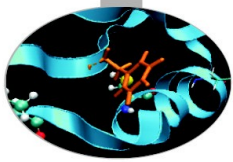
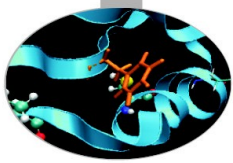
SuperComputing Applications and Innovation



Introduction to Unix Environment: modules, job scripts, PBS

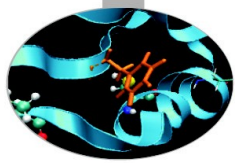
N. Spallanzani (CINECA)

In this tutorial you will learn...



- How to get familiar with UNIX environment @ CINECA
- How to submit your job to the PBS queueing system on Galileo
- Tutorial #1:
  - Getting familiar with modules and PBS
  - Example: launch a small script to the PBS queueing system
  - Analyze output results.

How to become a CINECA user



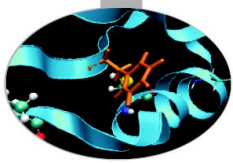
- Please fill out the form on:

  https://userdb.hpc.cineca.it/user/register

- You'll receive userdb credentials: Then
  - Click on “HPC Access” and follow the on-screen instructions
  - You'll be asked to upload an image of a valid ID document
  - Ask your PI or send an email to superc@cineca.it to be included on an active project.

- When everything is done an automatic procedure sends you (via 2 separate emails) the username and the password to access HPC systems.

How to log in



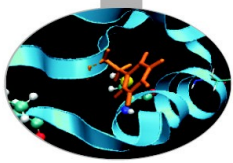
- Establish a ssh connection

  ssh <username>@login.galileo.cineca.it

- Remarks:
  - **ssh** available on all linux distros
  - **Putty** (free) or **Tectia** ssh on Windows
  - *secure shell plugin* for Google Chrome browser!
  - important messages can be found in the *message of the day*

Check the **user guide**!
https://wiki.u-gov.it/confluence/display/SCAIUS/HPC+at+CINECA%3A+User+Documentation

Storage and Filesystem



$HOME:

- Permanent, backed-up, and local.
- Quota = 50GB.
- For source code or important input files.

$CINECA_SCRATCH:

- Large, parallel filesystem (GPFS).
- Temporary (files older than 30 days automatically deleted), no backup.
- No quota. A cleaning procedure for files older than 30 days

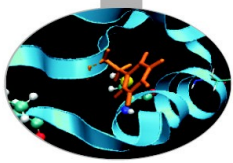
$WORK:

- Permanent, no backup, project specific, 1 Tb quota by default.

More info:

https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.4%3A+Data+storage+and+FileSystems

Accounting: saldo



```
[nspalla1@node165 ~]$ saldo -b
```

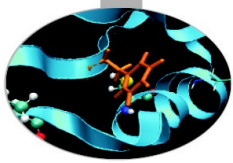
account	start	end	total (local h)	localCluster Consumed(local h)	totConsumed (local h)	totConsumed %	monthTotal (local h)	monthConsumed (local h)
cin_staff	20110323	20200323	400000004	1027976	25785995	6.4	3649635	10103
unp_prod	20130701	20161031	3800000	3134486	3460788	91.1	93592	0
cin_priorit	20131115	20191231	2000000	124084	1313371	65.7	26821	22192
unp_test	20150519	20161031	200000	650	650	0.3	11298	0
train_cnl2016	20160425	20160501	2000	0	0	0.0	0	0
train_cmdB2016	20160926	20161002	4000	0	0	0.0	20000	0

Accounting philosophy is based on the resources requested for the time of the batch job:

cost = no. of cores **requested** x job duration

In the CINECA system it is possible to have more than 1 budget (“account”) from which you can use time. The accounts available to your UNIX username can be found from the `saldo` command.

module, my best friend



- all the optional software on the system is made available through the **"module" system**
  - provides a way to rationalize software and its env variables
- on Galileo modules are divided in 3 *profiles*
  - **profile/base** (stable and tested modules)
  - **profile/eng** (contains specific software for engineering simulations)
  - **profile/advanced** (software not yet tested or not well optimized)
- each profile is divided in 4 categories
  - **compilers** (Intel, GNU, Portland)
  - **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - **tools** (e.g. Scalasca, GNU make, VNC, ...)
  - **applications** (software for chemistry, physics, ...)

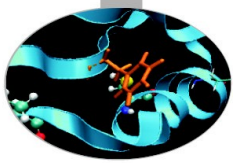
Modules

- CINECA' s work environment is organized in modules, a set of installed libs, tools and applications available for all users.

- "loading" a module means that a series of (useful) shell environment variables will be set

- E.g. after a module is loaded, an environment variable of the form "<MODULENAME>_HOME" is set

Module commands



> module available (or just “> module av”)
Shows the full list of the modules available in the profile you’re into, divided by: environment, libraries, compilers, tools, applications

> module (un)load <module_name>

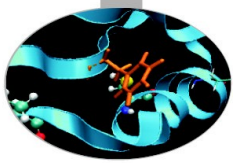
(Un)loads a specific module

> module show <module_name>
Shows the environment variables set by a specific module

> module help <module_name>
Gets all informations about how to use a specific module

> module purge
Gets rid of all the loaded modules

Launching jobs



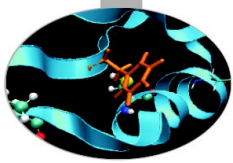
- It's time to learn how to prepare a job for its execution. The parallel execution requires a batch script.

- Marconi, Pico and Galileo have the **PBS** scheduler.

- The job script scheme is:

```
#!/bin/bash
#PBS keywords
variables environment
execution line
```

Environment setup and execution line



The execution line starts with mpirun: Given: `./myexe arg_1 arg_2`

`mpirun –np 16 ./myexe arg_1 arg_2`

`–np` is the number of **processes** you want to use

`arg_1 arg_2` are the normal arguments of myexe

In order to use mpirun, **openmpi** (or **intelmpi**) has to be loaded. Also, if you linked dynamically, you have to remember to load every library module you need (automatically sets the LD_LIBRARY_PATH variable).

The environment setting usually starts with "**cd $PBS_O_WORKDIR**". That's because by default you are launching on your home space the executable may not be found.

$PBS_O_WORKDIR points to the directory from where you're submitting the job .

PBS keywords

```
#PBS –N jobname                                                    # name of the job
#PBS -o job.out                                                    # output file
#PBS -e job.err                                                    # error file
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=4:mem=120GB # resources
#PBS -l walltime=24:00:00                                          # hh:mm:ss
#PBS -q <queue>                                                    # chosen queue
#PBS -A <my_account>                                               # name of the account
```

select = number of chunk requested

ncpus = number of cpus per chunk requested

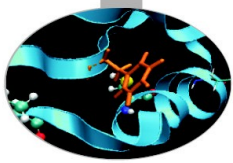
mpiprocs = number of mpi tasks per node/chunk

mem = RAM memory per chunk

ngpus = number of CUDA devices per node

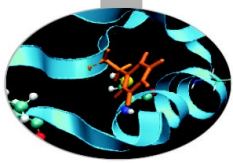
nmics = numer of Intel-Phi devices per node

PBS job script template



```
#!/bin/bash
#PBS -l walltime=2:00:00
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=4:mem=120GB
#PBS -o job.out
#PBS -e job.err
#PBS -q parallel
#PBS -A <account_no>
#PBS -m mail_events          ==>     specify email notification
                                     (a=aborted,b=begin,e=end,n=no_mail)
#PBS -M user@email.com

cd $PBS_O_WORKDIR

module load autoload intelmpi/openmpi
module load somelibrary

mpirun ./myprogram < myinput
```

PBS commands



qsub

qsub <job_script>

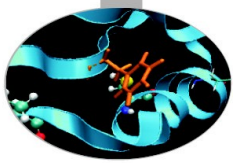
Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

qstat

qstat -a

Shows the list of all your scheduled jobs, along with their status (idle, running, closing, …) Also, shows you the job id required for other qstat options

PBS commands



qstat

qstat -f <job_id>

Provides a long list of informations for the job requested.
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

qdel

qdel <job_id>

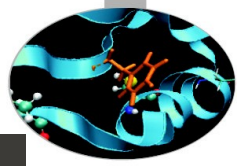
Removes the job from the scheduled jobs by killing it

qalter

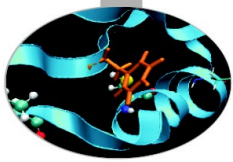
qalter -l <resources> <job_id>

Alter one or more attributes of one or more PBS batch jobs.

PBS commands: qstat

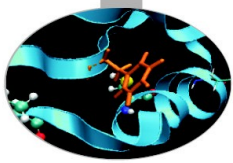


```
File  Edit  View  Search  Terminal  Help

node129:
                                                                      Est
                                                          Req'd  Req'd   Start
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
451912.node129  amessina parallel abq_parall    --    4  32    4gb 00:10 H    --
1587463.node129 aemerson parallel qsub.job      --   12 192  120gb 00:30 H    --
1587543.node129 aemerson parallel qsub.job      --    9 144   90gb 00:30 H    --
1596264.node129 ggrazios parallel a7_nor_wat 113134   1  16   14gb 04:00 R    --
1596269.node129 dborello parallel sub.sh      26304  16  64   16gb 04:00 R    --
1596279.node129 adimasci parallel Xnavis95_c  83686   1   1    2gb 04:00 R    --
1596277.node129 adimasci parallel Xnavis95_c  87830   1   1    2gb 04:00 R    --
1596278.node129 adimasci parallel Xnavis95_c 101923   1   1    2gb 04:00 R    --
1596287.node129 gagate00 parallel esegui_8SP  19442   1  16   14gb 04:00 R    --
1596304.node129 mrizzini parallel ipcOptimal  90425   1  12    9gb 04:00 R    --
1596305.node129 mrizzini parallel ipcOptimal  72089   1  12    9gb 04:00 R    --
1596306.node129 mrizzini parallel ipcOptimal  71042   1  12    9gb 04:00 R    --
1596307.node129 mrizzini parallel ipcOptimal   8235   1  12    9gb 04:00 R    --
1596308.node129 mrizzini parallel ipcOptimal 102900   1  12    9gb 04:00 R    --
1596309.node129 mrizzini parallel ipcOptimal  42979   1  12    9gb 04:00 R    --
1596310.node129 mrizzini parallel ipcOptimal  92927   1  12    9gb 04:00 R    --
1596311.node129 mrizzini parallel ipcOptimal  90698   1  12    9gb 04:00 R    --
1596290.node129 gagate00 parallel esegui_4SP  78531   1  16   14gb 04:00 R    --
1596291.node129 gagate00 parallel esegui_4SP  37027   1  16   14gb 04:00 R    --
1596292.node129 gagate00 parallel esegui_2SP  78795   1  16   14gb 04:00 R    --
1596312.node129 mrizzini parallel ipcOptimal 105767   1  12    9gb 04:00 R    --
1596313.node129 mrizzini parallel ipcOptimal  87469   1  12    9gb 04:00 R    --
[agrottes@node129 ~]$
```



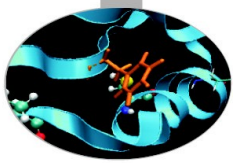
Scripts for running MD codes on Galileo

Gromacs 5.0.4, pure MPI on Galileo



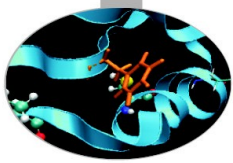
```
#!/bin/bash
#PBS -N gmx-PureMPI
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=120GB
#PBS -l walltime=1:00:00
#PBS -A train_cmdB2016
#PBS -q R1619976
#PBS -W group_list=train_cmdB2016

cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload gromacs/5.0.4

export OMP_NUM_THREADS=1             ==> set nr. Of OpenMP threads to 1 per node

mdrun=$(which mdrun_mpi)
cmd="$mdrun -s topol.tpr -v -maxh 1.0 -nb cpu"
mpirun -np 16 $cmd
```

Gromacs 5.0.4 MPI+CUDA on Galileo



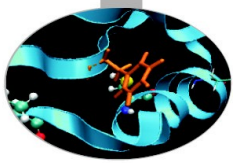
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=2:mpiprocs=2:ngpus=2:mem=15GB
#PBS -l walltime=1:00:00
#PBS -A train_cmdB2016
#PBS -q R1619980
#PBS -W group_list=train_cmdB2016

cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load cuda/6.5.14
module load autoload gromacs/5.0.4

export OMP_NUM_THREADS=1             ==> set nr. Of OpenMP threads to 1 per node
#                                    ==> set total mpi tasks = 2 and bind to two GPUs

mdrun=$(which mdrun_mpi_cuda)
cmd="$mdrun -s topol.tpr -v -maxh 1.0 -gpu_id 01 "
mpirun -np 2 $cmd
```

Gromacs 5.0.4 MPI/OpenMP+CUDA



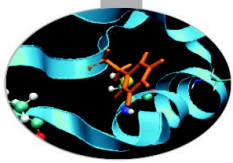
```
#!/bin/bash
#PBS -N MPI-OpenMP-GPU
#PBS -l select=1:ncpus=16:mpiprocs=4:ngpus=4:mem=120GB
#PBS -l walltime=1:00:00
#PBS -A train_cmdB2016
#PBS -q R1619980
#PBS -W group_list=train_cmdB2016

# go to submission dir
cd $PBS_O_WORKDIR

# load gromacs 5.0.4
module load profile/advanced
module load cuda/6.5.14
module load autoload gromacs/5.0.4


# we have asked for 1 nodes = 4 GPUs
# => set total mpi tasks = 4  (4 per node) and set omp tasks to fill up each node
export OMP_NUM_THREADS=4

mdrun=$(which mdrun_mpi_cuda)
cmd="$mdrun -s topol.tpr -deffnm MPI_OpenMP-GPU -v -maxh 1.0 -gpu_id 0123"
mpirun -np 4 $cmd
```

Gromacs 5.0.4 Intel Phi
(symmetric mode on Galileo)



```
#!/bin/bash
#PBS -N MPI-OpenMP-GPU
#PBS -l select=1:ncpus=16:mpiprocs=8:nmics=1:mem=14GB
#PBS -q
#PBS -l walltime=1:00:00
#PBS -A train_cmdB2016

cd $PBS_O_WORKDIR

module load intel
module load intelmpi
module load gromacs
module load mkl

string=$HOSTNAME
MICNAME=${string-mic0}
echo -e "$HOSTNAME:4\n$MICNAME:4" > machinefile

export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/mkl/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/tbb/lib/mic:${LD_LIBRARY_PATH}

export I_MPI_MIC=1
export I_MPI_PIN_MODE=mpd
export MIC_ENV_PREFIX=MIC
export MIC_KMP_AFFINITY=verbose,compact,0  # KMP_AFFINITY for MIC threads
export IVB_KMP_AFFINITY=verbose,compact,1  # KMP_AFFINITY for Host threads
export MIC_OMP_NUM_THREADS=30 # number of OMP threads on MIC
export IVB_OMP_NUM_THREADS=4  # nunber of OMP threads on Host

exe="-s topol-10k.tpr -deffnm test -maxh 2.0 -v"
mpirun -n 16 -genvall -machinefile machinefile ./symmetric.sh
```

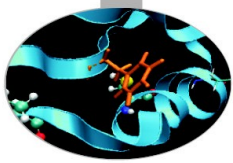
Native execution of codes on Intel Phi



MIC-native programs need to be executed inside the MIC card itself.
In order to log into a MIC card you have to:

- login to a MIC node with a PBS interactive session requesting at least 1 mic (nmics=1);
- use the "get_dev_list" script (available by loading the "superc" module on Galileo) in order to get the name of the specific MIC card assigned to you.
- get_dev_list will produce in output an hostfile named <job_id>_dev_hostfile containing the lists of the assigned cards;
- connect through ssh into the MIC card (in the example node254-mic0)

Native execution of codes on Intel Phi



Example of interactive session

```
> qsub -A <account_name> -I -l select=1:ncpus=1:nmics=1
qsub: waiting for job 10876.io01 to start
 qsub: job 10876.io01 ready
…

cd $PBS_O_WORKDIR
module load superc
get_dev_list
cat ${PBS_JOBID}_dev_hostfile
 node254-mic0
…

ssh node254-mic0  (*)
…
```

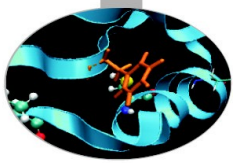
Native execution of codes on Intel Phi

In order to SSH access the mic card you have to create the public key of the Galileo username in your $HOME from login node

https://wiki.u-gov.it/confluence/display/SCAIUS/How+to+connect+by+a+public+key

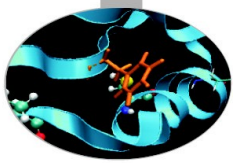
```
> ssh-keygen

> cd $HOME/.ssh

> cat id_dsa.pub >> authorized_keys
```

NAMD 2.10 MPI+CUDA on Galileo



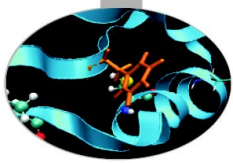
```
>module load autoload namd/2.9
>module help namd/2.9

#!/bin/bash
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=4:mem=120GB
#PBS -l walltime=0:30:00
#PBS -o namd.out
#PBS -e namd.err
#PBS -A train_cmdB2016
#PBS -W group_list=train_cmdB2016
#PBS -q R1619980

cd $PBS_O_WORKDIR                ==> change to current dir

module load profile/advanced
module load autoload namd/2.10

namd2=$(which namd2_cuda)        ==> set path to namd executable

mpirun -np 16 $namd +idlepoll +devices 0,1,2,3 md.namd
```

NAMD 2.10 Intel Phi on Galileo



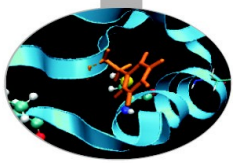
```
>module load autoload namd/2.10
>module help namd/2.10

#!/bin/bash
#PBS -l select=1:ncpus=16:mpiprocs=16:nmics=2:mem=120GB
#PBS -l walltime=0:30:00
#PBS -o namd.out
#PBS -e namd.err
#PBS -A train_cmdB2016
#PBS -W group_list=train_cmdB2016
#PBS -q R1619976

cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload namd/2.10

namd=$(which namd2.mic)              ==> set path to namd executable

mpirun -np 16 $namd md.namd          ==> run MIC version of NAMD
```

Amber-14 on Galileo (pure MPI version)



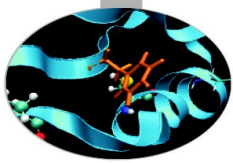
```
#!/bin/bash
#PBS -N amber
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=120GB
#PBS -o job.out
#PBS -q R1619976
#PBS -A train_cmdB2016
#PBS -W group_list=train_cmdB2016

cd $PBS_O_WORKDIR                    ==> change to current dir
module load autoload amber/14

cmd="pmemd.MPI -O -i mdin -o mdout -p prmtop -c inpcrd -r restrt -x mdcrd"
mpirun -np 16 $cmd
```

Amber 14 – MPI+CUDA version



```
#!/bin/bash
#PBS -l select=1:ncpus=2:mpiprocs=2:ngpus=2:mem=15GB
#PBS -l walltime=1:00:00
#PBS -o job.out
#PBS -q R1619980
#PBS -A train_cmdB2016
#PBS -W group_list=train_cmdB2016

cd $PBS_O_WORKDIR
module load autoload amber/14

# for best performance use 1 mpi task/1 gpu.  In this example we have 1*2 gpus = 2 MPI tasks.

cmd="pmemd.cuda.MPI -O -i mdin -o mdout -p prmtop -c inpcrd -r restrt -x mdcrd"

mpirun -np 2 $cmd
```

Tutorial 1: getting familiar with PBS



- Connect to Galileo: ssh username@login.galileo.cineca.it
- Password: stblPWdm
- Copy gzipped tar file from here:

  /gpfs/scratch/userinternal/agrottes/CorsoMD-PATC2016/Tutorial1.tar.gz
- Extract archive: tar zxvf Tutorial1.tar.gz
- Run jobs using the scripts provided and compare performances on varying nr. of MPI processes and OpenMP threads