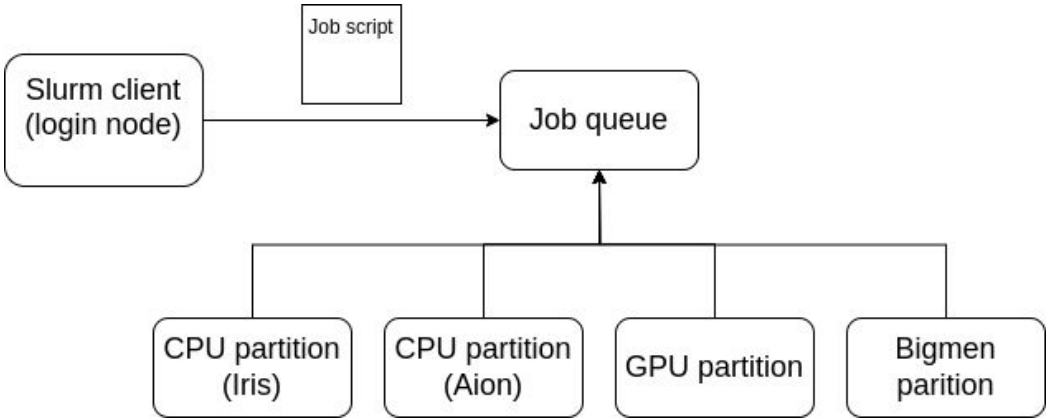
Scheduler and cluster resources

HPC platform of the University of Luxembourg

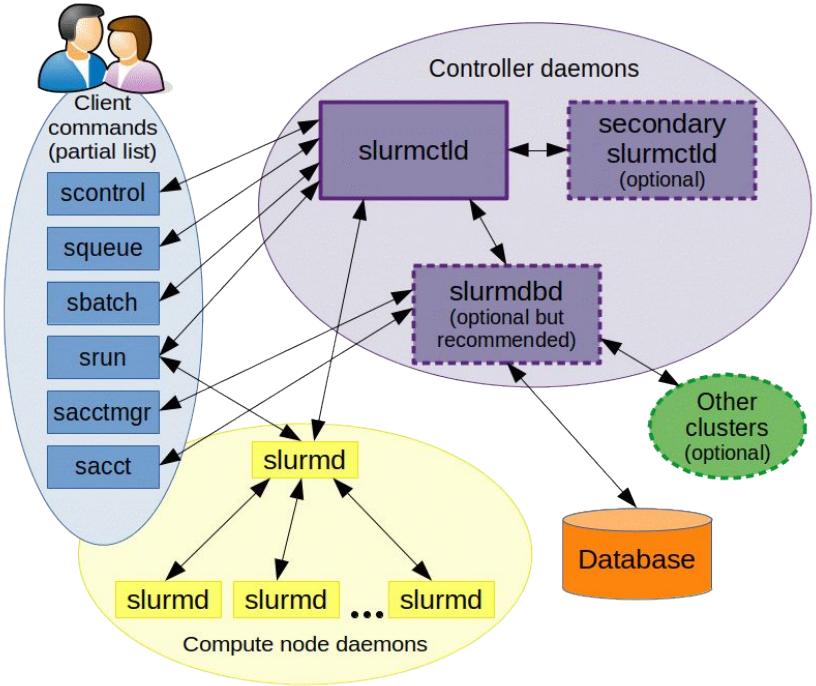
The scheduler

The Slurm scheduler controls access to resources.

- Users submit a batch script or request an interactive allocation.
- When the resources are available their job will launch.
- There are various methods to group and request nodes depending on the resources they provide.



Submission script



On the login node:

- Submit jobs

```
$ sbatch submission_script.sh
```

- Check queued job status

```
$ squeue --user=<username>
```

- Check job details

```
$ sstat
$ seff
```

Source: slurm.schedmd.com

Partitions

- In Slurm multiple nodes with shared features are grouped into partitions.
- Explicitly request nodes with specifics features by specifying feature tags with the `--constraint` option flag.

Type	Default/MaxTime	MaxNodes (per job)
interactive	30min - 2h	2
batch (cpu)	2h-48h	64
gpu	2h-48h	4
bigmem	2h-48h	1

QoS

QoS (Quality of Service)

Quality of Service or QoS is a set of job constraints used to amend the constraints of a job.

For example: l**onger run time** or a **high priority queue** for a given job.

You can type `sqos` to learn about all existing QoS and their restrictions.

Job constraints

1. Partition QoS limit
2. Job QoS limit
3. User association
4. Account association(s), ascending the hierarchy
5. Root/Cluster association
6. Partition limit

QoS

Type	Max # of running jobs	Max duration
normal	100	2 days
long	4 per users, 6 per user group	14 days
best effort	300	50 days

Interesting QoS

- **long**: for longer jobs, max **4** jobs per user, up to **14** days
- **besteffort**: a preemptible (your jobs can be killed when the cluster is too busy with other normal jobs and restarted when resources are available again), max **100** jobs per user, up to **50** days

Submission script

```
#!/bin/bash --login

#SBATCH --job-name=gpu_example
#SBATCH --output=%x-%j.out
#SBATCH --error=%x-%j.err

### Request one GPU tasks for 4 hours and 1/4 of available cores
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=7
#SBATCH --gpus-per-task=1
#SBATCH --time=0-04:00:00

### Submit to the `gpu` partition of Iris
#SBATCH --partition=gpu
#SBATCH --qos=normal

srun job_script
```

Content of the submission script `submission_script.sh` on the left.

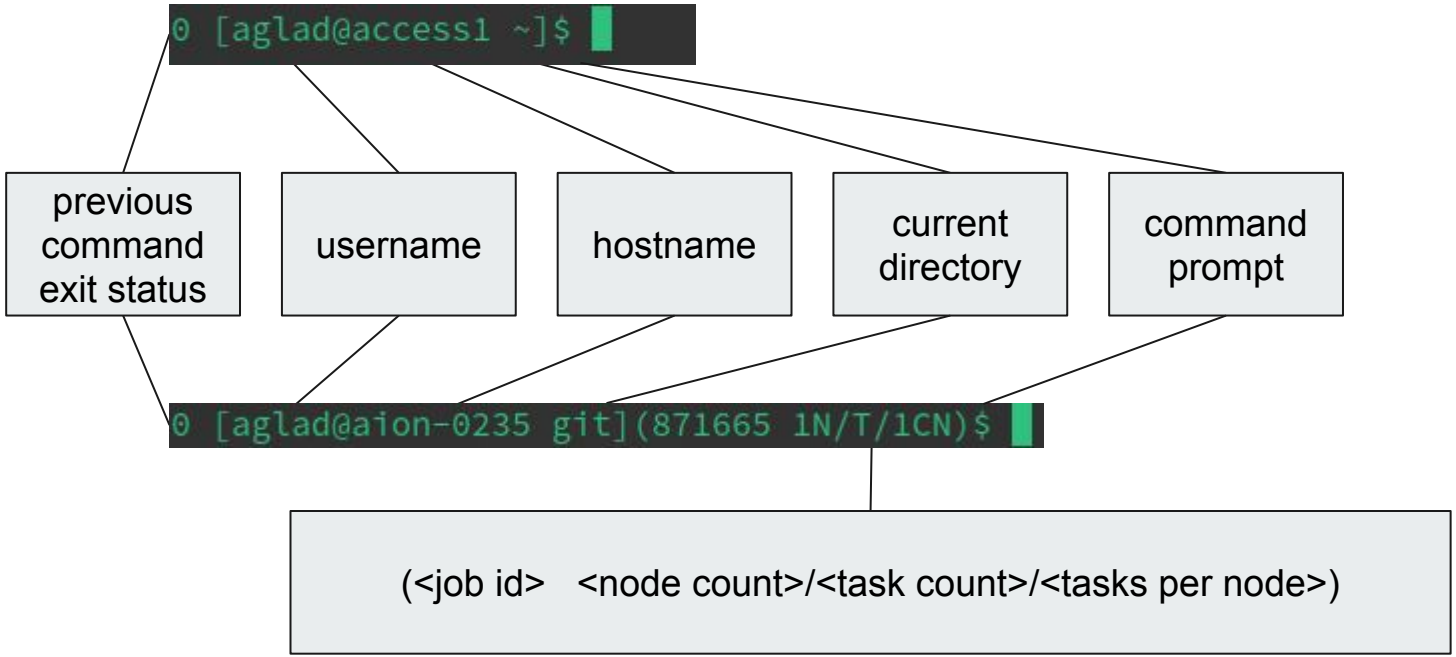
```
$ sbatch submission_script.sh
```

Interactive job

```
$ salloc --nodes=1 --ntasks-per-node=1 --cpus-per-task=7 --gpus-per-task=1 \
  --partition=gpu --qos=normal --time=0-4:00:00

salloc: Granted job allocation 4111109
(4111109 1N/1T/1CN)$ srun job_script
```

The interactive command prompt

The bash shell

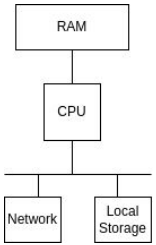


Accelerators

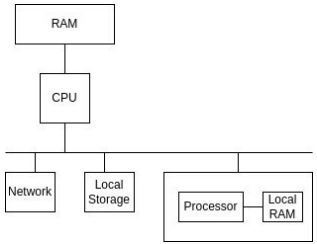
HPC is used to solve difficult computational problems.

- Traditional systems consists of a lot of conventional servers connected to a low latency and high throughput network and network storage.
- With the introduction of accelerators such as GPUs, there is a much wider variety of systems.

Conventional HPC node



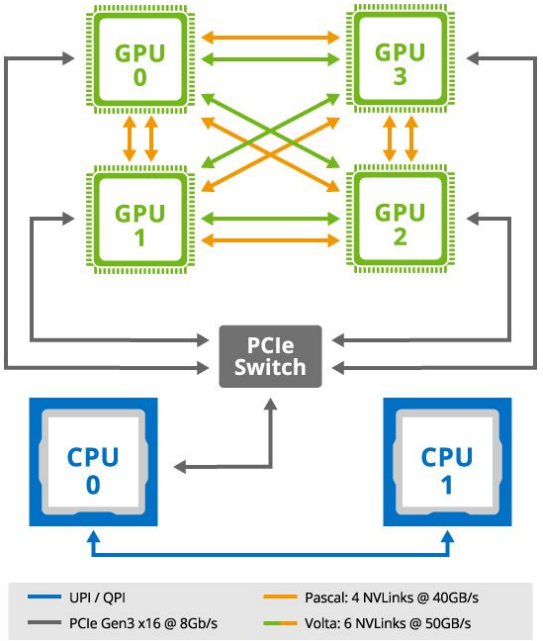
HPC node with accelerator



Accelerators

Accelerators in UL HPC

- 24 GPU nodes:
  - 4 V100-SXM2 per node
  - 32GB VRAM per GPU card
  - 768GB RAM per node

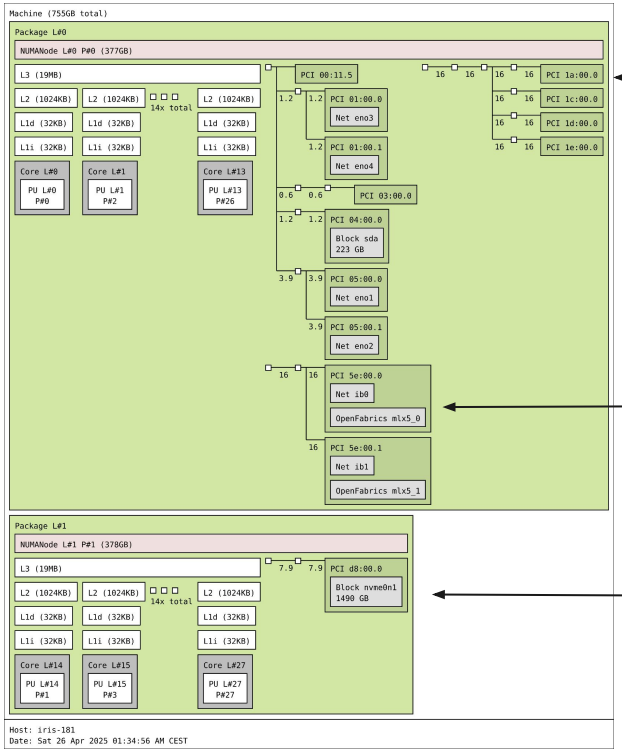


Accelerators

Accelerators in UL HPC

- 24 GPU nodes:
  - 4 V100-SXM2 per node
  - 32GB VRAM per GPU card
  - 768GB RAM per node
- Non-balanced architecture
  - 4 GPUs and 2 interconnect cards on socket 0
  - Storage on socket 1
- Inspect architecture:

```
$ hwloc-ls
```



Process and thread affinity

Asymmetric access

- Place processes close to their resources.
- Spread processes to access more resources.

Affinity information:

```
$ taskset --cpu-list --pid ${BASHPID}
pid 2468535's current affinity list: 0-127
```

Process and thread affinity

Process binding

- A set of cores where the process can float within a node
- Define with `srun`, and `mpirun` also supported.

Base method:

- `mask_cpu:<list>`: comma separated list of cores where tasks can be placed
- For instance for 2 tasks of 16 cores each:

```
mask_cpu:0xffff,0xffff0000
```

- Automatic mask generation options: `cores, ldoms, sockets`

Process and thread affinity

Process binding

```
$ srun --ntasks=2 --cpu-bind=mask_cpu:0xffff,0xffff0000 \
  bash -c ' \
    echo -n "task ${SLURM_PROCID} (node ${SLURM_NODEID}): "; \
    taskset --cpu-list --pid ${BASHPID} \
  ' | sort

task 0 (node 0): pid 2480321's current affinity list: 0-15
task 1 (node 0): pid 2480322's current affinity list: 16-31
```

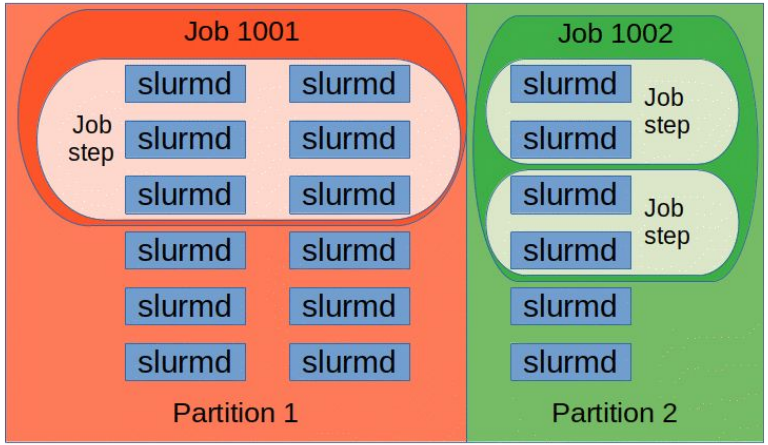
- `--cpu-bind` automatic options use `--cpus-per-task` to define a mask.
- Use full node allocations (--exclusive), as `--cpus-per-task` can interfere with the mask generation.

Job steps

- The scheduler executes jobs in steps.
- Every time you call srun (mpirun) a new job step starts.
- *Resource distribution within an allocation can only be performed in job steps.*

For instance, you cannot use `--cpu-bind` without calling srun (or equivalent options for mpirun).

The `sstat ` command prints detailed report for resource usage in each step.



Source: slurm.schedmd.com

Job steps

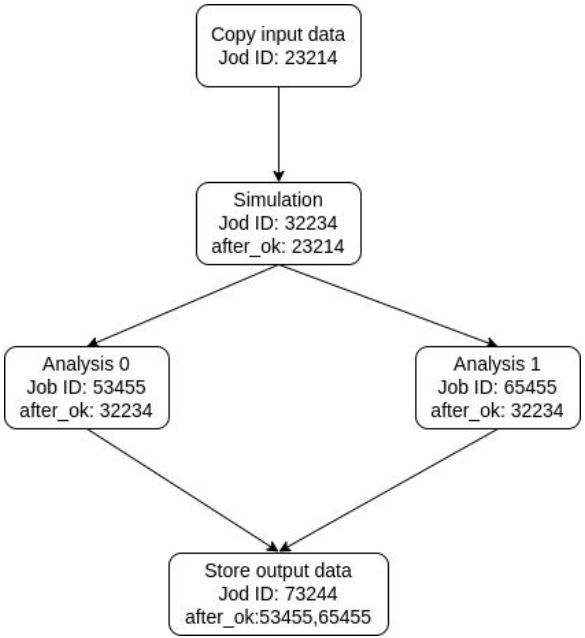
- Given that enough resources are available job steps can run in parallel.
- Remember to wait your job steps! The `srun` command is normally blocking.

```
#!/bin/bash --login

#SBATCH --job-name=parallel_steps_example
#SBATCH --output=%x-%j.out
#SBATCH --error=%x-%j.out
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=16
#SBATCH --time=0-04:00:00
#SBATCH --partition=batch
#SBATCH --qos=normal

srun --ntasks=4 job_case_0 &
srun --ntasks=4 job_case_1 & # Note that for ntasks: 4+4 <= 8

wait
```

Job dependencies

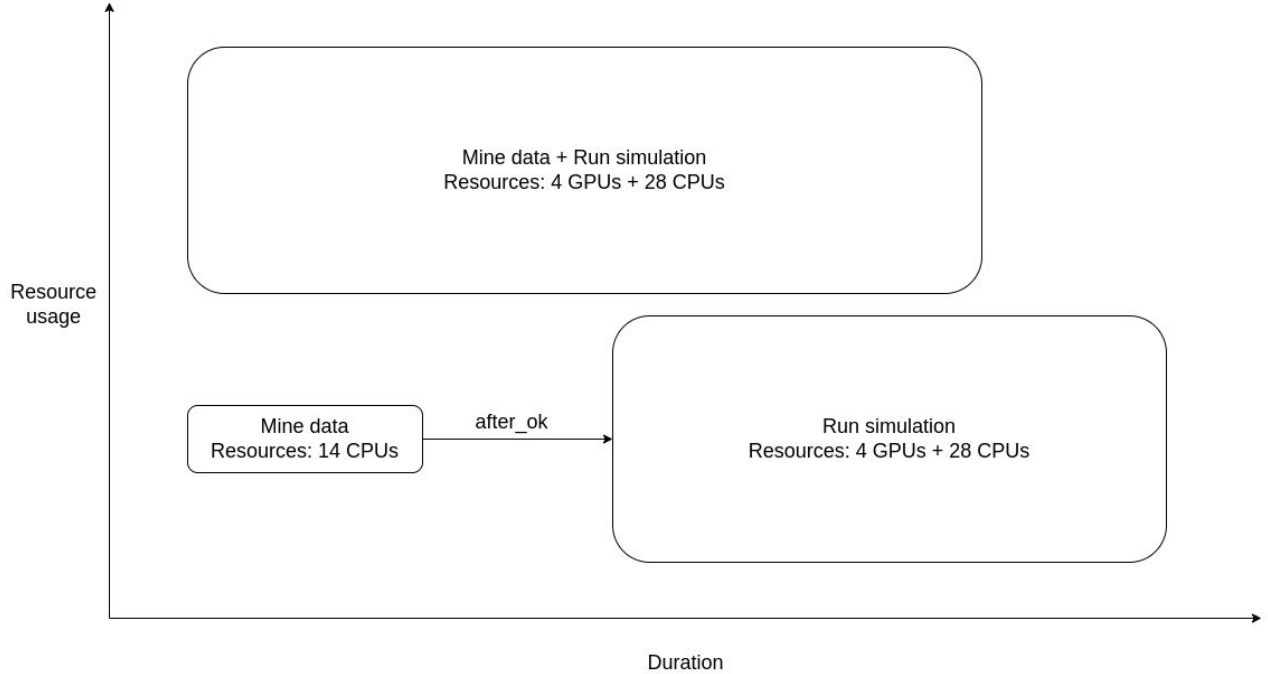
- Execute a queued job given that particular conditions are met.
- For instance when the input for a job depends from a previous job.

```
$ copy_input_data_id=$(sbatch --parsable copy_input_data.sh)
$ simulation_id=$(sbatch \
  --dependency=afterok:${copy_input_data_id} --parsable \
  simulation.sh)
$ analysis_0_id=$(sbatch \
  --dependency=afterok:${simulation_id} --parsable \
  analysis_0.sh)
$ analysis_1_id=$(sbatch \
  --dependency=afterok:${simulation_id} --parsable \
  analysis_1.sh)
$ analysis_0_id=$(sbatch \
  --dependency=afterok:${analysis_0_id},${analysis_1_id} \
  --parsable \
  store_output_data.sh)
```



Job dependencies

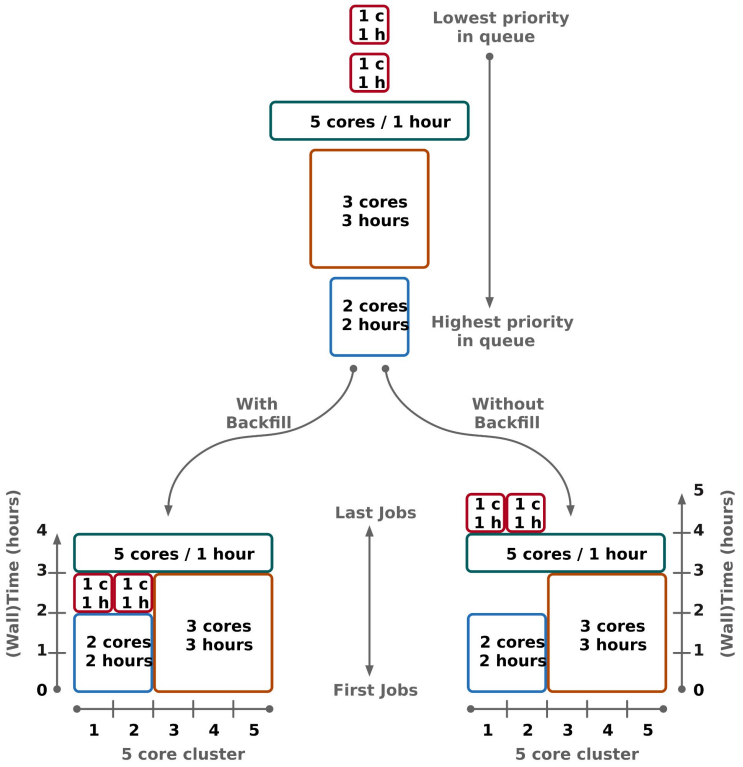
Why use dependencies?



Job dependencies

Why use dependencies?

- Backfilling is used in our cluster.
- Small jobs fill in gaps between larger jobs.
- Small job are scheduled earlier!
- Breaking a large job into steps means that overall your jobs will start earlier.



Source: docs.gcc.rug.nl

Software distribution

Modules, containers, and environments

Modules

Module search

$ module av the-program-you-want

On the right, we search with the keyword “Python”. The list of results contains various elements which are sorted by category (e.g. chem = Chemistry, lang = Programming languages, …)

We can see that two version of the Python language are available: 2.7.18 and 3.8.6. If no version is specified, the default choice (D) will be assumed, here 3.8.6.

```
[jschleich@iris-056 ~](3174323 1N/T/1CN)$ module av Python

-------------- /opt/apps/resif/iris-rhel8/2020b/broadwell/modules/all ----
 bio/TopHat/2.1.2-GCC-10.2.0-Python-2.7.18
 chem/spglib-python/1.16.0-foss-2020b
 chem/spglib-python/1.16.0-intel-2020b              (D)
 devel/flatbuffers-python/1.12-GCCcore-10.2.0
 devel/pkgconfig/1.5.1-GCCcore-10.2.0-python
 devel/protobuf-python/3.14.0-GCCcore-10.2.0
 lang/Python/2.7.18-GCCcore-10.2.0
 lang/Python/3.8.6-GCCcore-10.2.0                   (D)
 lang/SciPy-bundle/2020.11-foss-2020b-Python-2.7.18
 lib/Boost.Python/1.74.0-GCC-10.2.0

Where:
 D:  Default Module
```

Modules

Module list

List the currently loaded modules

```
$ module list
```

Module load

```
$ module load program-name
```

Module purge

Unload all loaded modules

```
$ module purge
```

```
0 [jschleich@iris-056 ~](3174323 1N/T/1CN)$ module list
No modules loaded
0 [jschleich@iris-056 ~](3174323 1N/T/1CN)$ module load lang/Python
0 [jschleich@iris-056 ~](3174323 1N/T/1CN)$ module list

Currently Loaded Modules:
  1) compiler/GCCcore/10.2.0              7) lang/Tcl/8.6.10-GCCcore-10.2.0
  2) lib/zlib/1.2.11-GCCcore-10.2.0       8) devel/SQLite/3.33.0-GCCcore-10.2.0
  3) tools/binutils/2.35-GCCcore-10.2.0   9) tools/XZ/5.2.5-GCCcore-10.2.0
  4) tools/bzip2/1.0.8-GCCcore-10.2.0    10) math/GMP/6.2.0-GCCcore-10.2.0
  5) devel/ncurses/6.2-GCCcore-10.2.0    11) lib/libffi/3.3-GCCcore-10.2.0
  6) lib/libreadline/8.0-GCCcore-10.2.0  12) lang/Python/3.8.6-GCCcore-10.2.0



0 [jschleich@iris-056 ~](3174323 1N/T/1CN)$ module purge
0 [jschleich@iris-056 ~](3174323 1N/T/1CN)$ module list
No modules loaded
```

Modules

Module avail

List the currently *available* modules

```
$ module avail
```

```
------------------------------------------- /opt/apps/easybuild/environment/modules -------------------------------------------
   env/deprecated/2020b  (S)    env/legacy/2020b    (S)      env/release/2023b (S,D)
   env/development/2024a (S)    env/release/default (S,L)    env/testing/2023b (S)

------------------------------------------------ /cvmfs/software.eessi.io/init/modules ------------------------------------------------
   EESSI/2023.06

------------------------------ /opt/apps/easybuild/systems/aion/rhel810-20250405/2023b/epyc/modules/all ------------------------------
   ai/PyTorch/2.3.0-foss-2023b                                  math/GMP/6.3.0-GCCcore-13.2.0
   bio/GROMACS/2024.4-foss-2023b-PLUMED-2.9.2                   math/Gurobi/11.0.0-GCCcore-13.2.0
   bio/GROMACS/2024.4-foss-2023b                    (D)         math/ISL/0.26-GCCcore-13.2.0
   bio/Seaborn/0.13.2-gfbf-2023b                                math/KaHIP/3.16-gompi-2023b
   cae/OpenFOAM/v2312-foss-2023b                                math/METIS/5.1.0-GCCcore-13.2.0
   cae/occt/7.8.0-GCCcore-13.2.0                                math/MPC/1.3.1-GCCcore-13.2.0
   chem/CP2K/2023.1-foss-2023b                                  math/MPFR/4.2.1-GCCcore-13.2.0
   chem/LAMMPS/29Aug2024-foss-2023b-kokkos                      math/MUMPS/5.6.1-foss-2023b-metis
   chem/Libint/2.7.2-GCC-13.2.0-lmax-6-cp2k                     math/Osi/0.108.9-GCC-13.2.0
   chem/MDI/1.4.29-gompi-2023b                                  math/ParMETIS/4.0.3-gompi-2023b
   chem/PLUMED/2.9.2-foss-2023b                                 math/Qhull/2020.2-GCCcore-13.2.0
   chem/kim-api/2.3.0-GCC-13.2.0                                math/SCOTCH/7.0.4-gompi-2023b
   chem/libxc/6.2.2-GCC-13.2.0                                  math/ScaFaCoS/1.0.4-foss-2023b
   compiler/GCC/13.2.0                                          math/Voro++/0.4.6-GCCcore-13.2.0
   compiler/GCCcore/13.2.0                                      math/gmpy2/2.1.5-GCC-13.2.0
--More--
```

Modules

Sticky meta-modules

Modules that change the set of available modules.

Default set of modules:

```
------------------------------------------ /opt/apps/easybuild/environment/modules -------------------
   env/deprecated/2019b  (S)    env/development/2023 (S,D)    env/release/default (S,L)
   env/development/2023b (S)    env/legacy/2019b     (S)      env/release/2020b   (S,D)

------------------------------------------ /cvmfs/software.eessi.io/init/modules --------------------
   EESSI/2023.06

------------------------------------------ /opt/apps/resif/aion/2020b/epyc/modules/all ---------------
   bio/ABySS/2.2.5-foss-2020b                                lib/libtirpc/1.3.1-GCCcore-10.2.0
   bio/BEDTools/2.30.0-GCC-10.2.0                            lib/libtool/2.4.6-GCCcore-10.2.0
   bio/BLAST+/2.11.0-gompi-2020b                             lib/libunwind/1.4.0-GCCcore-10.2.0
   bio/BWA/0.7.17-GCC-10.2.0                                 lib/libvorbis/1.3.7-GCCcore-10.2.0
   bio/BamTools/2.5.1-GCC-10.2.0                             lib/libwebp/1.1.0-GCCcore-10.2.0
   bio/BioPerl/1.7.8-GCCcore-10.2.0                          lib/libxml2/2.9.10-GCCcore-10.2.0
   bio/Bowtie2/2.4.2-GCC-10.2.0                              lib/libyaml/0.2.5-GCCcore-10.2.0
   bio/FastQC/0.11.9-Java-11                                 lib/lz4/1.9.2-GCCcore-10.2.0
   bio/GROMACS/2021-foss-2020b                               lib/nettle/3.6-GCCcore-10.2.0
   bio/HTSlib/1.12-GCC-10.2.0                                lib/pybind11/2.6.0-GCCcore-10.2.0
   bio/SAMtools/1.12-GCC-10.2.0                              lib/scikit-build/0.11.1-foss-2020b
   bio/TopHat/2.1.2-GCC-10.2.0-Python-2.7.18                 lib/snappy/1.1.8-GCCcore-10.2.0
   cae/ABAQUS/2021-hotfix-2207                               lib/tbb/2020.3-GCCcore-10.2.0
   cae/ABAQUS/2022                                (D)        lib/tqdm/4.56.2-GCCcore-10.2.0
   cae/Neper/4.6.0-foss-2020b                                lib/zlib/1.2.11-GCCcore-10.2.0
```

`module load env/release/2023b`

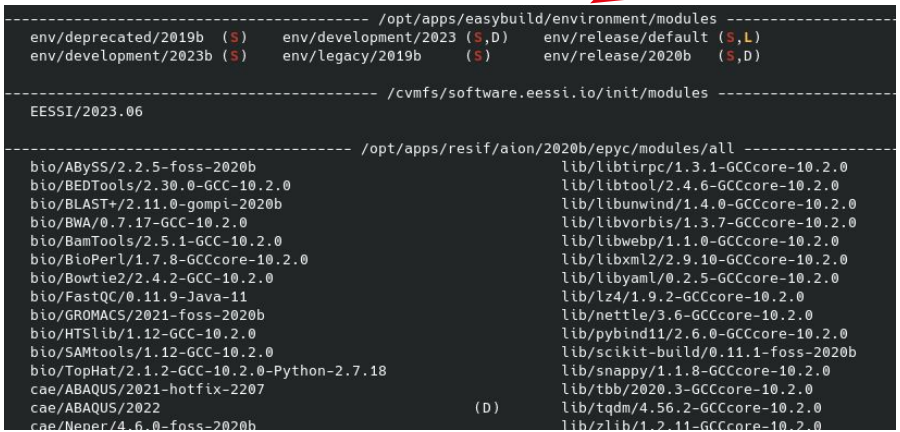
```
------------------------------------------ /opt/apps/easybuild/environment/modules -------------------
   env/deprecated/2019b  (S)      env/development/2023 (S,D)    env/release/default (S)
   env/development/2023b (S,L)    env/legacy/2019b     (S)      env/release/2020b   (S,D)

------------------------------------------ /cvmfs/software.eessi.io/init/modules --------------------
   EESSI/2023.06

------------------------- /opt/apps/easybuild/systems/aion/rhel810-20250216/2023b/epyc/modules/all ----
   ai/PyTorch/2.3.0-foss-2023b                               math/GMP-ECM/7.0.5-GCCcore-13.2.0
   bio/GROMACS/2024.4-foss-2023b-PLUMED-2.9.2                math/GMP/6.3.0-GCCcore-13.2.0
   bio/GROMACS/2024.4-foss-2023b                  (D)        math/Givaro/4.2.0-GCCcore-13.2.0
   cae/OpenFOAM/v2312-foss-2023b                             math/Gurobi/11.0.0-GCCcore-13.2.0
   chem/CP2K/2023.1-foss-2023b                               math/IML/1.0.5-gfbf-2023b
   chem/LAMMPS/29Aug2024-foss-2023b-kokkos                   math/ISL/0.26-GCCcore-13.2.0
   chem/Libint/2.7.2-GCC-13.2.0-lmax-6-cp2k                  math/KaHIP/3.16-gompi-2023b
   chem/MDI/1.4.29-gompi-2023b                               math/LinBox/1.7.0-gfbf-2023b
   chem/PLUMED/2.9.2-foss-2023b                              math/METIS/5.1.0-GCCcore-13.2.0
   chem/kim-api/2.3.0-GCC-13.2.0                             math/MPC/1.3.1-GCCcore-13.2.0
   chem/libxc/6.2.2-GCC-13.2.0                               math/MPFI/1.5.4-GCCcore-13.2.0
   compiler/GCC/13.2.0                                       math/MPFR/4.2.1-GCCcore-13.2.0
   compiler/GCCcore/13.2.0                                   math/MUMPS/5.6.1-foss-2023b-metis
   compiler/Go/1.22.1                                        math/NTL/11.5.1-GCC-13.2.0
   compiler/LLVM/16.0.6-GCCcore-13.2.0                       math/Normaliz/3.10.3-gfbf-2023b
```

Modules

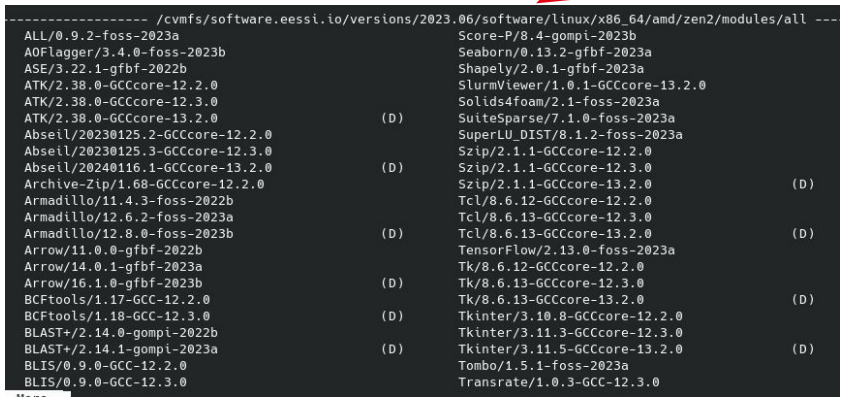
Sticky meta-modules

Modules that change the set of available modules.

Default set of modules:



`module load EESSI/2023.06`



Modules

Sticky meta-modules

Modules that change the set of available modules

- Local modules:

  `$ module load env/X/Y`

  where `X = development, release, deprecated`
- EESSI modules:

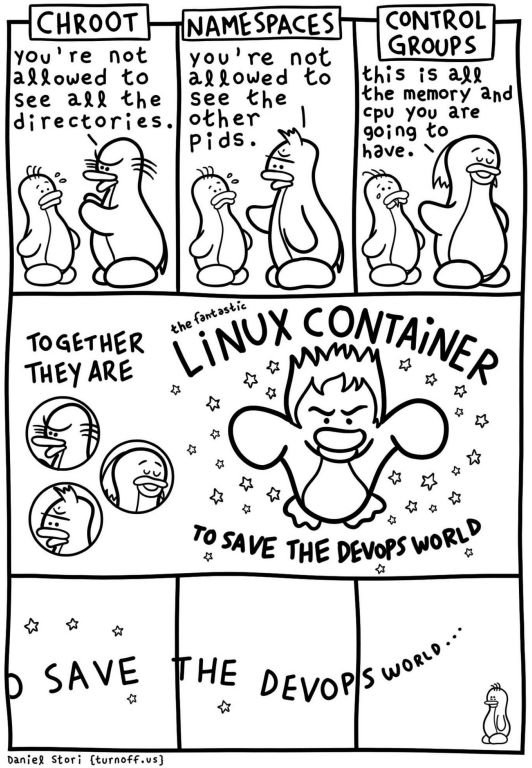
  `$ module load EESSI/2023.06`
- Local modules more optimized, EESSI more standardized across systems
- Purge sticky modules with `--force` flag:

  `$ module --force purge`

Containers

A wide and loose set of technologies used to allow software applications to run in isolated user spaces.

- **Chroot:** system operation changing the apparent root directory.
- **Namespaces:** Linux kernel feature that allows mapping between resources.
- **Control groups (cgroups):** Linux kernel feature that limits and isolates the resource usage.



Containers

Containers allow the creation of isolated and reproducible environments.

Containers can:

- Install and execute programs in user space.
- Modify the user environment.
- Override component of the Operating System such as C standard library (libc).

Containers cannot:

- Modify components of the kernel such as drivers.

Containers

HPC container tools

- Apptainer
- Singularity

Provided functionality

- Describe the construction of a container into a single file.
- Expand the container in the file system.
- Repackage the container.
- Execute commands inside the container.

```
Bootstrap: docker
From: debian:{{ VERSION }}
Stage: build

%arguments
   VERSION=stable-20250407-slim

%environment
   export LC_ALL=C
   export PATH=/usr/games:${PATH}

%post -c /bin/bash
   export DEBIAN_FRONTEND=noninteractive
   apt-get --assume-yes update
   apt-get --assume-yes --no-install-recommends install lolcat cowsay

%runscript
   #!/bin/bash

   echo $@ | cowsay | lolcat

%test
   cowsay --version
   lolcat --version

%labels
   Author hpc-team@uni.lu
   Version v0.0.1

%help
   This is a demo container used to illustrate a def file that uses
all
   supported sections.
```

Containers

HPC container tools

- Apptainer
- Singularity

Provided functionality

- Describe the construction of a container into a single file.
- Expand the container in the file system.
- Repackage the container.
- Execute commands inside the container.

```
$ apptainer build --sandbox lolcow
lolcow.sif

INFO: Starting build...
INFO: Verifying bootstrap image lolcow.sif
INFO: Extracting local image...
INFO: Creating sandbox directory...
INFO: Build complete: lolcow

$ apptainer run lolcow.sif 'Hello world!'

 _______________
< Hello world! >
 ---------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

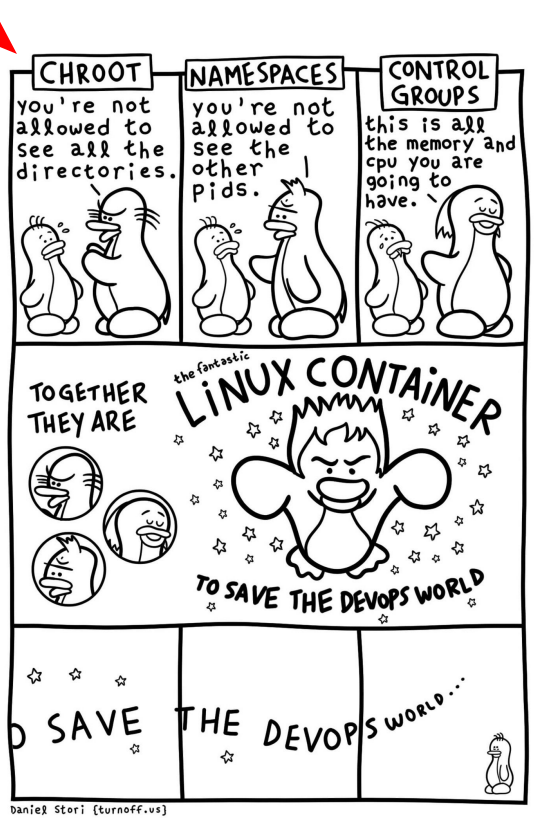
Containers

Minimal example

An isolate chroot system

1. Create the root of the chroot file system.
2. Copy the required executables.
3. Copy the linker.
4. Copy the libraries required by the executables.
5. Create a text file to test the executables.

Change to the new system and test!



Containers

```
$ mkdir --parents ${HOME}/jail/{bin,lib,lib64,home/myusername}
$ for binary in /bin/bash /bin/ls /bin/cat; do \
    cp ${binary} ${HOME}/jail/bin/ \
  done; unset binary
$ for linker in /lib64/ld-linux-x86-64.so*; do \
    cp ${linker} ${HOME}/jail/lib64/ \
  done; unset linker
$ while IFS="" read -r library; do \
    cp ${library} ${HOME}/jail/lib/ \
  done < <(ldd /bin/bash /bin/ls /bin/cat \
    | grep -E '=>' | awk 'BEGIN {FS="(=>)|( +)"} {print $4}' \
    | sort | uniq); unset library
$ echo 'Welcome to chroot jail!' > ${HOME}/jail/home/myusername/hello.txt
```

Change to the new system and test!

Containers

Change to the new system and test!

```
$ sudo chroot ${HOME}/jail /bin/bash
bash-5.2# cat ${HOME}/jail/home/myusername/hello.txt

Welcome to chroot jail!
```

Containers

Replicate the structure of the linux file system in a directory. For instance:

- `/bin`: Directory containing executable programs.
- `/lib`: Shared libraries necessary to boot the system.

Resources are mount in the file system.

- Network
- User directories

```
$ ls -l ${HOME}/jail
total 0
drwxr-sr-x 1 gkaf gkaf  0 Apr 28 01:08 bin
drwxr-sr-x 1 gkaf gkaf 20 Apr 28 01:08 home
drwxr-sr-x 1 gkaf gkaf  0 Apr 28 01:08 lib
drwxr-sr-x 1 gkaf gkaf  0 Apr 28 01:08 lib64
```

Environments

Similarly to containers environments allow the creation of isolated and reproducible program installations.

Environments can:

- Install and execute programs in user space.
- Modify the user environment.

Environments cannot:

- *Override component of the Operating System such as C standard library (libc).*
- Modify components of the kernel such as drivers.

Environments

Similarly to containers environments replicate the structure of the linux file system in a directory. For instance:

- `/bin`: Directory containing executable programs.
- `/lib`: Shared libraries necessary to boot the system.

Unlike containers, environments do not manage any resources.

```
$ python -m venv ${HOME}/environments/mkdocs
$ ls -l ${HOME}/environments/mkdocs
total 8
drwxr-xr-x 1 gkaf gkaf 338 Apr 23 14:08 bin
drwxr-xr-x 1 gkaf gkaf  20 Apr 23 14:07 include
drwxr-xr-x 1 gkaf gkaf  20 Apr 23 14:07 lib
lrwxrwxrwx 1 gkaf gkaf   3 Apr 23 14:07 lib64 -> lib
-rw-r--r-- 1 gkaf gkaf 255 Apr 23 14:07 pyvenv.cfg

$ source ${HOME}/environments/mkdocs/bin/activate
$ pip install mkdocs

...
$ mkdocs serve

...
$ deactivate
```

Partnership with LuxProvide

Access to the tier-1 system of Luxembourg for university researcher

UNIVERSITÉ DU LUXEMBOURG

UL HPC / LuxProvide

University Account for Meluxuni HPC

Availability for GPU, CPU, FPGA and big memory CPU compute nodes

Please contact: matteo.barborini@uni.lu, julien.schleich@uni.lu

Wide scale infrastructure for computing research

Contribution of regular nodes, accelerators, and special resource types

UNIVERSITÉ DU LUXEMBOURG

Wide scale infrastructure for computing research

Grid'5000

- Accessible through institutions across France, Belgium, and Luxembourg
- Large amount and variety of resources:
  - 15000 cores, 800 compute-nodes
  - systems with PMEM, GPU, SSD, NVMe, Ethernet, Infiniband, and Omni-Path
  - support for open science and reproducible research

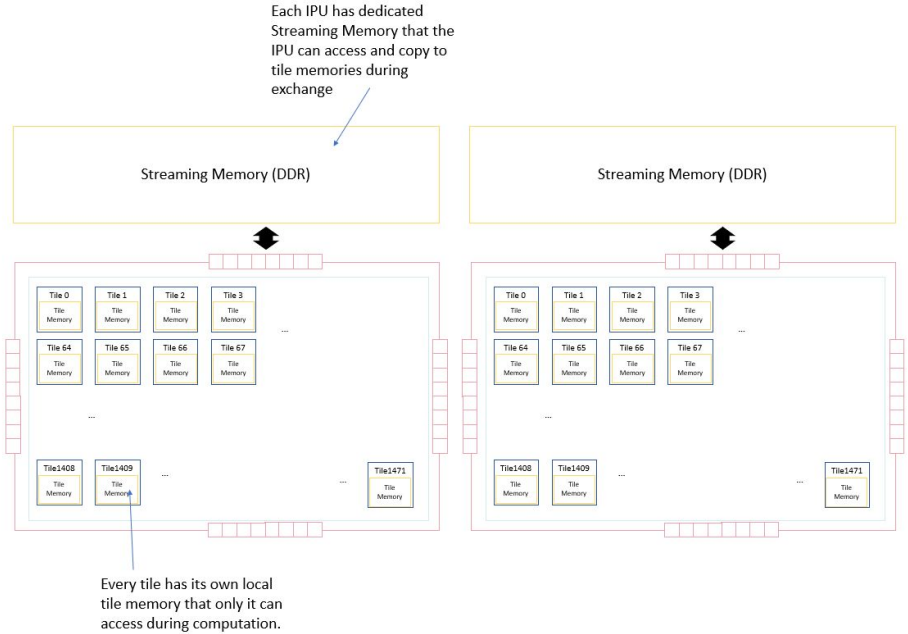
Slices-RI

- EU wide Infrastructure for Large-scale Experimental Computer Science
- HPC, edge computing, and quantum computing
- Grid'5000 to merge into Slices

Wide scale infrastructure for computing research

Contribution of UL HPC to Grid'5000/Slices-RI

- Basic Infrastructure service:
  - 48 CPU nodes
  - 28 cores and 128GB RAM per node
- Machine Learning service:
  - 7 small GPU servers with 4 AMD MI210 GPUs with 64GB VRAM
  - 1 large GPU server with 8 AMD MI300X OAM with 192GB GPUs VRAM
  - 1 **Bow Pod16** system with a total of 16 **Bow IPUs** (MPI on a chip by Graphcore)



Thank you!

Any questions?

Thank you!

Next?

Programing Parallel Applications on HPC systems!

Programming Parallel Applications on HPC systems

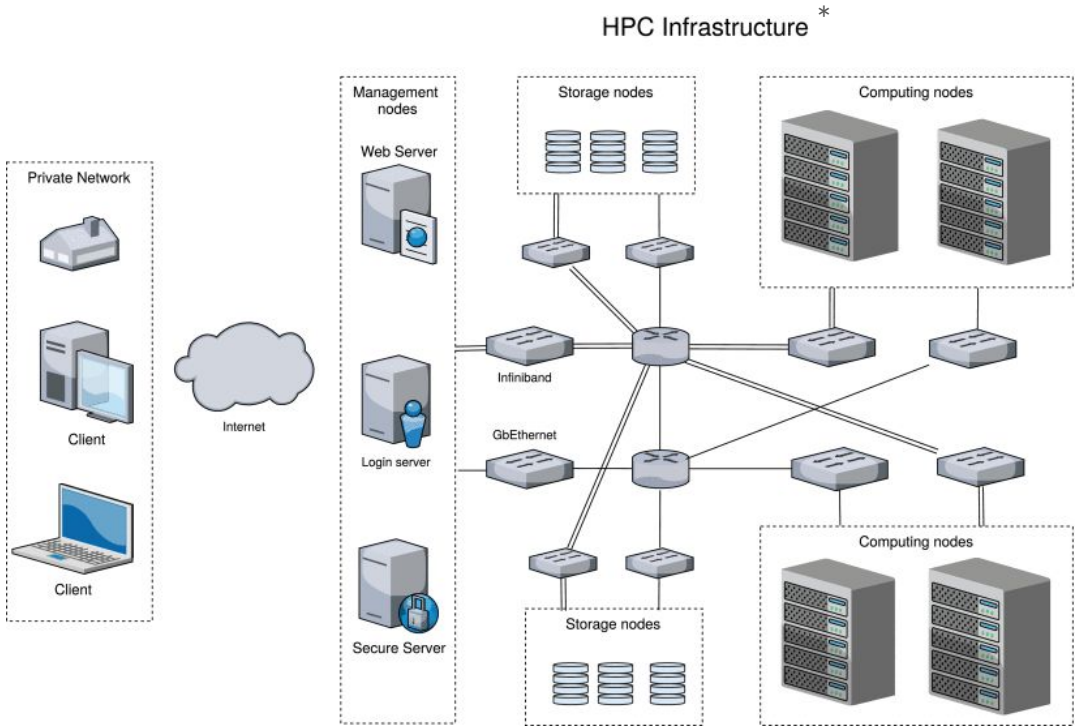
An overview of modern programming frameworks and techniques used in HPC systems.

Outline

- Main Hardware characteristics for efficient parallelization
- Parallel computing on HPC
  - Basic concepts
  - Message Passing Interface (MPI)
  - Shared Memory Parallelization (OpenMP)
- Compilers, libraries and linking
  - MPI, openMP
  - Parallelization of linear algebra libraries (MKL,OBLAS)
- Running parallel applications
  - Environment Variables
  - Partitioning of threads
  - Slurm scheduler Examples
- Performance Testing
  - Weak Scaling
  - Strong Scaling

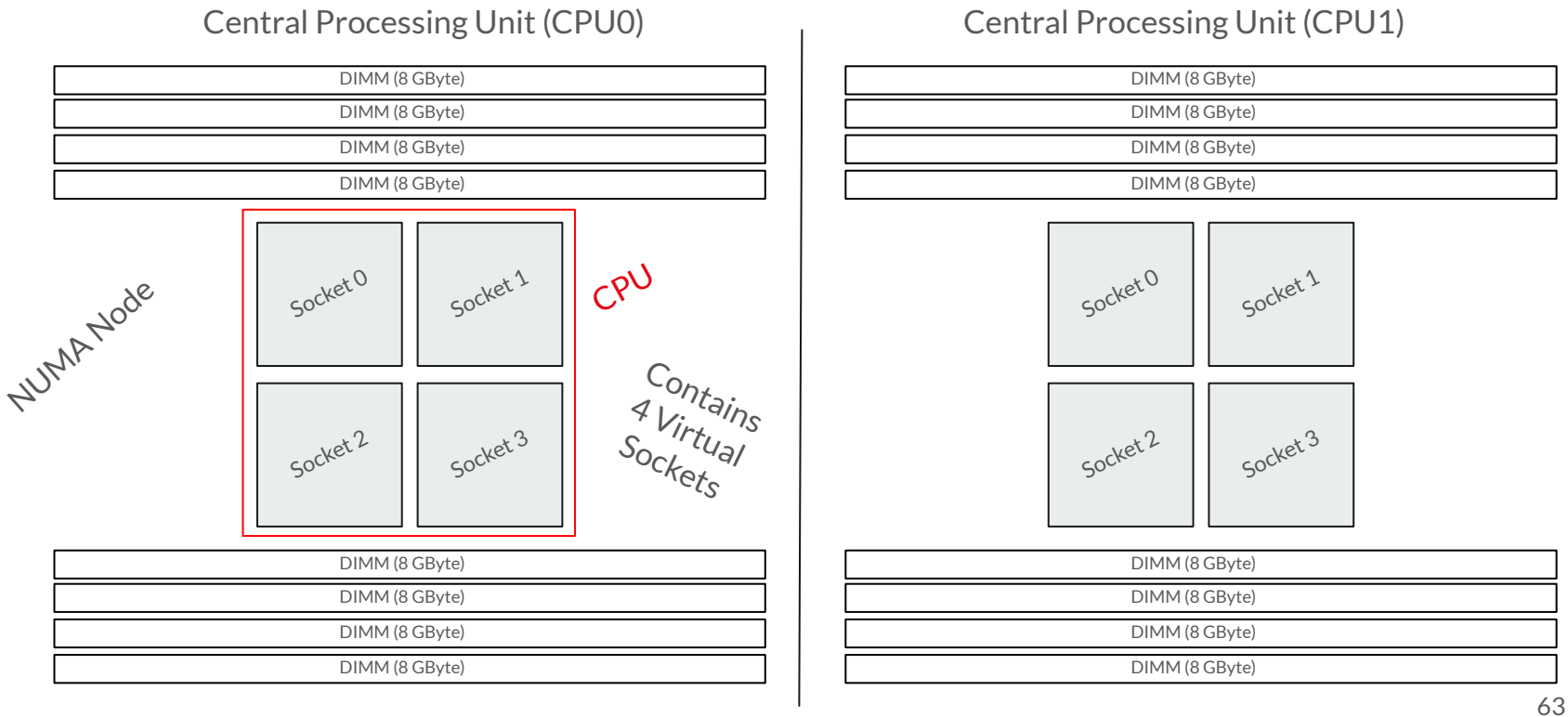
Hardware knowledge

HPC facility

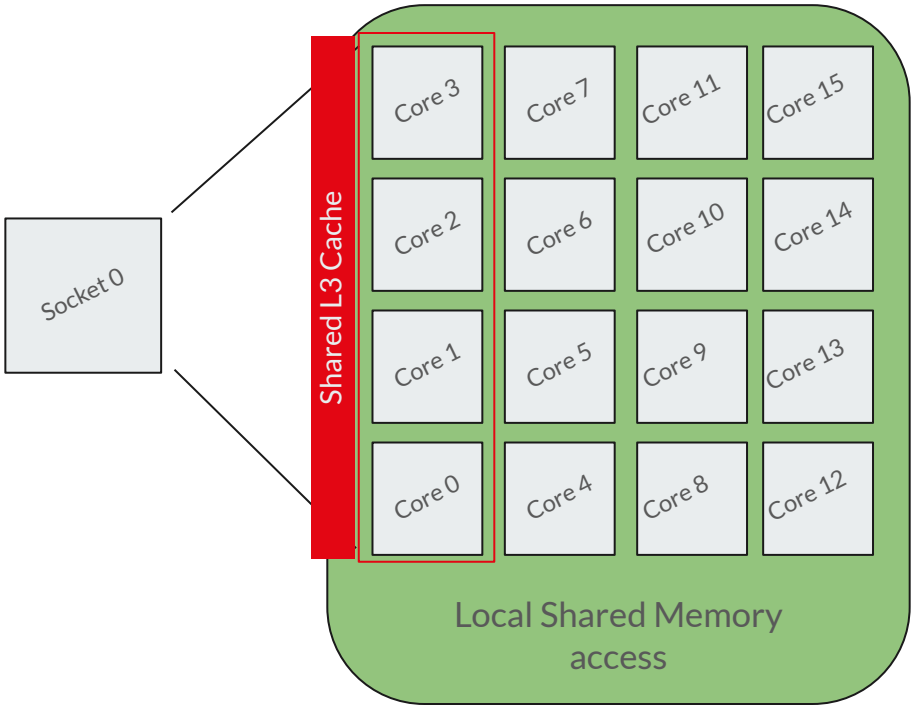


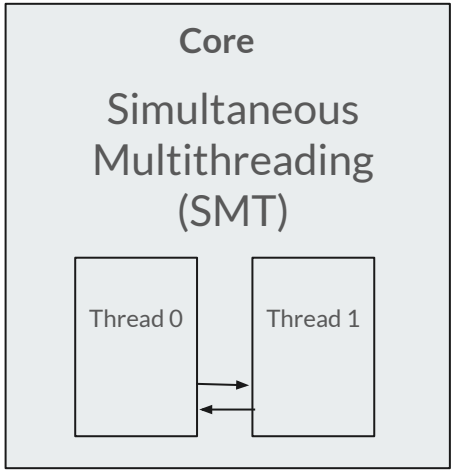
* F. Reghenzani, G. Massari and W. Fornaciari, "Timing Predictability in High-Performance Computing With Probabilistic Real-Time," in *IEEE Access*, vol. 8, pp. 208566-208582, 2020, doi: 10.1109/ACCESS.2020.3038559.

Compute Node and Central Processing Units



Single CPU, 4 virtual sockets, 64 Cores





Can take advantage of Idleness between parallel calculations.

Parallelization protocols

Basic protocols for parallel computing

MPI (Message Passing Interface) is a **standardized and portable** message-passing system designed to allow **processes to communicate with each other** in a parallel computing environment — especially across **distributed memory systems** like clusters or supercomputers.

- **Programming Model**: Distributed memory
- **Execution**: Independent processes with separate memory spaces
- **Ease of Use**: More complex (requires explicit message passing)
- **Communication**: Explicit (send and receive messages between processes)
- **Scalability**: Scales well across multiple nodes/machines
- **Best For**: Large-scale distributed computing

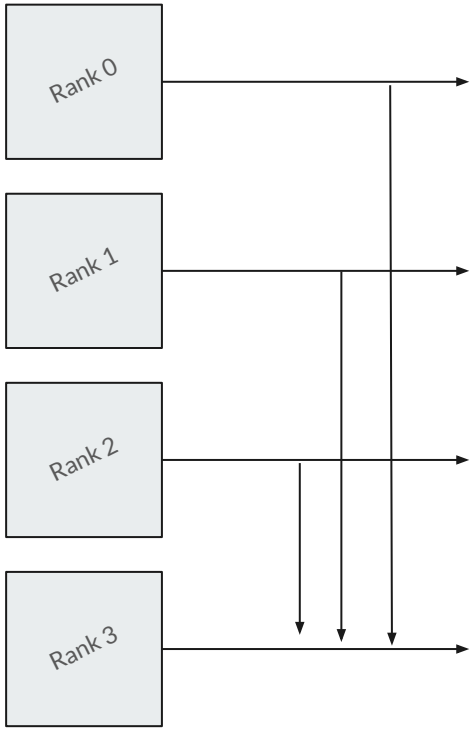
OpenMP (Open Multi-Processing) is an API (Application Programming Interface) that supports **multi-platform shared-memory parallel programming** in **C, C++, and Fortran**.

- **Programming Model**: Shared memory
- **Execution**: Threads run in parallel on shared memory space
- **Ease of Use**: Easier to implement (uses compiler directives in C/C++/Fortran)
- **Communication**: Implicit (threads share variables by default)
- **Scalability**: Limited by number of cores on a single machine
- **Best For**: Multi-core shared-memory systems

OpenMP

Massage Passing Interface

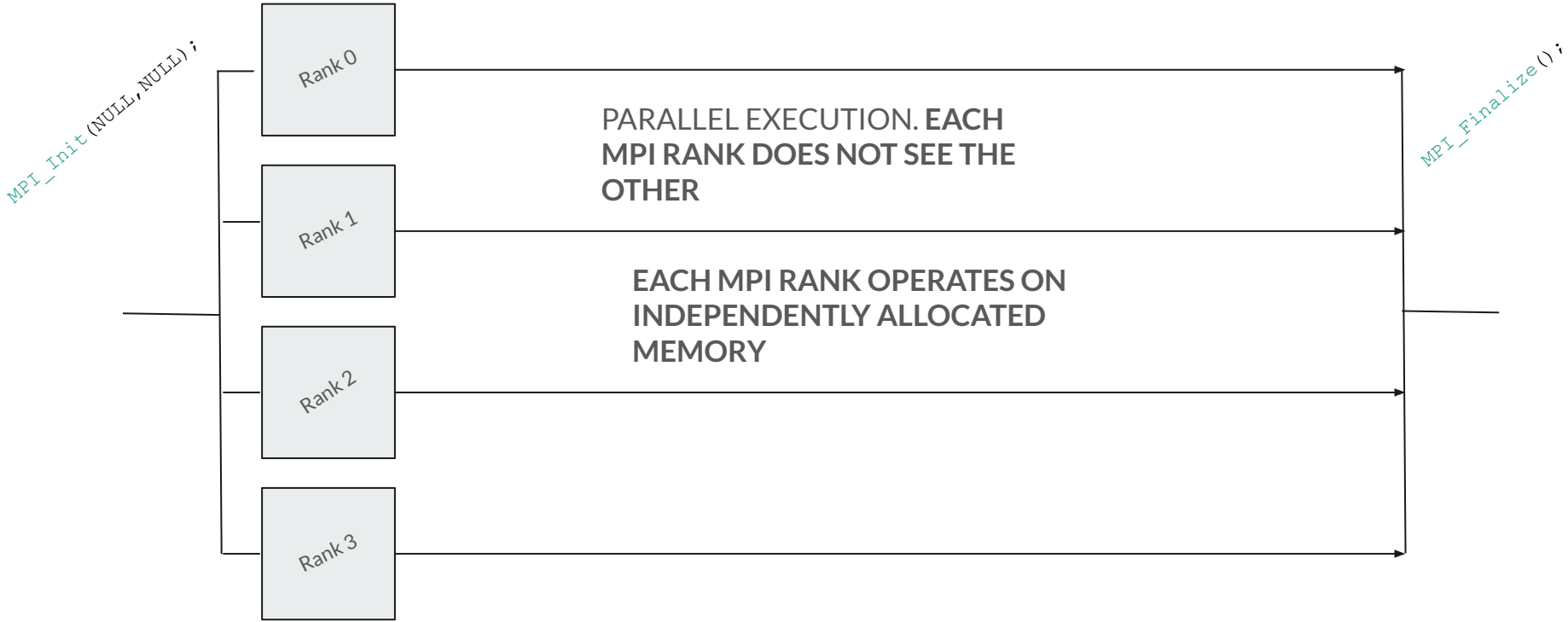
Message Passing Interface (MPI)



MPI (Message Passing Interface) is a **standardized and portable** message-passing system designed to allow **processes to communicate with each other** in a parallel computing environment — especially across **distributed memory systems** like clusters or supercomputers.

- **Programming Model**: Distributed memory
- **Execution**: Independent processes with separate memory spaces
- **Ease of Use**: More complex (requires explicit message passing)
- **Communication**: Explicit (send and receive messages between processes)
- **Scalability**: Scales well across multiple nodes/machines
- **Best For**: Large-scale distributed computing

Message Passing Interface (MPI)



Initialization and finalization of MPI

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

int main(int argc, char** argv[]) {
       // Initialize the MPI environment
       MPI_Init(NULL,NULL);

       // Get the number of processes
       int num_of_ranks;
       MPI_Comm_size(MPI_COMM_WORLD, &num_of_ranks);

       // Get the rank of the process
       int mpi_rank;
       MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

       // Print a hello message from each process
       printf("Hello from process %d of %d\n", mpi_rank, num_of_ranks);

        [ . . . ]

       // Finalize the MPI environment
       MPI_Finalize();

       return 0;
}
```

Initialization and finalization of MPI Environment

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

int main(int argc, char** argv[]) {
    // Initialize the MPI environment
    MPI_Init(NULL,NULL);

    // Get the number of processes
    int num_of_ranks;
    MPI_Comm_size(MPI_COMM_WORLD, &num_of_ranks);

    // Get the rank of the process
    int mpi_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

    // Print a hello message from each process
    printf("Hello from process %d of %d\n", mpi_rank, num_of_ranks);

    [ . . . ]

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

Linking of the MPI headers

Initialization of MPI environment (The number of MPI ranks and the position is set by the scheduler such as SLURM)

Finalization of the MPI environment

Initialization and finalization of MPI Environment

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

int main(int argc, char** argv[]) {
    // Initialize the MPI environment
    MPI_Init(NULL,NULL);

    // Get the number of processes
    int num_of_ranks;
    MPI_Comm_size(MPI_COMM_WORLD, &num_of_ranks);

    // Get the rank of the process
    int mpi_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

    // Print a hello message from each process
    printf("Hello from process %d of %d\n",mpi_rank, num_of_ranks);

    [ . . . ]

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

Linking of the MPI headers

Initialization of MPI environment (The number of MPI ranks and the position is set by the scheduler such as SLURM)

Fundamentally, all this is executed in parallel by each MPI rank….

Finalization of the MPI environment

Initialization and finalization of MPI Environment

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

int main(int argc, char** argv[]) {
    // Initialize the MPI environment
    MPI_Init(NULL,NULL);

    // Get the number of processes
    int num_of_ranks;
    MPI_Comm_size(MPI_COMM_WORLD, &num_of_ranks);

    // Get the rank of the process
    int mpi_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

    // Print a hello message from each process
    printf("Hello from process %d of %d\n", mpi_rank, num_of_ranks);

     [ . . . ]

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

Linking of the MPI headers

Initialization of MPI environment (The number of MPI ranks and the position is set by the scheduler such as SLURM)

Interrogating the system (MPI_COMM_WORLD) on the number of ranks

Interrogating the system (MPI_COMM_WORLD) on the name of a specific rank.

Finalization of the MPI environment

Initialization and finalization of MPI Environment

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

int main(int argc, char** argv[]) {
    // Initialize the MPI environment
    MPI_Init(NULL,NULL);

    // Get the number of processes
    int num_of_ranks;
    MPI_Comm_size(MPI_COMM_WORLD, &num_of_ranks);

    // Get the rank of the process
    int mpi_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

    // Print a hello message from each process
    printf("Hello from process %d of %d\n", mpi_rank, num_of_ranks);

    [ . . . ]

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

Linking of the MPI headers

Initialization of MPI environment (The number of MPI ranks and the position is set by the scheduler such as SLURM)

The result is that eact MPI RANK retrieves the total number of RANKS, and its name….and it prints the result.

Finalization of the MPI environment

Compilation with MPI

On a laptop machine in order to use MPI it is essential to install for example Intel MPI or OpenMPI libraries

Latest version of the OpenMPI package can be found in https://www.open-mpi.org/

After the installation it is essential to export the bin and lib directories

```
export PATH=$HOME/Software/OpenMPI-5.0.6/bin:$PATH
export LD_LIBRARY_PATH=$HOME/Software/OpenMPI-5.0.6/lib:$LD_LIBRARY_PATH
```

On Meluxina or the ULHPC clusters the MPI libraries are usually loaded together with the toolchains

Meluxina:	`module load foss`	`module load intel`
ULHPC:	`module load toolchain/foss`	`module load toolchain/intel`

Compilation with MPI

By loading the MPI libraries different wrappers will be made available

```
mpicc, mpif90, mpic++, mpiifort, mpicxx …
```

To check the characteristics of each wrapper (compiler, options, ecc.) it is sufficient to run the command

```
mpicc --show, mpif90 --show, mpic++ --show, mpicxx
--show
```

To compile a simple C application such as the one defined in the previous slides we run:

```
mpicc -o myapplication.exe myapplication.cpp -I$HOME/Software/OpenMPI-5.0.6/include
```

or for Fortran 90 and higher:

```
mpif90 -o myapplication.exe myapplication.f90 -I$HOME/Software/OpenMPI-5.0.6/include
```

Running the simple C application

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

int main(int argc, char** argv[]) {
	// Initialize the MPI environment
	MPI_Init(NULL,NULL);

	// Get the number of processes
	int num_of_ranks;
	MPI_Comm_size(MPI_COMM_WORLD, &num_of_ranks);

	// Get the rank of the process
	int mpi_rank;
	MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

	// Print a hello message from each process
	printf("Hello from process %d of %d\n", mpi_rank,
num_of_ranks);

	// Finalize the MPI environment
	MPI_Finalize();

	return 0;
}
```

Running application on a laptop

```
mpiexec -n ${num_of_ranks}
    application.exe
mpirun -n ${num_of_ranks}
    application.exe
```

Running application on a cluster such as MeluXina or ULHPC we use the command of the SLURM manager

```
srun -n ${num_of_ranks} application.exe
```

Assuming `${num_of_ranks} = 4` we have as are result of running the application:

```
Hello from process 2 of 4
Hello from process 1 of 4
Hello from process 0 of 4
Hello from process 3 of 4
```

Running the simple C application

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

int main(int argc, char** argv[]) {
        // Initialize the MPI environment
        MPI_Init(NULL,NULL)

        // Get the number o
        int num_of_ranks;
        MPI_Comm_size(MPI_C

        // Get the rank of
        int mpi_rank;
        MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

        // Print a hello message from each process
        printf("Hello from process %d of %d\n", rank, size);

        // Finalize the MPI environment
        MPI_Finalize();

        return 0;
}
```

THEY ARE ASYNCHRONOUS

Running application on a laptop

```
mpiexec -n ${num_of_ranks}
     application.exe
mpirun -n ${num_of_ranks}
                .exe
```

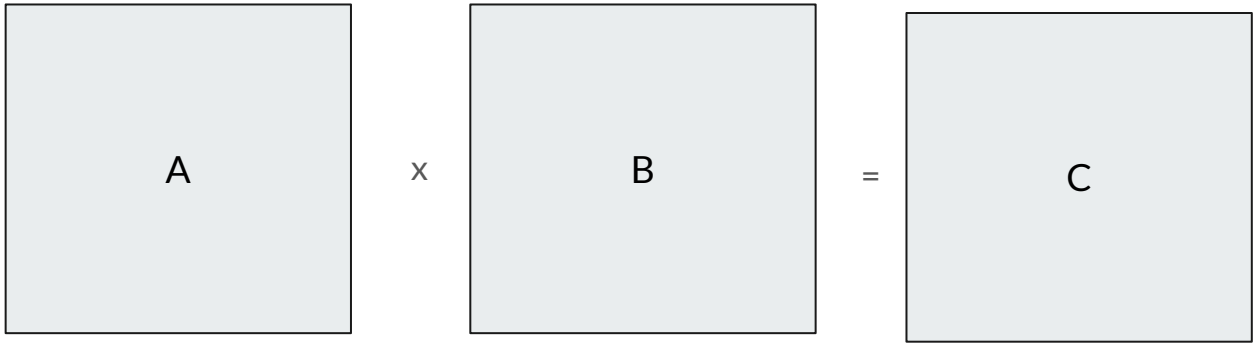
er such as MeluXina or
of the SLURM manager

```
srun -n ${num_of_ranks} application.exe
```

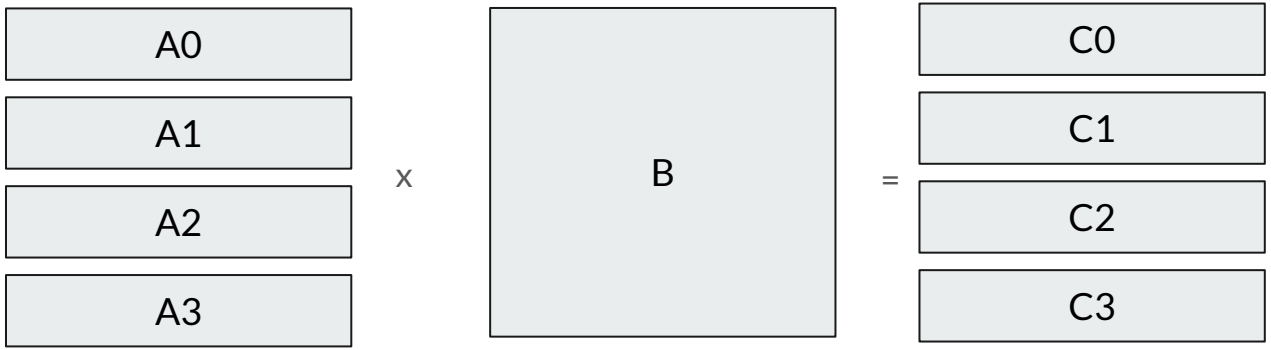
Assuming `${num_of_ranks} = 4` we have as are result of running the application:

```
Hello from process 2 of 4
Hello from process 1 of 4
Hello from process 0 of 4
Hello from process 3 of 4
```

Illustrative example (Matrix multiplication)

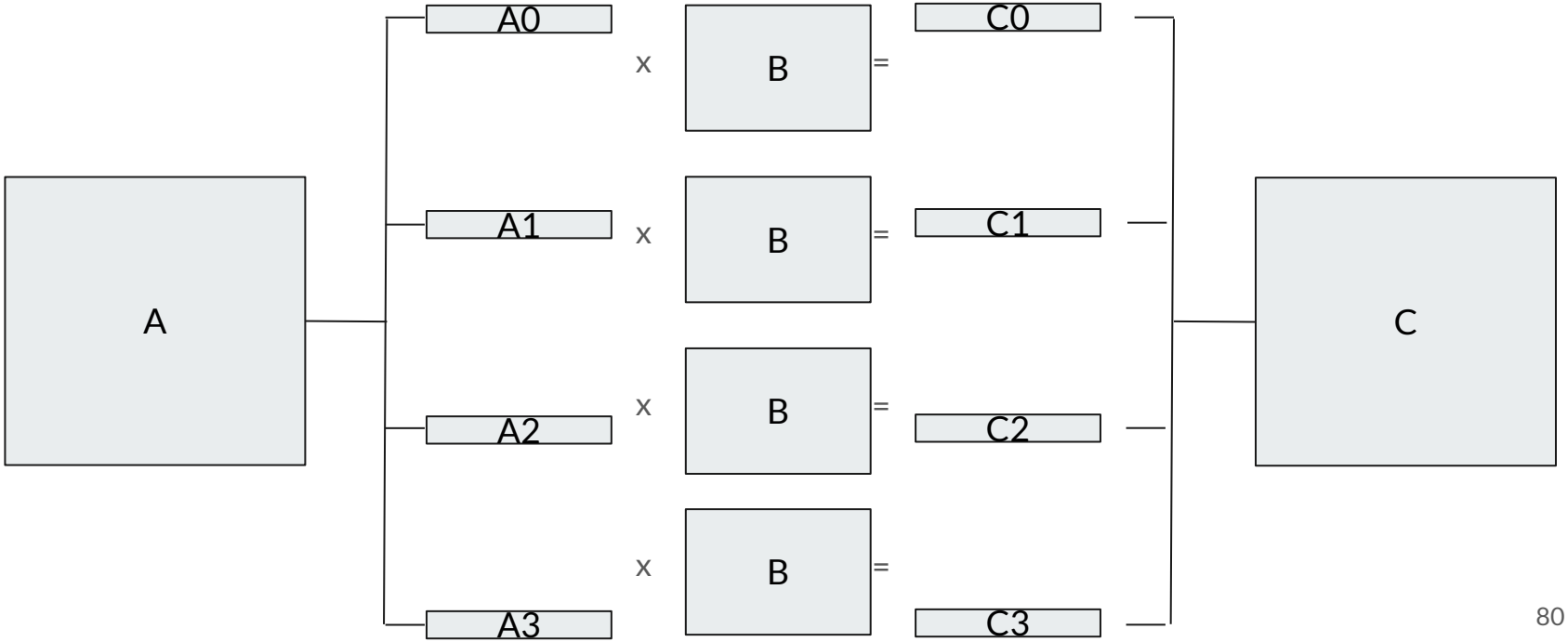


We want to split the matrix multiplication on 4 ranks:

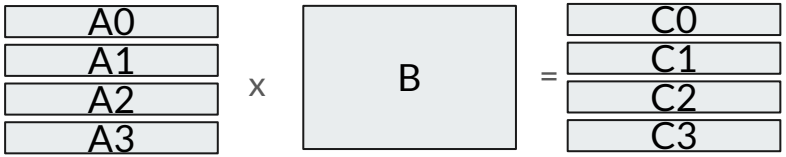


Illustrative example (Matrix multiplication)

A is split on 4 ranks, B is common to all ranks. The various MPI rands at the end communicate their result to rank 0.



Matrix multiplication (MPI_Bcast)



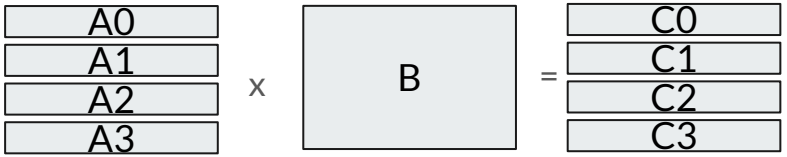
Let us assume that A and B are initialized initially on rank 0. We need to share first the dimension of the matrices

```
MPI_Bcast( num_rows,    1, MPI_INTEGER, 0, MPI_COMM_WORLD);
MPI_Bcast( num_columns, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
```

Then we send the B matrix to all other ranks

```
MPI_Bcast( &B[0][0], num_rows*num_columns,MPI_DOUBLE_PRECISION, 0,
MPI_COMM_WORLD);
```

Matrix multiplication (MPI_Bcast)



Let us assume that A and B are initialized initially on rank 0. We need to share first the dimension of the matrices

```
MPI_Bcast( num_rows,    1, MPI_INTEGER, 0, MPI_COMM_WORLD);
MPI_Bcast( num_columns, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
```

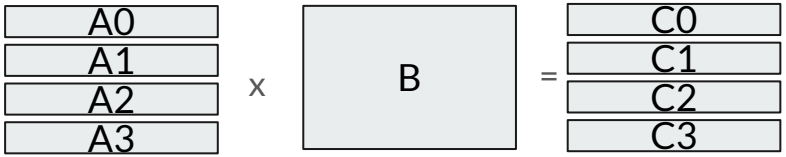
Then we send the B matrix to all other ranks

```
MPI_Bcast( &B[0][0], num_rows*num_columns, MPI_DOUBLE_PRECISION, 0,
MPI_COMM_WORLD);
```

Pointers to variables (first element for arrays and matrices)

Number of elements

Matrix multiplication (MPI_Bcast)



Let us assume that A and B are initialized initially on rank 0. We need to share first the dimension of the matrices

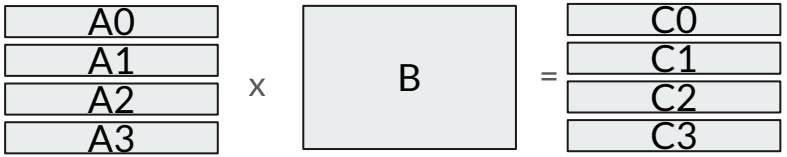
```
MPI_Bcast( num_rows,    1, MPI_INTEGER, 0, MPI_COMM_WORLD);
MPI_Bcast( num_columns, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
```

Then we send the B matrix to all other ranks

```
MPI_Bcast( &B[0][0], num_rows*num_columns, MPI_DOUBLE_PRECISION, 0,
MPI_COMM_WORLD);
```

Variable types

Matrix multiplication (MPI_Bcast)



Let us assume that A and B are initialized initially on rank 0. We need to share first the dimension of the matrices

```
MPI_Bcast( num_rows,    1, MPI_INTEGER, 0, MPI_COMM_WORLD);
MPI_Bcast( num_columns, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
```

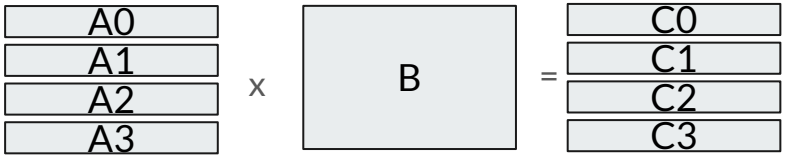
Then we send the B matrix to all other ranks

```
MPI_Bcast( &B[0][0], num_rows*num_columns, MPI_DOUBLE_PRECISION, 0,
MPI_COMM_WORLD);
```

COMM WORLD all receiving the information

Rank becasting the information (RANK 0)

Matrix multiplication (MPI_Send / MPI_Recv)



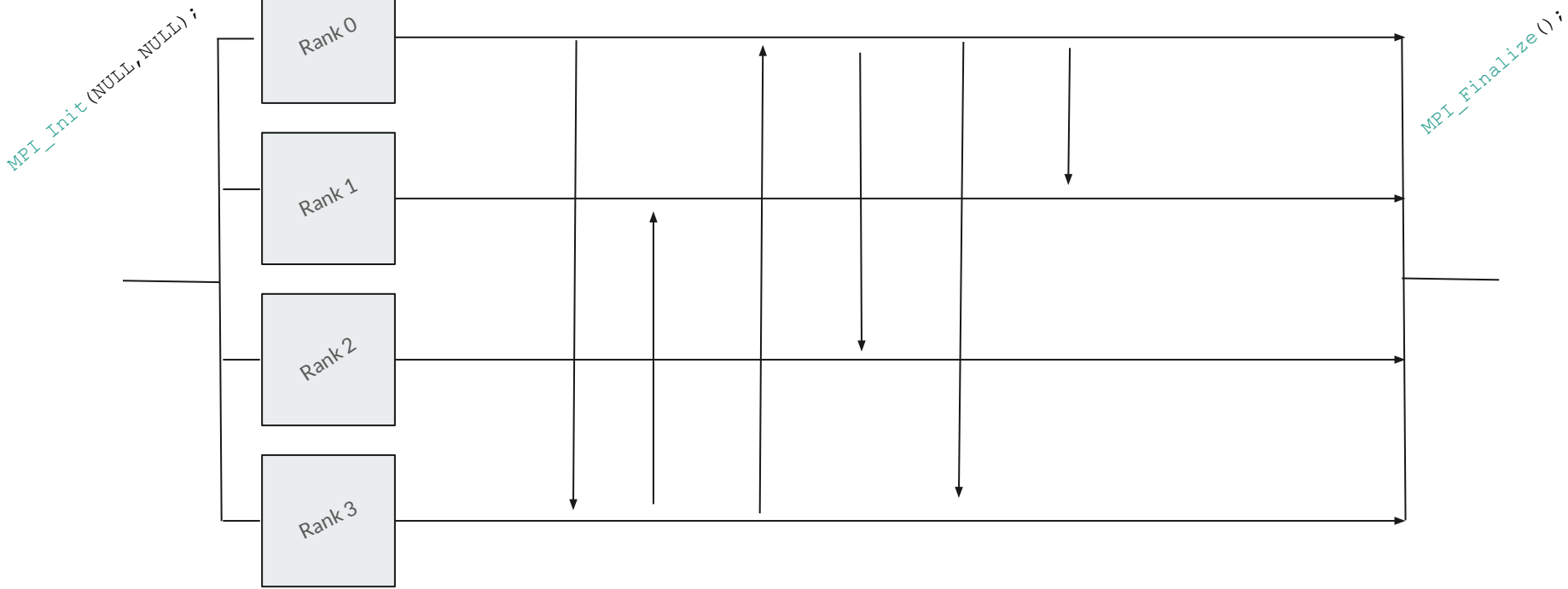
We now need to send parts of the A matrix to each rank.

```
for (rank = 0 ; rank < num_of_ranks ; ++rank ){
      if (mpi_rank == 0) {
            // Process 0 sends data
            index = rank*num_row_fraction
            MPI_Send(&A[index][0], num_columns*num_row_fraction, MPI_DOUBLE_PRECISION, rank, 0,MPI_COMM_WORLD);

      } else if (mpi_rank == rank) {
            // Process 1 receives data
            MPI_Recv(&A[index][0], num_columns*num_row_fraction, MPI_DOUBLE_PRECISION, 0, 0,MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      }
}
```

Now each rank as B, and a fragment of A, and it can compute the fragment of C, that has to be sent back to main rank.

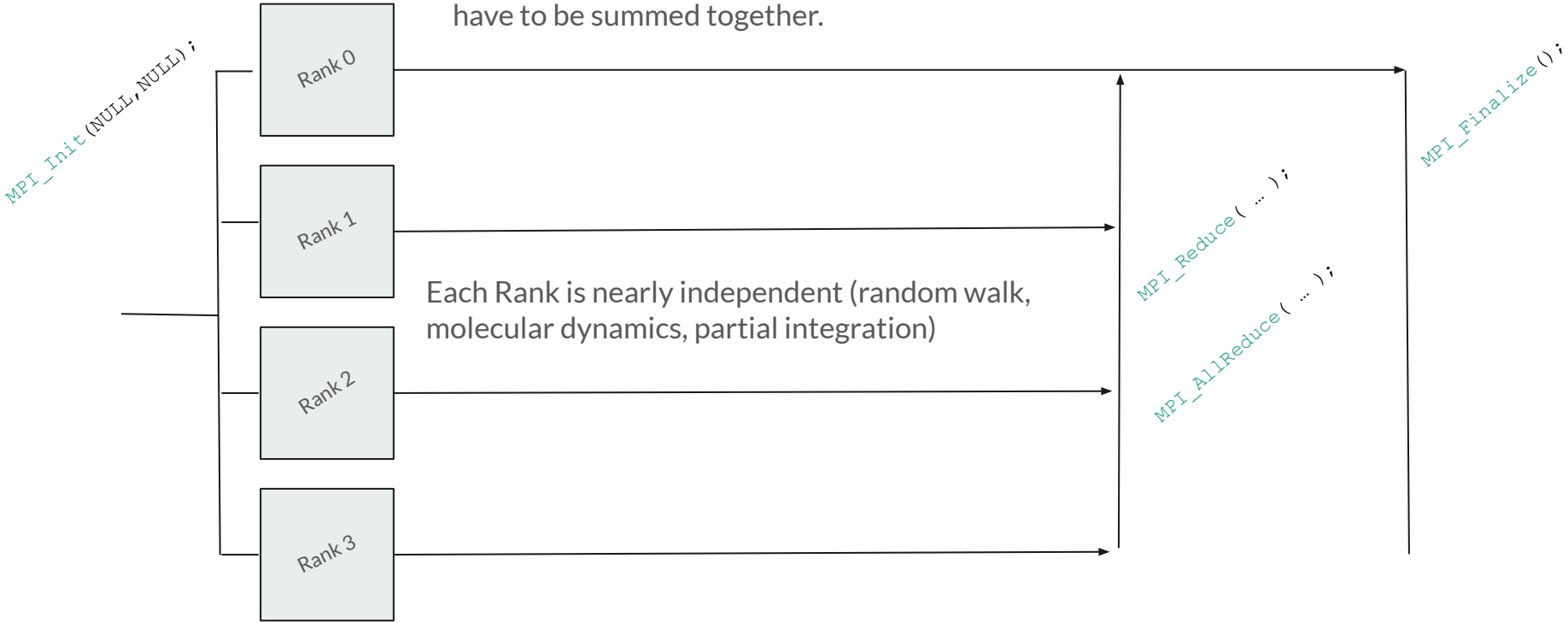
MPI Drawback

NOT EFFICIENT IF TOO MANY COMMUNICATIONS OF LARGE DATA BLOCKS!!!



MPI Best cases

Computation of independent LOCAL quantities (INTEGRALS) that have to be summed together.



MPI reduce & allreduce

They sum over (different operations are possible) each value shared to the MPI_COMM_WORLD

```
MPI_Reduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0,
MPI_COMM_WORLD);
MPI_Allreduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM,
MPI_COMM_WORLD);
```

MPI reduce & allreduce

They sum over (different operations are possible) each value shared to the MPI_COMM_WORLD

```
MPI_Reduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0,
MPI_COMM_WORLD);
MPI_Allreduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM,
MPI_COMM_WORLD);
```

Number of elements and variable type

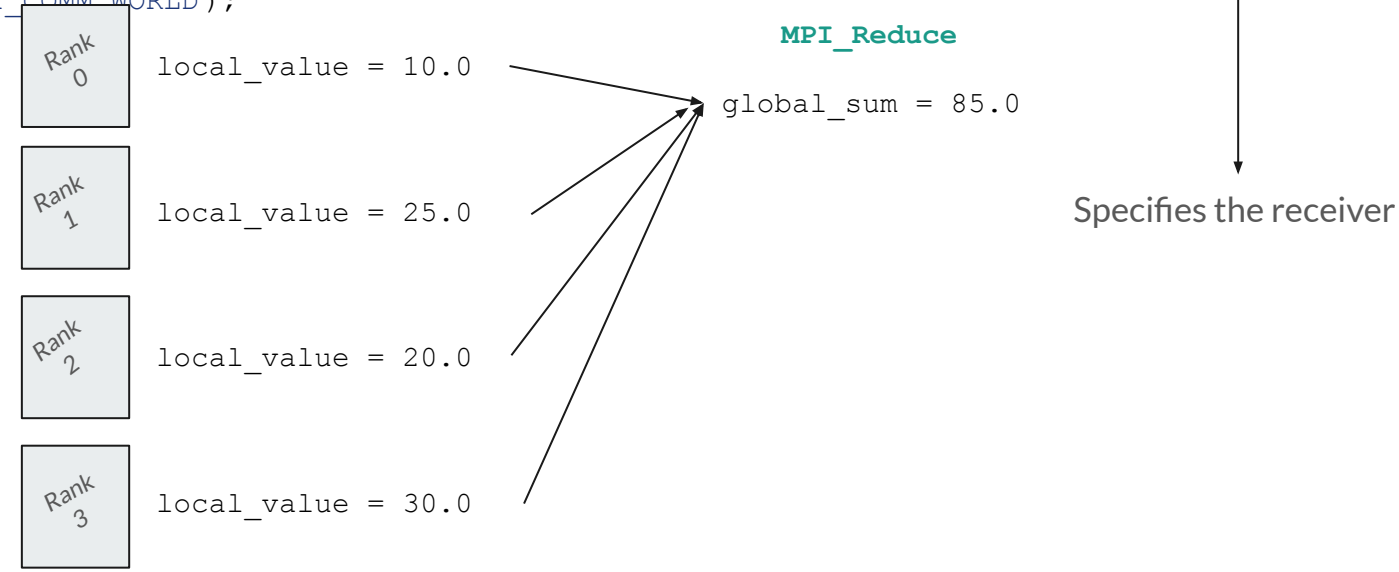
Global variable

Variable stored by each rank

MPI reduce & allreduce

They sum over (different operations are possible) each value shared to the MPI_COMM_WORLD

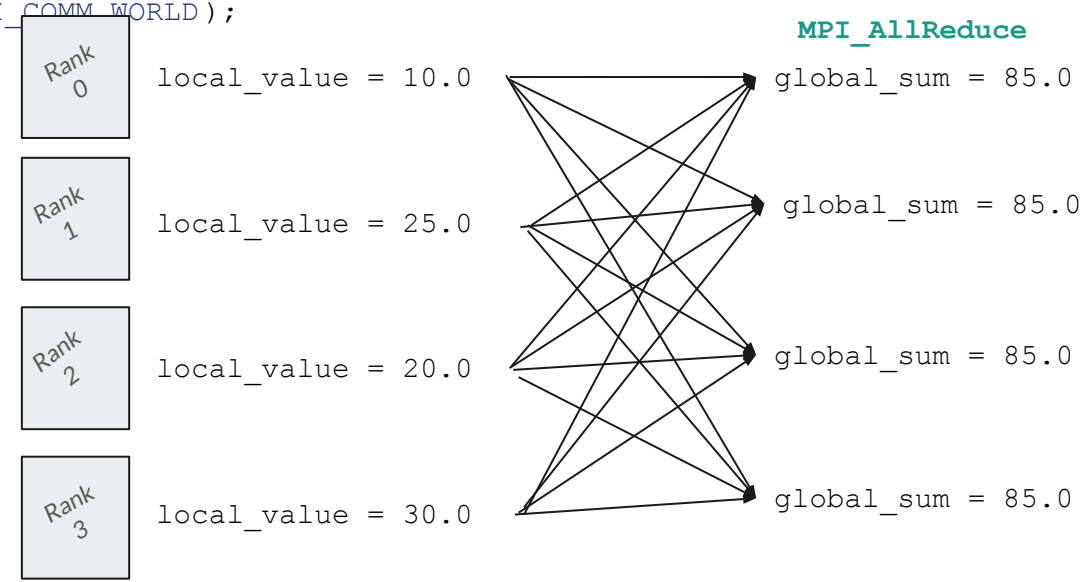
```
MPI_Reduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0,
MPI_COMM_WORLD);
MPI_Allreduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM,
MPI_COMM_WORLD);
```



MPI reduce & allreduce

They sum over (different operations are possible) each value shared to the MPI_COMM_WORLD

```
MPI_Reduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0,
MPI_COMM_WORLD);
MPI_Allreduce(&local_value, &global_sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM,
MPI_COMM_WORLD);
```

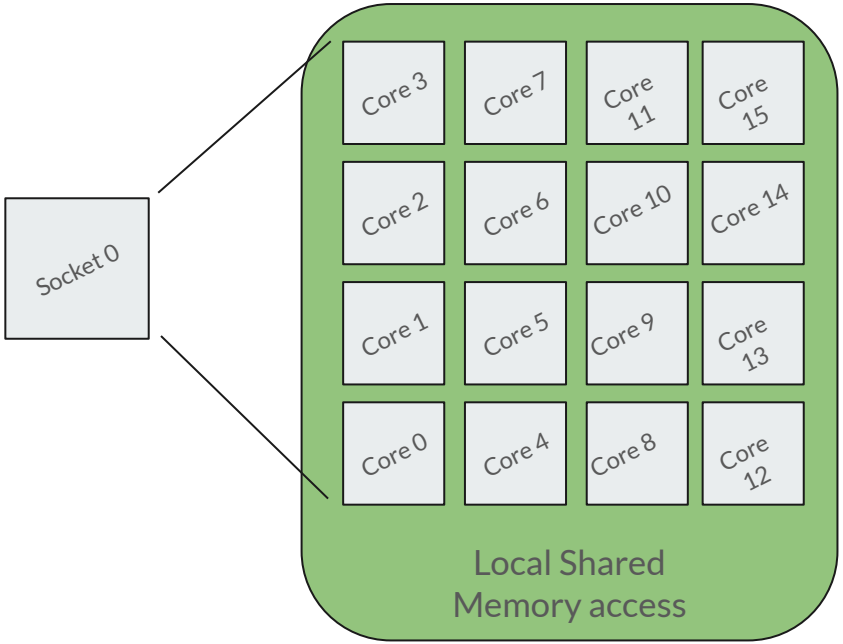


Open Multi-Processing

Open Multi-Processing (OpenMP)

OpenMP (Open Multi-Processing) is an API (Application Programming Interface) that supports **multi-platform shared-memory parallel programming** in **C, C++, and Fortran**.

- **Programming Model**: Shared memory
- **Execution**: Threads run in parallel on shared memory space
- **Ease of Use**: Easier to implement (uses compiler directives in C/C++/Fortran)
- **Communication**: Implicit (threads share variables by default)
- **Scalability**: Limited by number of cores on a single machine
- **Best For**: Multi-core shared-memory systems



Compilation with OpenMP

OpenMP preprocessing is integrate in all compilers (gcc, g++, gfortran, ifort, icc, … ecc.)

Full guide to the directives can be found here: https://www.openmp.org/

To compile a simple C application such as the one defined in the previous slides we run:

```
icc -openmp -o myapplication.exe myapplication.cpp
```

or for gcc:

```
gcc -fopenmp -o myapplication.exe myapplication.cpp
```

Set the environment variable OMP_NUM_THREADS at runtime
To do this outside the code, you would set the environment variable before running the program:

```
export OMP_NUM_THREADS=4    #(for Linux/macOS)
set OMP_NUM_THREADS=4       #(for Windows)
```

OMP environment

Set the environment variable OMP_NUM_THREADS at runtime
To do this outside the code, you would set the environment variable before running the program:

```
export OMP_NUM_THREADS=4    #(for Linux/macOS)
set OMP_NUM_THREADS=4       #(for Windows)
```

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
        int num_threads;

        num_threads = omp_get_num_threads();  // Number of threads currently running
        printf("Number of threads: %d\n", num_threads);

        // Parallel region starts here
        #pragma omp parallel
        {
        int thread_id = omp_get_thread_num(); // Get Thread identification number
        printf("Hello from thread %d out of %d threads\n", thread_id, omp_get_num_threads());
        }

        return 0;
}
```

OMP environment

Set the environment variable OMP_NUM_THREADS at runtime
To do this outside the code, you would set the environment variable before running the program:

```
export OMP_NUM_THREADS=4    #(for Linux/macOS)
set OMP_NUM_THREADS=4       #(for Windows)
```

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
        int num_threads;
        num_threads = omp_get_num_threads();  // Number of threads currently running
        printf("Number of threads: %d\n", num_threads);

        // Parallel region starts here
        #pragma omp parallel
        {
        int thread_id = omp_get_thread_num(); // Get Thread identification number
        printf("Hello from thread %d out of %d threads\n", thread_id, omp_get_num_threads());
        }

        return 0;
}
```

Including headers

Retrieving number of threads from environment.

OMP environment

Set the environment variable OMP_NUM_THREADS at runtime
To do this outside the code, you would set the environment variable before running the program:

```
export OMP_NUM_THREADS=4    #(for Linux/macOS)
set OMP_NUM_THREADS=4       #(for Windows)
```

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
        int num_threads;
        num_threads = omp_get_num_threads();  // Number of threads currently running
        printf("Number of threads: %d\n", num_threads);

        // Parallel region starts here
        #pragma omp parallel
        {
        int thread_id = omp_get_thread_num(); // Get Thread identification number
        printf("Hello from thread %d out of %d threads\n", thread_id, omp_get_num_threads());
        }

        return 0;
}
```

Parallel execution

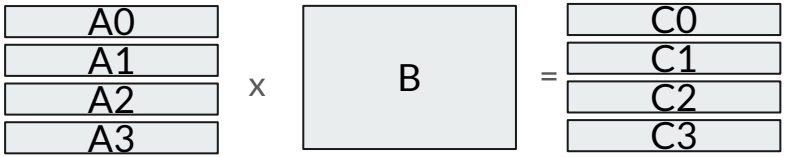
Single execution in OMP environment

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
        int num_threads;
        num_threads = omp_get_num_threads();  // Number of threads currently running
        printf("Number of threads: %d\n", num_threads);

        // Parallel region starts here
        #pragma omp parallel
        {
        int thread_id = omp_get_thread_num(); // Get Thread identification number

            // Only one thread (typically the first one that reaches this block) will execute this block
            #pragma omp single
            {
            printf("Hello from thread %d out of %d threads\n", thread_id, omp_get_num_threads());
            }
        }

        return 0;
}
```

Parallel region

Executed only once by first thread reaching it

Distribution of operations (Matrix multiplication)

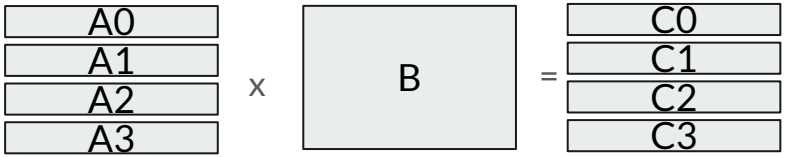



```
#pragma omp parallel private(i0,i1,i,j,k,thread_id) shared(A,B,C,N,block_length)
{
      thread_id = omp_get_thread_num(); // Get Thread identification number
      i0 = thread_id*block_length
      i1 = i0+block_length
      for (i = i0; i < i1; i++) {
            for (j = 0; j < N; j++) {
                  for (k = 0; k < N; k++) {
                        C[i][j] = A[i][k] * B[k][j];
                  }
            }
      }
}
```

Shared Variables

Private variable inside the parallel environment

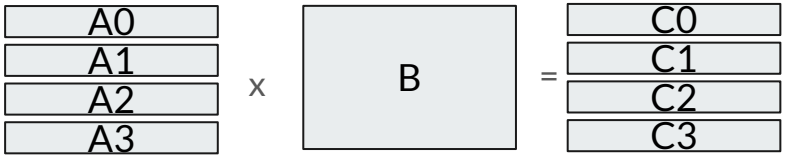
Distribution of operations (Matrix multiplication)




```
#pragma omp parallel private(i0,i1,i,j,k,thread_id) shared(A,B,C,N,block_length)
{
      thread_id = omp_get_thread_num(); // Get Thread identification number
      i0 = thread_id*block_length
      i1 = i0+block_length
      for (i = i0; i < i1; i++) {
            for (j = 0; j < N; j++) {
                  for (k = 0; k < N; k++) {
                        C[i][j] = A[i][k] * B[k][j];
                  }
            }
      }
}
```

Manual Partitioning of the matrices for each OMP thread

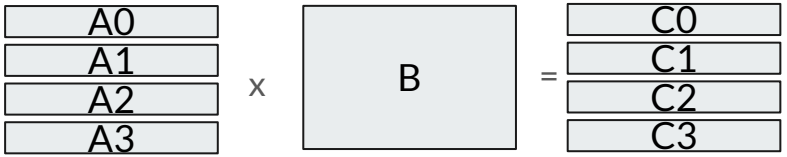
Parallel Loops (Matrix multiplication)




```
#pragma omp parallel for private(i,j,k) shared(A,B,C,N)
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        for (k = 0; k < N; k++) {
            C[i][j] = A[i][k] * B[k][j];
        }
    }
}
```

OpenMP optimizes the distribution of the loops over all the threads, starting from the most external!! (Beware of different syntax)

Reduce operations (Let's sum all elements of C)




```
double sum = 0.0
#pragma omp parallel for private(i,j,k) shared(A,B,C) reduction(+:sum)
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        for (k = 0; k < N; k++) {
            C[i][j] = A[i][k] * B[k][j];
            sum += C[i][j];
        }
    }
}
```

Command of reduction (in this case sum) of all the sums computed by each thread. (Temporary `sum' variables for each thread are generated)

Running applications with MPI and OpenMP

Pure MPI on HPC schedulers

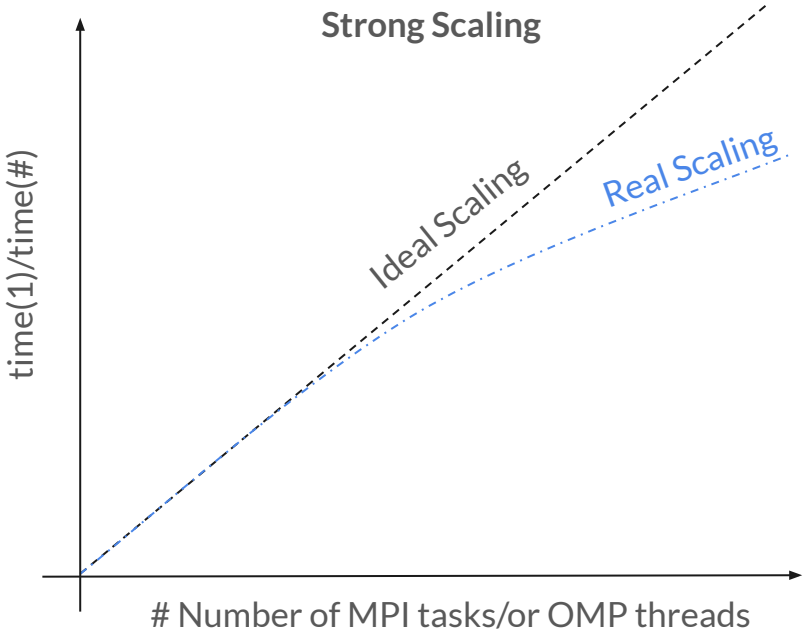
```
#!/bin/bash --login
# Multi-node parallel application MPI launcher, using 256 MPI processes

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=128    # MPI processes per node
#SBATCH --cpus-per-task=1        # OMP threads per MPI task
#SBATCH --ntasks-per-socket=16

# 1. The `--partition' option is set to `batch' by default in UL HPC, but it may
#    need to be explicitly defined in other clusters.
# 2. ${SLURM_CPUS_PER_TASK} is the value defined in the option flag `--cpus-per-task'

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK} # =1

srun -n 256 my_mpi_application.exe
```

Mixed MPI and OMP

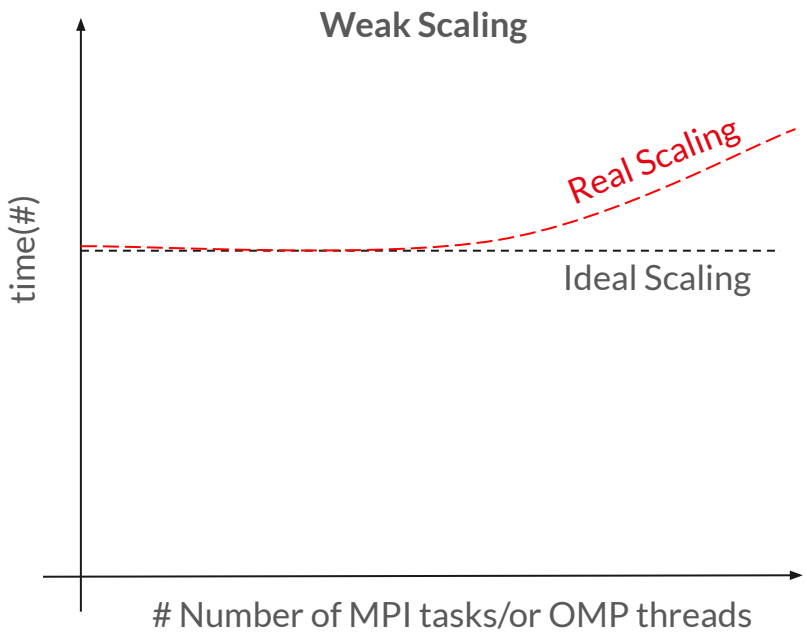
```
#!/bin/bash --login
# Multi-node parallel application MPI launcher, using 16 MPI processes each with 16 OMP
threads

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8    # MPI processes per node
#SBATCH --ntasks-per-socket=1
#SBATCH --cpus-per-task=16

# 1. The `--partition' option is set to `batch' by default in UL HPC, but it may
#    need to be explicitly defined in other clusters.
# 2. ${SLURM_CPUS_PER_TASK} is the value defined in the option flag `--cpus-per-task'

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK} # =16

srun -n 16 my_mpi_application.exe
```

Testing before production runs

Performance Testing (PLEASE TEST YOUR APPLICATIONS!!)



The total task is partitioned over the cores



Each core does the same task

Thank you!

Any questions?

Empty