IMMM Cluster Doc

R.Busselez / S.Bourdais / F.Chavanon

Table of contents

1. IMMM Cluster's documentation

1.1 About our cluster

The computation cluster of Institut des Molécules et Matériaux du Mans is hosted and managed by InfraLab.

Our cluster is composed with heterogenous nodes, devoted to high performance computations in the field of physics, chemistry and chemical physics.

To make calculation on the IMMM's cluster, you must have an account on **authlab**, facilities are open to IMMM member's including PhD and master thesis.

1.2 Cluster Documentation

- Machines and cluster specifications
- Accessing the cluster
- List of Software and Libraries
- Beginning with SLURM

To start a calculation, you **must use the queue management** SLURM any calculation launched without using the queue management software will be killed without warning.

1.3 Advanced Slurm commands

- Slurm Commands to follow jobs and cluster state
- Cluster configuration of Slurm
- Slurm options for sbatch submission
- Slurm options for interactive shell
- Examples of resources request
- Variables environment

1.4 Software installation

- Installed Software
- Development environment
- Python and Python libraries versions

1.5 Contacts

To open an account please contact IMMM authlab's managers Remi Busselez or Ivan Labaye

1.6 About this documentation

2. Ressources

2.1 Machines List and monitoring

2.1.1 Policy of cluster utilisation

- The cluster is dedicated to Research purpose
- PhD students and master students can access to the cluster
- Rules of good practice

2.2 Architecture of the cluster

Access to the cluster is only allowed through *roadrunner* login node, this node is a virtual machine managing user's connections to the cluster via *authlab* and **must not be used for calculations purpose**.

User's personal directory (a.k.a `home` in Unix world) and all the software installed are shared among all nodes through a network file system (NFS) hosted on **fury**.

2.3 Machine (computation nodes) in the SLURM cluster

To use the cluster, the user establish a connection (via ssh) on **roadrunner** (`roadrunner.univ-lemans.fr`) login node and submit his jobs via SLURM scheduler.

2.3.1 Computation nodes

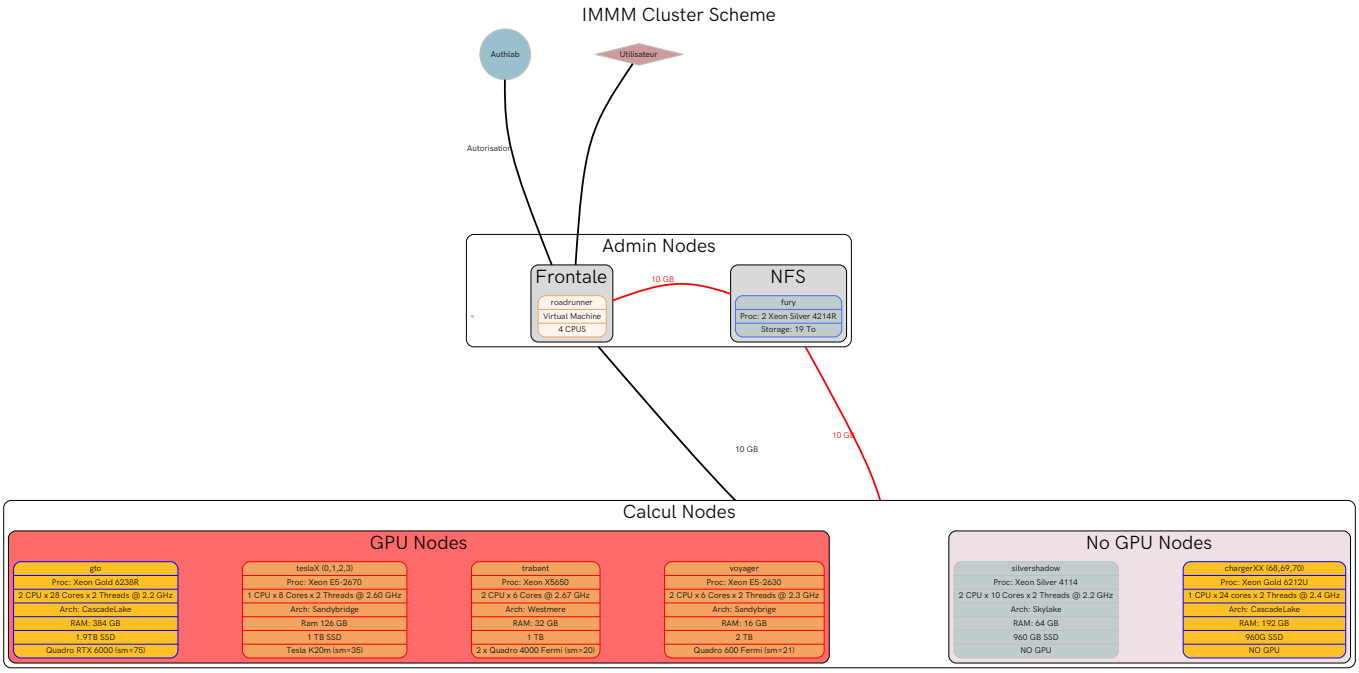
The cluster is composed with the following nodes

Hostname	CPUs (phys×cores×thread)	CPU Architecture name	Memory	GPUs
gto	112 (2x28x2)	cascadelake	384 Go	Quadro RTX 6000
charger	48 (1x24x2)	cascadelake	192 Go	No GPU
tesla{0,1,2,3}	16 (2×8×1)	sandybridge	128 Go	2 × Tesla K20m
silvershadow	40 (2x10x2)	skylake	64 Go	No GPU
voyager	24 (2×6×2)	sandybridge	16 Go	Quadro 600
trabant	12 (2×6×1)	westmere	32 Go	2 × Quadro 4000

All nodes are in the slurm *cluster* default partition.

All node have their own local `/data/` partition to be used as a *scratch* directory (fastest place to store big or temporary files).

> **Warning**
>
> For all computations using many Input/Output access on disks or writing huge temporary files it is mandatory to use `/data` directory on computation nodes instead of NFS.



Full Screen

3. Access to IMMM Cluster

As `authlab` registered user, you can connect to the cluster via login node. We first suppose that you try to connect from a machine located inside Le Mans University Network.

3.1 Inside Le Mans University Network

The login node roadrunner `roadrunner.univ-lemans.fr` is a Linux based operating system. You then need a ssh client to connect to the login node.

3.1.1 From an Unix/Linux/MacOs machine

SSH connection via command line

In the example below we assume your login is `jdupon` and the machine you work on name's yourmachine, the **jdupon** username is for example purpose, **replace** it with your authlab's login.

The simplest way to connect is using your *Terminal* Your terminal must resemble to something equivalent

```
you@yourmachine:~$
```

Then type the following command. `$` symbol is here to delimitate the prompt (left side) from the command to type (right side).

```
you@yourmachine:~$ ssh jdupon@roadrunner
```

The console ask's for your password. In the case of first connection you also have to permits the registration of the key.

Then you should see now that your prompt is changed

```
jdupon@roadrunner:~$
```

Your console is now directly linked to `roadrunner` login node and not to `yourmachine` local machine. Any command passed to the console will be executed on `roadrunner`

3.1.2 From a Windows machine

Maybe you can consider install a GNU/Linux distribution ? It's free with a lot of cool features !!! Have a look on Ubuntu, Debian, or Fedora.

If not...

Windows 10

Since July 2016, you can install a GNU/Linux subsystem into your Windows 10 OS.

1. go to *Parameters*
   a. *Update & Security*
   b. *For developers* (left column)
   c. check *Developer mode*
2. go to *Control Panel*
   a. *Programs*
   b. *Turn Windows Features On or Off*
   c. Enable the *Windows Subsystem for Linux (Beta)*
   d. *OK*
3. Reboot
4. Enter *bash* in the Windows's menu

Now, you can follow the Unix documentation.

Putty

First download and install PuTTy. Then open the software, the configuration window open automatically. In the hostname (or IP address) fied enter the login node name (`roadrunner.univ-lemans.fr`) use the port 22 and check SSH connection in the left panel.

Then click open, a console open asking for login, enter your login (`jdupon` in the above example) then your password when asked.

Congratulations you are connected.

3.2 Outside Le Mans University Network

If you are outside Le Mans University network, then you have to enter first through the access machine `transit.univ-lemans.fr` or using a VPN

3.2.1 With Le Mans Université VPN

After an email kindly asking the activation of the VPN to the DSI. Once authorised, follow the tuto to activate the VPN. Once the VPN activated, you are now considered inside the network.

3.2.2 Transit (from outside local network)

If you want to connect to roadrunner from outside of the university, you have to ask permission to connect to `transit.univ-lemans.fr`. Ask the DSI to open an access through transit. Once it have been done, you can connect to the login node through transit.

```
you@yourmachine:~$ ssh jdupon@transit.univ-lemans.fr
jdupon@transit:~$ ssh jdupon@roadrunner
jdupon@roadrunner:~$
```

It is also possible to use the ProxyJump option of the modern ssh version `-J` instead of the tunnel

```
you@yourmachine:~$ ssh -J jdupon@transit.univ-lemans.fr roadrunner
jdupon@roadrunner:~$
```

3.3 Advanced Tips and Tricks

See this section

4. Using the Cluster, Resource Management with SLURM

A cluster is an assembly of machine devoted to computing. It is composed of a login node and computation nodes.

4.1 Cluster Access

See this page

4.2 What is SLURM

SLURM (Simple Linux Utility for Resource Management) is a cluster management and job scheduling system needed for sharing the computations ressources of the different nodes between users.

Homepage : http://slurm.schedmd.com

For more information, have a look on manpages, specially man sbatch.

4.3 Understanding SLURM

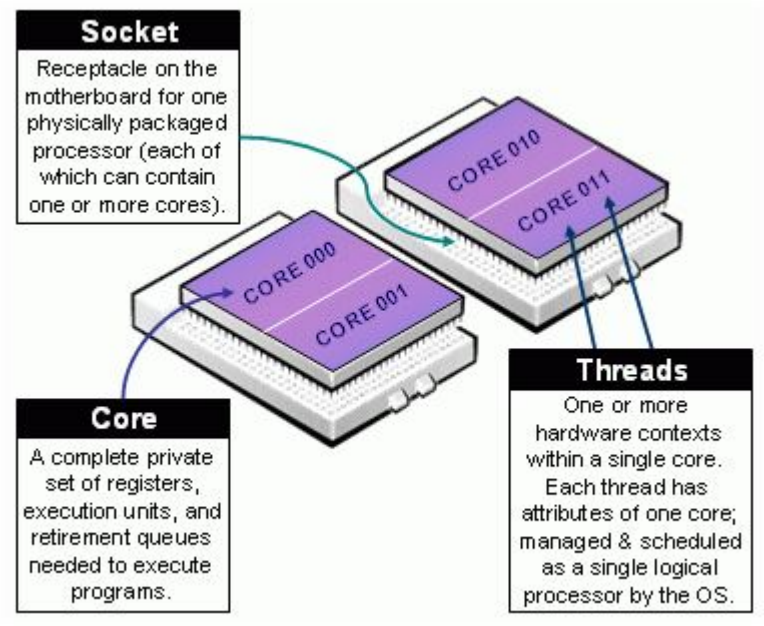
4.3.1 Nodes, sockets, cores and CPUs

- A node correspond to one physical machine of a cluster
- A node contains one or more physical processing units (sockets), each socket can contain multiple physical cores (cores) and RAM, a node can also host one or more Graphical Processing Unit (GPU)
- Some processors have hyperthreading capabilities, in those cases, one *physical core* gives *two logical units* called ***threads*** .
- In the case of hyperthreading, one physical core permits to execute two parallel threads

Important

For **SLURM** a CPU is in fact a logical unit of computation, namely

hyperthread	Physical Core	SLURM CPUS
No	1	1
yes	1	2



4.3.2 Jobs and Tasks

For SLURM a **job** correspond to a request made by an user, this request can concern any feature known by SLURM in terms of Nodes, Cpu's number, memory available, GPU, or computation time.

A job may be composed of differents *steps* and each step make one **task** or multiple parallel tasks. One task correspond to one *process* which may require one or multiple CPUS.

4.3.3 Features

Administrator can affect some *features* on the nodes depending of their characteristics. The user is then able to precise the features requested so that SLURM submit the job to the desired nodes.

4.3.4 Nodes properties and features of our cluster

Hostname	Partition	Features	gres
trabant	cluster	intel, westmere, cuda	gpu:quad4000:2
quadro	cluster	intel, sandybridge, cuda	
voyager	cluster	intel, sandybridge, cuda	
tesla*	cluster	intel, sandybridge, cuda	gpu:k20:2
silvershadow	cluster	intel, skylake	(null)
charger*	cluster	intel,cascadelake	(null)
gto	cluster	intel,cascadelake,cuda	gpu:rtx6000:1

By default, all job will be submitted on *cluster* partition.

> **Note**
>
> Ressources are allocated during the entire duration of the job. A job cannot have access to more ressources than requested. In particular take care of the time asked for the job

> **Warning**
>
> If the user does not precise options the cluster defaults are applied:
>
> - 1 Node
> - 1 Task
> - 1 CPU
> - Unlimited RAM (job can take all the memory available)
> - Unlimited time

4.4 Following the jobs and state of cluster

SLURM have many commands to follow the jobs, each command having numerous options. We give here the most commons and basics.

4.4.1 Node state

The command `sinfo` display *nodes states*, the node states can be:

- alloc : The node is entirely used
- mix : The node is partly used
- idle : No job on this name
- drain : the node is finishing submitted jobs but does not accept new ones (typically the node is about to be shutdown for maintenance)

This command is useful to have an order of estimation of the ressources available on the cluster

4.4.2 Queue state

The command `squeue` will give you informations about all the jobs queued on the cluster. You will have information about the state of the jobs (*running* or *waiting*) and the order of priority of the waiting jobs.

- `squeue -u <user>` display the jobs of the given user
- `squeue -p <nomPart>` display the jobs running of the partition `<nomPart>`
- `squeue -i <sec>` refresh the lists of jobs every `<sec>`

4.4.3 Cancelling a job

To cancel one of your job (running or waiting) you can use the command

```
scancel <jobID>
```

With `<jobId>` corresponding to the number of concerned job given with `squeue`

You are not allowed to cancel the job of another user.

More brutally you can use

```
scancel -u <user>
```

which will cancel **all** the job of the user `<user>`.

4.4.4 State of a Job

`sacct` display the state of the user job's the main possible states are:

- `CA` cancelled, the job has been cancelled by the user or administrator
- `CD` completed, job ends with success
- `CG` completing, job running
- `F` failed, job ending returning error
- `PD` pending, job is waiting for ressources
- `R` running, job running
- `TO` timeout, job ends when reaching the time limit

`sstat <jobID>` display informations about resources used by `<jobID>`

`scontrol show job <jobID>` gives detailed information on `<jobID>`

4.4.5 Submitting a job with slurm

To submit a job the user must use one of both possible ways:

- Define the job in a submission script and lauch the job with `sbatch` command
- Interactively launch the job using `srun` command

4.5 sbatch submission

The user create a shell script (`<Name_Given>.sh`) precising the ressources needed and finally the command launched (step 1). Then the job is launched using `sbatch` on the previous shell script (`sbatch <Name_Given>.sh`), a number is affected to this job and the job go in the queue (step 2).

Job description in a bash file

A SLURM submission script is composed of two parts

1. A set of options corresponding to sbatch command options. Those options permits to precise the number of cpus, memory, gpu SLURM will reserve for your job
2. The command line of the program to run which will dispose of the resources required.

The options in the bash script must begin with the `#SBATCH` directive. An example of a script permitting of launching the `sleep` command is given below

```
#!/bin/bash
# exemples d'options

#SBATCH --partition=cluster    # choix de la partition où soumettre le job
#SBATCH --time=10:0            # temps max alloué au job (format = m:s ou h:m:s ou j-h:m:s)
#SBATCH --ntasks=1             # nb de tasks total pour le job
#SBATCH --cpus-per-task=1      # 1 seul CPU pour une task
#SBATCH --mem=1000             # mémoire nécessaire (par noeud) en Mo
#SBATCH --job-name myjobname

#exécution du programme (remplacer exe par le nom du programme
# ou la ligne de commande à exécuter)
sleep 10
```

4.5.1 sbatch options

The help of sbatch is available through

```
sbatch --help
```

or Reading The Fine Manual

```
man sbatch
```

We give a list of most common options used with sbatch. Do not forget in your submission file to use the sbatch shebang `#SBATCH`

Options (raccourci)	Options (longue)	Signification and use
`-c <ncpus>`	`--cpus-per-task=<ncpus>`	Number of CPUS **per task** (1 task by default)
`-n <ntasks>`	`-ntasks=<ntasks>`	Numbers of **tasks** (1 by default)
	`--ntasks-per-node=<n>`	Number of **tasks** on each node (1 by default)
`-N <N>`	--nodes=	Number of requested nodes (1 by default)
`-C <features>`	`--constraint=<features>`	Requested features given as a boolean list with OR (|) or AND (&). Example `--constraint="cascadelake&cuda"` ou bien `--constraint="cascadelake|skylake"`
`-w <hosts>`	`--nodelist=<hosts>`	Desired nodes, be careful, all the nodes specified are reserved ! If you want one special node, it do the trick if your list contain only one node name
`-x <hosts>`	`--exclude=<hosts>`	Exclude the specified nodes from the reservation you can specify a list using comma separated hostname list: `--exclude=gto,silvershadow` The list understand substitution, hence `--exclude=tesla0,tesla1,tesla2,tesla3` is strictly equivalent to `--exclude=tesla[0-3]` or `--exclude=tesla0*3`
	`--gres=gpu:<model>:<number>`	Ask for a reservation of generic resources. In our cluster only gpu have this functionality. By default only 1 gpu is requested of any `<model>` to reserve a given model see the node table. Tesla's nodes are the only ones having 2 GPU per node. Take care as it is possible to requeest multiple gpu on different nodes. To request 2 gpu on one tesla node you have to specify `--nodes=1`
`-t <time>`	`--time=<time>`	maximum time allocated (format = m:s or h:m:s or j-h:m:s) the request `--time=0:0` gives an unlimited time
`-J <jobname>`	`--job-name=<jobname>`	give `<jobname>` to the job in addition of his number
	`--mail-type=<type>`	mail notification type = BEGIN, END, FAIL ou ALL
	`--mail-user=<user>`	mail adress for notification
`-o <out>`	`--output=<out>`	name of job log file

A good overview of sbatch/srun options is given here

4.6 srun for interactive shell

Another way for using SLURM is to request resources and open an interactive shell using those resources.

This can be done with

```
srun --pty /bin/bash
```

In this case a bash login with default values (1 node, 1 Task, 1 Job, 1 Job/task, 1 CPU) will be open on an available node. You will be able to work directly on a computation machine.

To open an interactive shell with specified node and resources the options of the previous sbatch options table can be used

```
srun --ntasks=1 --cpus-per-task=2 --pty /bin/bash
```

will open an interactive session with 2 CPU on any node available.

Profiling your request will open an interactive session of a requested node As an example if you want 16 CPU's on a charger machine

```
srun --nodes=1 --ntasks=1 --cpus-per-task=16 --constraint="cascadelake" --exclude=gto --pty /bin/bash
```

will do the trick, this command line is strictly equivalent to

```
srun -N 1 -n 1 -c 16 -C "cascadelake" -x gto --pty /bin/bash
```

> **Warning**
>
> After the end of your interactive session, do not forget to log out in order to free the resources. In an interactive session the resources are kept as long as the session is open.

4.7 Examples of resources request

In this example, you will use one node (--nodes / -N), with one task (--ntasks / -n).

```
#!/bin/bash

#SBATCH --job-name=myjobname
#SBATCH --nodes=1
#SBATCH --ntasks=1

my_program
```

By default 1 CPU/Task is reserved `my_program` programme will hence run in serial.

You can open an interactive session with the same reservation with

```
srun --job-name=myjobname --nodes=1 --ntasks=1 --pty /bin/bash
```

Once logged you can simply launch your program in an interactive way

```
my_program
```

4.7.1 OpenMP

For an OpenMP job, you have to specify *OMP_NUM_THREADS* environment variable to the number of CPU you want to use. With SLURM, you have to make a reservation according to the number of CPU's you want to use, with **--cpus-per-task / -c** parameters. In this following sample, the OMP_NUM_THREADS will be set according to your reservation.

```
#!/bin/bash

#SBATCH --job-name=myjobname
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4

# Set OMP_NUM_THREADS to the same value as -c
# with a fallback in case it isn't set.
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi

an_openmp_program
```

An interactive shell can be requested with the same resources requests

```
srun --job-name=myjobname --nodes=1 --ntasks=1 --cpus-per-task=4 --pty /bin/bash
```

Then once logged changing `OMP_NUM_THREADS` variable once

```
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
```

Then use your OpenMP program interactively

4.7.2 MPI

For a MPI job, usually, you have to specify the number of processes to use, as an example `mpirun -np 4 ./toto.x` will launch the program `toto.x` with 4 mpi processes.

With SLURM, the *-np* parameters **is not necessary**, mpiexec will get by default the parameters from the reservation.

In the simplest case your MPI Job will only run on *one node* (without communication between nodes) and you request 8 processes.

In this case the number of node is one, eight task's are required and the cpu-per-task is 1 (default), your jobfile will ressemble

```
#!/bin/bash

#SBATCH --job-name=myjobname
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=1 # Not mandatory (default)

mpiexec  a_mpi_program
```

With interactive shell

```
srun --job-name=myjobname --nodes=1 --ntasks=8 --pty /bin/bash
```

And run your MPI program

```
mpiexec a_mpi_program
```

You may also want to distribute your MPI job accross the nodes, although it is not optimal due to the latency of the IP communication between nodes.

However, in this following sample, your MPI program will run on 4 nodes, with one task per node each task taking one CPU:

```
#!/bin/bash

#SBATCH --job-name=myjobname
#SBATCH --nodes=4
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1 # Not mandatory (default)

mpiexec a_mpi_program
```

Or in interactive manner

```
srun --job-name=myjobname --nodes=4 --ntasks=4 --pty /bin/bash
```

The user is strongly advised that sharing the MPI processes amongst nodes needs homogeneity in CPU/GPU architecture. Hence, for CPU computations sharing must be done on same architecture type nodes (all nodes have sandybridge architecture or all nodes have cascadelake architecture). In the same manner for GPU computation, the nodes must have the same GPU architecture.

Using `--nodes=2 --ntasks=4` will not necessarily send two process per node. For that purpose, you have to use `--ntasks=4 --ntasks-per-node=2` or `--nodes=2 --ntasks-per-node=2`.

4.7.3 OpenMP and MPI

You can use both OpenMP / MPI parallelization strategy. Hence one MPI task can use multiple CPU'S, not only one as above.

In the following sample, your OpenMP/MPI program will run on one node, with 4 MPI task's using each 6 OpenMP CPU'S (for a total of 24 CPUS used) :

```
#!/bin/bash
#SBATCH --job-name My_MPI-OpenMP_job
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --cpus-per-task=6

# Set OMP_NUM_THREADS to the same value as -c (cpus-per-task)
# with a fallback in case it isn't set.
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi

mpiexec a_mpi_openmp_program
```

In the same manner you can request those resources in an interactive shell

```
srun --job-name=myjobname --nodes=1 --ntasks=4 --cpus-per-task=6 --pty /bin/bash
```

Afterwards it is necessary to match the `--cpus-per-task` to the `OMP_NUM_THREADS` environment variable before launching OpenMP/MPI program

```
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
mpiexec  a_mpi_openmp_program
```

4.7.4 GPU

For GPU oriented programs, you have to make a reservation of the GPU, a reservation of the node heberging the GPU is not enough. The reservation is done by the `--gres` option of SLURM, for multiple GPU nodes you must also indicate the number of GPU you want to reserve.

In the following sample we want to reserve one GPU for a program using 4 OpenMP threads

```
#!/bin/bash
#SBATCH --job-name=OMP-GPU_job
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --gres=gpu

# Set OMP_NUM_THREADS to the same value as -c (cpus-per-task)
# with a fallback in case it isn't set.
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi

my_gpu_openmp_prog
```

If you want to make a reservation for two gpu's you have to pass explicitly the number of GPU's

```
#!/bin/bash
#SBATCH --job-name=OMP-GPU_job
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --gres=gpu:2

# Set OMP_NUM_THREADS to the same value as -c (cpus-per-task)
# with a fallback in case it isn't set.
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi

my_gpu_openmp_prog
```

It is also possible to reserve a particular model of GPU with the `--gres=gpu` option (rtx6000, k20, quadro)

```
#!/bin/bash
#SBATCH --job-name=OMP-GPU_job
#SBATCH --nodes=1
```

```
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --gres=gpu:rtx6000

# Set OMP_NUM_THREADS to the same value as -c (cpus-per-task)
# with a fallback in case it isn't set.
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi

my_gpu_openmp_prog
```

GPU and MPI: exclusive vs default mode

By default the GPU cards are in exclusive mode, with this mode, only one CPU is allowed to communicate with GPU card. For some MPI program using GPU (as Lammps) and multiple access, the cards access value must be changed. This can be done using the `noexcl` in the list of constraints of the SLURM submission and keeping the same values of `--gres=gpu` according to the number of GPU you want to reserve.

```
#SBATCH --constraint=noexcl
#SBATCH --gres=gpu
```

4.7.5 Machine File

Some software need for their own MPI implementation a *machinefile* file. Here is a sample to generate one in your jobfile :

```
#!/bin/bash
#SBATCH --job-name myjobname
#SBATCH --nodes=4
#SBATCH --ntasks=4

MACHINEFILE=machinefile.${SLURM_JOB_ID}
scontrol show hostname $SLURM_JOB_NODELIST > $MACHINEFILE

my_program -machinefile $MACHINEFILE
```

4.8 SLURM Variable Environment

A number of variable environment are used by SLURM, those variables permits to automatize the jobfile script in a smart way using bash scripting. A list of variable is available on the SLURM documentation

We recall here the most used

4.8.1 Information about the job

Variable Name	Signification
SLURM_JOB_ID	Number of job
SLURM_JOB_NAME	Name of the job (`J` or `--job-name`) option
SLURM_JOB_NUM_NODES	Number of nodes allocated
SLURM_SUBMIT_DIR	Directory from which the job is launched (directory of your jobfile)
SLURMD_NODENAME	Name of the nodes running the job
SLURM_CPUS_ON_NODE	Number of CPUS on the allocated node
SLURM_CPUS_PER_TASK	Number of cpus requested per task. Only set if the `--cpus-per-task` option is specified
SLURM_JOB_CPUS_PER_NODE	Count of processors available to the job on this node
SLURM_NTASKS	value of `-n`, `--ntasks` option

5. Compilers, Softwares and Libraries installed

5.1 Module environment

Due to natural HPC constraints concerning version of libraries and sofware installed the differents flavors of libraries and software installed can be reached using module environment.

A quick overview of the libraries and software installed

```
module avail
```

Loading a particular version of software in your environment can be simply done by

```
module load software/version
```

To see all the sofware and libraries loaded in your environment can be seen with `list` of `module`

```
module load cuda/6.5
module list
```

Give as output

```
Currently Loaded Modulefiles:
 1) gcc/4.8   2) cuda/6.5
```

Once loaded a software can be uncharged doing

```
module remove sofware/version
```

To remove all the loaded software and libraries

```
module purge
```

Beware when removing a software from your environment, some software needs libraries as dependencies when calling the software, the dependencies are automatically loaded. These dependencies are however not always automatically removed when removing the software

To change version of a software loaded from `X` to `Y` you can use `switch` of `module`

```
module load sofware/version.X
module swith software/version.X software/version.Y
```

5.2 Softwares

5.2.1 Abinit

ABINIT is an open-source suite of programs for materials science, distributed under the GNU General Public License. ABINIT implements density functional theory, using a plane wave basis set and pseudopotentials, to compute the electronic density and derived properties of materials ranging from molecules to surfaces to solids.

ABINIT have been compiled with different compilers in different versions with MPI support and without OpenMP support.

You can list availables versions with `module avail abinit`.

Two slurm jobfiles are proposed, you have to edit and run it, following SLURM documentation.

Abinit template submission MPI only

```
#!/bin/bash
# Put this file where your abinit files are.
# edit "FILE" variable to match your "files" file.
FILE=my.files
# choose your abinit version
# Replace X.Y by desired version and compiler
module load abinit/X.Y

# SLURM Control (can be overwritten by the command line)
#SBATCH --job-name=NameOfMyJob
#SBATCH --nodes=1
#SBATCH --ntasks=NT # Replace NT by the number of CPUS you want
#SBATCH --cpus-per-task=1 # One cpu / task


# Our current working directory
SRC=$(pwd)
src=$(basename ${SRC})
echo -n "# Start job : " ; date
echo "# Source directory : $SRC"
echo "# Compute host/directory : @${SLURM_NODELIST}:$SCRATCH/$src"
echo "# Abinit path : $(which abinit)"

echo "# Copying file to $SLURM_NODELIST"
# slurm-* are excluded : if not, you will not have the slurm output
rsync -av --exclude 'slurm-*' ${SRC} "${SCRATCH}"/

cd "${SCRATCH}/${src}"

echo -n "# Start abinit : " ; date

mpiexec  abinit < ${FILE} > abinit-${SLURM_JOBID}.log

echo -n "# End abinit : " ; date

cd "$SRC"
echo "# Copying file from $SLURM_NODELIST"
rsync -av "${SCRATCH}/${src}/" ./
# remove the scratch
rm -rf "${SCRATCH}/${src}"

echo -n "# End job : " ; date

echo "
Log of the abinit program are in the abinit-${SLURM_JOBID}.log file"
```

Abinit template submission OpenMP only

```
#!/bin/bash

# Put this file where your abinit files are.
# edit "FILE" variable to match your "files" file.
FILE=my.files
# choose your abinit version
# Replace X.Y by desired version and compiler
module load abinit/X.Y

# SLURM Control (can be overwritten by the command line)
#SBATCH --job-name NameOfMyJob
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=NCPUS # Number of CPUS wanted

# Automatically choose OMP_NUM_THREADS
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi


# Our current working directory
SRC=$(pwd)
src=$(basename ${SRC})
echo -n "# Start job : " ; date
echo "# Source directory : $SRC"
echo "# Compute host/directory : @${SLURM_NODELIST}:$SCRATCH/$src"
echo "# Abinit path : $(which abinit)"
echo "# Copying file to $SLURM_NODELIST"
# slurm-* are excluded : if not, you will not have the slurm output
rsync -av --exclude 'slurm-*' ${SRC} "${SCRATCH}"/

cd "${SCRATCH}/${src}"

echo -n "# Start abinit : " ; date

abinit < ${FILE} > abinit-${SLURM_JOBID}.log

echo -n "# End abinit : " ; date

cd "$SRC"
echo "# Copying file from $SLURM_NODELIST"
rsync -av "${SCRATCH}/${src}/" ./
# remove the scratch
rm -rf "${SCRATCH}/${src}"

echo -n "# End job : " ; date

echo "
Log of the abinit program are in the abinit-${SLURM_JOBID}.log file"
```

Abinit template submission MPI/OpenMP hybrid

```
#!/bin/bash

# Put this file where your abinit files are.
# edit "FILE" variable to match your "files" file.
FILE=my.files
# choose your abinit version
# Replace X.Y by desired version and compiler
module load abinit/X.Y

# SLURM Control (can be overwritten by the command line)
#SBATCH --job-name=NameOfMyJob
#SBATCH --nodes=1
#SBATCH --ntasks=NTMPI # Number of MPI Tasks
#SBATCH --cpus-per-task=OMP/MPI # Number of OpenMP / MPI process

# Automatically choose the good OMP_NUM_THREADS
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
else
    export OMP_NUM_THREADS=1
fi

# Our current working directory
SRC=$(pwd)
src=$(basename ${SRC})
echo -n "# Start job : " ; date
echo "# Source directory : $SRC"
echo "# Compute host/directory : @${SLURM_NODELIST}:$SCRATCH/$src"
echo "# Abinit path : $(which abinit)"
echo "# Copying file to $SLURM_NODELIST"
# slurm-* are excluded : if not, you will not have the slurm output
rsync -av --exclude 'slurm-*' ${SRC} "${SCRATCH}"/

cd "${SCRATCH}/${src}"

echo -n "# Start abinit : " ; date

mpiexec abinit < ${FILE} > abinit-${SLURM_JOBID}.log

echo -n "# End abinit : " ; date

cd "$SRC"
echo "# Copying file from $SLURM_NODELIST"
rsync -av "${SCRATCH}/${src}/" ./
# remove the scratch
rm -rf "${SCRATCH}/${src}"

echo -n "# End job : " ; date

echo "
Log of the abinit program are in the abinit-${SLURM_JOBID}.log file"
```

5.2.2 Gaussian

Gaussian is a computer program for computational chemistry. Use `module load gaussian` to load the latest version available.

Gaussian can only be used on one node with multiple processors via OpenMP. You have to set the number of shared processors in your input file as `%nprocshared=4` and run your batch file with slurm parameters `-c 4` or `--cpus-per-task=4` if you want to run it on 4 processors.

This following slurm's *jobfile* will automatically adjust your Gaussian input file according to your reservation (`--cpus-per-task`).

Gaussian Template using chk file as starting point (work also without beginning check point)

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --cpus-per-task=4

#Add the following to specify a constraint
#SBATCH --constraint sandybridge

#Keep and modify the following line to specify a given node
#SBATCH --nodelist charger68

#Keep and modify the following line to exclude some nodes
#SBATCH --exclude tesla[0-3],gto

# Set the name of your input file :
INPUT=input.com

# Load environment
module load gaussian/g16_B01

# Get number of shared processors
if [ -n "$SLURM_CPUS_PER_TASK" ] ; then
    CORES=$SLURM_CPUS_PER_TASK
else
    CORES=1
fi

# Adjust your input file:
perl -i -pe "s/%nprocshared=.*/%nprocshared=$CORES/" $INPUT
perl -i -pe "s/%nproc=.*/%nproc=$CORES/" $INPUT

# Set output file name based on input file name :
OUTPUT="${INPUT%.*}.log"
# ${SLURM_SUBMIT_DIR} is the variable environment giving the location where you launch the job
# Workdir is the dir you will work on
# By default, work on Gaussian scratch dir GAUSS_SRCDIR is automatically created
# when calling any gaussian module
# GAUSS_SCRDIR is equivalent to "/data/<user>/gauss_scrdir" directory on any computation node
WRKDIR=${GAUSS_SRCDIR}
# store the name of the checkpoint file chkfile in your input file
chkfile=$(grep '%[Cc][Hh][Kk]=' ${INPUT} | cut -d'=' -f2)
# Creation of files array to copy
files=(${INPUT} ${chkfile})
# Copy the files in computation node working directory
cp ${files[@]} ${WRKDIR}/.
# go to the working directory
cd ${WRKDIR}
# the log is written on roadrunner NFS
g16 $INPUT ${SLURM_SUBMIT_DIR}/$OUTPUT
# Recovering the files from workdir of computation nodes to roadrunner directory where you launche the job
cp ${files[@]} ${SLURM_SUBMIT_DIR}/.
# cleaning the temporary files
rm ${files[@]}
```

5.2.3 LAMMPS

LAMMPS is s a classical molecular dynamics code written in C++

5.2.4 Octopus

Octopus is a scientific program aimed at the ab initio virtual experimentation on a hopefully ever-increasing range of system types.

This version have been compiled with MPI support : `module load octopus/6.0-gnu`

5.2.5 Quantum-Espresso

Quantum-Espresso has been compiled optimally depending node's architecture, with Intel and GCC compilers.

- GCC : `module load quantum-espresso/6.5-gcc-8.3`
- Intel Compilers : `module load quantum-espresso/5.3.0-intel`

If you want to run in MPI environment, don't mix architectures.

5.2.6 QMC

5.2.7 Yambo

Yambo is a FORTRAN/C code for Many-Body calculations in solid state and molecular physics.

```
module load yambo/4.5.1-gcc-8.3.0
```

5.3 Libraries

5.3.1 HDF 5

5.3.2 Netcdf

5.3.3 Netcdf-Fortran

5.3.4 Plumed

5.3.5 Libxc

5.3.6 FFTW

5.3.7 Openblas

5.4 Development Tools

In this section you will find the development tools installed on the cluster and available on all nodes

5.4.1 GCC

The GNU compilers suite (`gcc` , `g++` & `gfortran`) are available in versions: **8.3, 6.3, 4.9, 4.8**

The default version is 8.3

Access to other versions are possible using `g<lang>-<version>` , where `<lang>` is the wanted language and `<version>` the desired version.

> **Warning**
>
> It is strongly recommended to use the module environment to override the default compiler in order to modify the link to the libraries.

5.4.2 Message Passing Interface library (MPI)

Default MPI library is the OpenMPI v3.0 packaged for Debian Buster. The MPI interface for GCC tools can directly be reached through

- `mpicc` wrapper for gcc MPI implementation
- `mpic++` wrapper for g++ MPI implementation
- `mpifort` wrapper for gfortran MPI implementation

Note that `mpif77` and `mpif90` although still presents are deprecated

5.4.3 Intel Composer Suite

Version 19 of Intel Composer suite is available via module environment. When calling the intel composer suite, you will load in your environment

- `icc` C compiler
- `icpc` C++ compiler
- `ifort` Fortran compiler

You will also have access to intel Math Kernel Library

The MPI interface will also change to use the **Intel MPI implementation** in place of OpenMPI

Once the Intel Composer loaded the links to default wrapper are changed

- `mpicc` wrapper for `icc` MPI implementation
- `mpicpc` wrapper for `icpc` MPI implementation
- `mpif77` wrapper for `ifort` with F77 norm
- `mpif90` wrapper for `ifort` with F90 norm
- `mpiifort` wrapper for `ifort` with Fortran 2008 norm and part of 2018

> **Note**
>
> When loading intel fortran 19, MPI implementation of GCC are changed to `mpicc.openmpi` `mpic++.openmpi` and `mpifort.openmpi`

5.4.4 CUDA Toolkit

CUDA Toolkit 6.5, 7.5, 8.0, 9.2 and 10.2 are available on the cluster. Take care of the matching between SM capacities of nodes, compilers versions and CUDA Toolkit versions.

A reminder is available here

5.5 Python environment

5.5.1 Python virtual environment

Python 3 (v 3.7) and 2 (v 2.7) are installed on the cluster with the Ipython shell. Common scientific python libraries are installed (numpy, scipy, matplotlib).

In the case of specific python libraries or software needs, it is recommended to use `pip` and virtualenv.

The virtualenv is a common solution to fix a desired python environment overriding the default one. For development purpose multiples virtual environment can be maintained in parallel by the same user.

To create a virtualenv called `<Venv>` from your home directory

```
virtualenv <Venv>
```

Which will create `<Venv>` directory.

You can also choose the python version of your choice as an example for python 2.7

```
virtualenv -p /usr/bin/python2.7 venv
```

Afterwards you will have to activate your python environment

```
source <Venv>/bin/activate
```

Normally the name of your virtual environment will appear at the left of your prompt.

To exit the `<Venv>` virtual environment use the command

```
deactivate
```

To install python package and libraries for your virtualenv, you can either use `pip` or after activating the virtualenv using the old and dirty way:

```
python setup.py build
python setup.py install
```

> **Important**
>
> When possible and for your own mental health it is highly recommended to use `pip` instead of the methode herebefore. If the program is not packaged see hereafter how to install a python program from source using pips

5.5.2 Python pip

Once your virtualenv selected you can search for your python library/software with

```
pip install <name_of_package>
```

To install a particular version:

```
pip install <name_of_package>==X.Y.Z
```

Where `X.Y.Z` are the version number.

To upgrade the package to the latest version pass `-U` or `--upgrade` to the command

```
pip install -U <name_of_package>
```

You can also use pip to install a non-packaged python library/program. Once the library/program downloaded and unzipped (or cloned via git), in a given directory, see where the `setup.py` of the project is present. In the same directory than the project you can launch:

```
pip install -e .
```

To build the package and add it to the virtualenv. Note that using the latter way, the dependencies of the project you are building from sources is not checked, you will have to check it dependencies before and install them via pip.

6. Compilers

By default, the compiler is **GCC 8.3**, but you can use the **Intel Compilers 19** (C, C++ and Fortran) using `module load intel/parallel_studio_xe_2019`.

6.1 Optimization

Because of the heterogenous nodes, you may want to optimize your code by specifying compilations options for a specific CPU architecture.

In this table, you will find for each node

1. Which `-mtune` or `-march` option to use with gcc compiler
2. Which `-march` option to use with Intel 19 compiler
3. The CUDA capabilities availables:

Hostname	GNU 8.3	Intel 19	Cuda arch (SM)
trabant	westmere	corei7	20
quadro	sandybridge	corei7-avx	20
voyager	sandybridge	sandybridge	21
tesla*	sandybridge	sandybridge	35
silvershadow	skylake	skylake or skylake-avx512	None
charger*	cascadelake	cascadelake	None
gto	cascadelake	cascadelake	75

Note that compiling directly on comptutions nodes using `-march=native -mtune=native` for gcc will do the best optimization for the processor and motherboard detected.

In any case you can do cross compilation optimization, compiling on one node with optimization for another node. As an exemple you can compile on trabant a code optimized for gto, using `-march=cascadelake -mtune=cascadelake`.

6.2 CUDA compilers compatibilities

Based on `include/host_config.h` ou `include/crt/host_config.h`, you have to follow this table to choose your CUDA version and the appropriate compiler.

The following CUDA versions are installed on the cluster and available through `module` environment.

CUDA	Intel	GNU	SM capability
6.5	=14.0	≥ 4.9	1.0 – 3.5
7.5	=15.0	≥ 4.9	2.0 – 5.x
8.0	15 - 16	≥ 5.0	2.0 – 6.x
9.2	15 - 17	≥ 7.0	3.0 - 7.2
10.2	15 - 19	≥ 8.0	3.0 - 7.5

Some hints and examples on `nvcc` compilation for `sm` and fat binary processe soon.

6.3 Installation of softwares

> **Danger**
>
> This operation is reserved for users belonging to the `clusteradm` group in authlab.

By default writing on `/opt` of the NFS needs root rights.

However softwares may be installed on `/opt` of the NFS by creating a file `opt-install.<soft>` on the user `home` directory.

Once the dummy file is created and after a short time, directory called `/opt/<soft>` and `/opt/modulefiles/<soft>` will be created if they do not exist and owning and permission will be changed to allow the user to make the changes.

7. Tips and Tricks

7.1 SSH Tips

7.1.1 Modify ~/.ssh/config to easily access login node

You can put your username inside the `~/.ssh/config` file of your local computer to avoid typing it in your command line.

```
Host roadrunner
  Hostname roadrunner.univ-lemans.fr
  User jdupon
```

and then

```
$ ssh roadrunner
```

Using the `ProxyJump` option you can also modify the `~/.ssh/config` of your local computer this way

```
Host roadrunner
  ProxyJump jdupon@transit.univ-lemans.fr
  Hostname roadrunner
  User jdupon
```

and then connect directly from inside or outside local network

```
$ ssh roadrunner
```

7.1.2 SSH Tunneling from outside local network

In one shell of your terminal create a redirection of the port 22 of roadrunner towards the port 10022.

```
you@yourmachine:~$ ssh -g -L 10022:roadrunner:22 -C jdupon@transit.univ-lemans.fr
```

Once this connection active, let this shell turned on. All the future connections using the local port 10022 are redirected to `roadrunner` port 22 through `transit`.

Open another shell and connect to roadrunner via ssh using the port 10022 on `localhost`:

```
you@yourmachine:~$ ssh -p 10022 localhost
jdupon@roadrunner:~$
```

Once the redirection active you can also use scp between your locale machine and `roadrunner` using the port mapping 10022 to 22.

```
you@yourmachine:~$ scp -P 10022 myfile_on_my_local_computer jdupon@localhost:~/Path_to_copy/.
```

or

```
you@yourmachine:~$ scp -P 10022 jdupon@localhost:~/Path_to_myfile/my_file_on_roadrunner ~/path_on_my_local_host/.
```

Or from `roadrunner` to your machine.

To make it simpler, maybe you can configure your ssh client with the `~/.ssh/config` :

```
Host transit
  Hostname transit.univ-lemans.fr
  User jdupon
  LocalForward 3128 proxy:3128

Host roadrunner.transit
  Hostname roadrunner
  User jdupon
  ProxyCommand ssh -W %h:%p transit
```

You can now, in one command, open a session on `roadrunner`, with tunnels :

```
you@yourmachine:~$ ssh roadrunner.transit
jdupon@roadrunner:~$
```

It also possible to transfer file from your computer to *roadrunner* directly :

```
you@home:~$ scp -r <src> roadrunner.transit:<dest>
```

If you want to access to restricted web pages, you can open a tunnel to the proxy.

```
you@yourmachine:~$ ssh -L 3128:proxy:3128 jdupon@transit.univ-lemans.fr
```

And then configure the proxy on your browser to the host **localhost** with the port **3128**.

7.1.3 X11 Forwarding

If you want to use graphical programs on a remote host, you have to allow *X Forwarding* :

```
$ ssh -X freelander
```

Or allow it permanently on your *~/.ssh/config* with : `ForwardX11 yes`

GLX errors

Since Xorg 1.17, you have to configure your local Xorg server to allow GLX through X11 forwarding. The idea is to start the X server with `+iglx`.

APPLE

If you have installed XQuartz `/opt/X11/bin/startx` or, if you have installed X through Macports `/opt/local/bin/startx`

```
serverargs="+iglx"
defaultserverargs="+iglx"
```

and in /etc/ssh/ssh_config (system wide, userspace isn't enough) : `ForwardX11 yes`

If it still failed, you can `rm ~/.Xauthority`.

LIGHTDM

`/usr/share/lightdm/lightdm.conf.d/50-xserver-command.conf`

```
[SeatDefaults]
# Dump core
xserver-command=X -core +iglx
```

GDM

(not yet tested) In `/etc/X11/xorg.conf`, add

```
Section "ServerFlags"
Option "AllowIndirectGLX" "on" # or "off"
EndSection
```

7.2 Gaussian Tips

Some Gaussian tips and the life is easier

7.2.1 Convert a sdf file to Gaussian

Use OpenBabel (take care as babel will be deprecated soon)

```
obabel -m -isdf fichier_sdf.sdf -og03 fichiers_gaussian.com
```

7.2.2 Conversion script

Introduce the mandatory text at the beginning of the files

```
#!/bin/bash
for i in *.com
do
    part1='%chk='
    part2='.chk\n'
    ligne1=$part1$i$part2
    ligne2='%nproc=8\n'
    ligne3='%mem=1gb\n'
    ligne4='# b3lyp/6-31+G(d) opt scrf(iefpcm,solvent=toluene)'
    entree=$ligne1$ligne2$ligne3$ligne4
    sed -i "s|#Put Keywords Here, check Charge and Multiplicity.|$entree|" ${i}
done
```

7.2.3 rendre le fichier executable

```
chmod +x nomdufichier
```

7.2.4 Script de suppression d'une ligne dans un fichier (la quatrième)

```
sed -i '4d' ${i}
```

suppression de lignes à la fin du fichier

```
for i in *.com
do
    sed -i '9,$ d' ${i}
done
```

7.2.5 remplacer des caractères par d'autres

```
for i in *.com
do
    echo ${i}
    sed -i 's|freq=readfc|opt freq |' ${i}
done
```

Autre manière

```
for i in *.com
do
    echo ${i}
    sed -i 's|opt(maxstep=10)|opt geom=check guess=read|' ${i}
    sed -i '9,61d' ${i}
done
```

Extraction des enthalpies libres d'une liste de fichiers de sortie gaussian dans un répertoire.

```
for i in *.log
do
    echo $i >> results.csv
    grep "Sum of electronic and thermal Free Energies=" $i | tail --line=1 >> results.csv
done
```

8. Good Practices A.K.A. The Good, The Bad and The Ugly

In the following section, you will find a list of admonitions for a fair use of the cluster.

8.1 The Good

General

- Be a nice Cluster-citizen: respect the defined Acceptable Use Policy
- Read documentation thoroughly and first try to verify the known path; reuse existing (and tested) launch-scripts mechanisms for job submission in the queueing system
- Consider sysadmin time planning: realize that all incoming issues have to be prioritized

srun vs sbatch

- Use srun to login on a computation node in order to: - Compile your codes - Launch some short tests of long computations
- Do not forget that exiting shell opened with srun kill instantly your job and free your reservation
- **never compile on login node**
- Use sbatch to run long computation

Before submitting long runs

- Please check:
  - Are there any errors in the script ?
  - Are the required modules loaded ?
  - Is the correct executable used ?
- Check your computer requirements upfront, and request the correct resources in your job submission script
  - Parallelisation strategy (MPI, OpenMP, MPI/OpenMP, MPI/GPU, OpenMP/GPU)
  - Number of Nodes requested
  - Number of tasks
  - Number of cpu's per task
  - If needed reserve GPU's
- Consider a short try of your jobfile using the interactive srun command, if the job's beginning run fine stop it and relaunch with sbatch

Network File System Management

- Use the *scratch* directory (`/data/<user>`) present on each computation nodes instead of NFS to write/read large files
  - Envisage a copy of the needed files from NFS to scratch directory
  - Move to the scratch directory
  - Do the computation requiring important I/O on scratch disk
  - Copy back the data from scratch to NFS
  - Remove the copied data from scratch (via the submission script or manually afterwards)
- Make your scripts generic (respect any defined Directory Structure and apply staging techniques, where needed)
- **Use variable aliasing**, avoid hardcoding of full path names, remember that any system may be modified, upgraded or simply replaced before your project finishes
- Do some form of checkpointing if your individual jobs run for more than 1 day

Tip

- Check your jobs at runtime. You could login to the computation node and check the proper execution of your jobs reading the log file
- Try to benchmark the software for scaling issues when using MPI or for large I/O issues
- Unless you have own reasons, opt for a scripting language for your code integration but, faster optimized language for the "application kernel" (in order to obtain both of maintainability & performance!)
- Do code versioning for the sources or scripts you develop

8.2 The Bad

Not Smart

- Do not forget to log out after a `srun`
- Avoid scp to transfer huge data files prefer rsync instead
- Do not forget to clean up your scratch directory especially if you are creating huge temporary files
- Avoid keeping data long time on scratch, no backup's are realized on scratch disks

Not Fair

- Do not request more than half of the resources for your own, you never know when colleagues will work
- Do not be greedy on resources, do not requests more resources than your code can use.
- When feasible, perform some short checks about the scalability of the longest and heaviest computations
- If you wake up early the Monday morning, the queue will be empty. You can launch series of jobs, but please wait until afternoon to fill it beyond reasonable.

8.3 The Ugly

Misconduct

- Users must not log onto the compute nodes to run a job directly overriding the scheduler
- If you believe interactive use of a particular node is necessary for your research then please use srun with -w option to login to a particular node
- Users must not use their `home` directory to create/read large files, this include temporary files and wave functions files of DFT or Quantum-Chemistry.

Care of Cluster

- No Bright light
- Do not get him wet
- No food after midnight