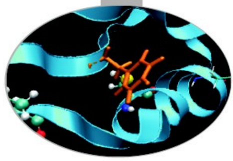
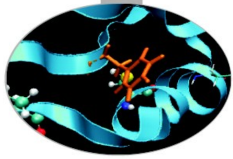


Introduction to Unix Environment: modules, job scripts, PBS

A. Grottesi (CINECA)

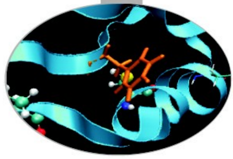
Bologna – PATC 2015



Today you will learn...

- Basic commands for UNIX environment @ CINECA
- How to submit your job to the PBS queueing system on Eurora
- 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



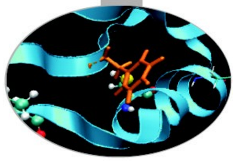
- Please fill out the form on:

  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/password to access HPC systems

How to log in



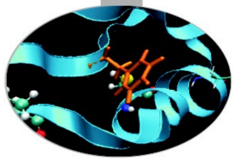
- Establish a ssh connection

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

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

Check the **user guide**! http://www.hpc.cineca.it/content/documentation

Storage and Filesystem



$HOME:

- Permanent, backed-up, and local.
- Quota = 5GB.
- For source code or important input files.

$CINECA_SCRATCH:

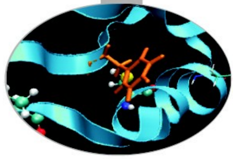
- Large, parallel filesystem (GPFS).
- Temporary (files older than 30 days automatically deleted), no backup.
- No quota. A cleaning procedure for files older than 30 days

$WORK:

- Permanent, backed-up, project specific, 1 Tb quota by default.

More info:
http://www.hpc.cineca.it/content/data-storage-and-filesystems-0

Accounting: saldo



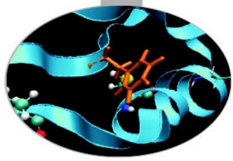
```
[mcestari@node342] (~)
$ saldo -b

---------------------------------------------------------------------------------------------------------
account                start         end        total        localCluster    totConsumed       totConsumed
                                           (local h)   Consumed(local h)      (local h)                 %
---------------------------------------------------------------------------------------------------------
try11_test          20110301    20111201        10000                   0              2               0.0
cin_staff           20110323    20200323    200000000               64581        6689593               3.3
ArpaP_prod          20130130    20131101      1500000                   0              0               0.0
```

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

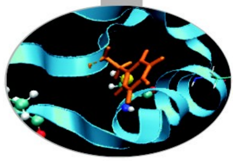


- all the optional software on the system is made available through the **"module" system**
  - provides a way to rationalize software and its env variables

- modules are divided in 3 *profiles*
  - **profile/base** (stable and tested modules)
  - **profile/engineering** (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

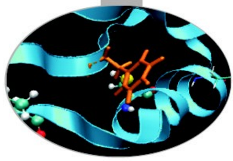


- CINECA’ s work environment is organized in modules, a set of installed libs, tools and applications available for all users.

- “loading” a module means that a series of (useful) shell environment variables wil 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

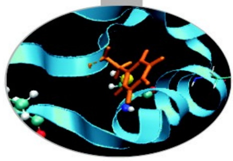
> module help <module_name>

Gets all informations about how to use a specific module

> module purge

Gets rid of all the loaded modules

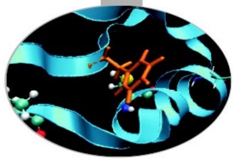
Launching jobs



- It’s time to learn how to prepare a job for its execution. The parallel execution requires a batch script.
- Eurora, 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 -n 16 ./myexe arg_1 arg_2
```

–n is the number of **cores** you want to use

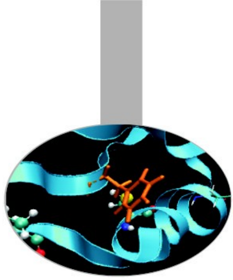
arg_1 arg_2 are the normal arguments of myexe

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:mem=ngpus=2     # resources
#PBS -l walltime=1:00:00                              # 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

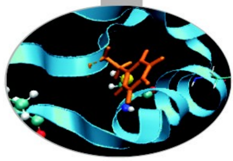
mpiprocs = number of mpi tasks per node/chunk

mem = RAM memory per chunk

ngpus = number of CUDA devices per node

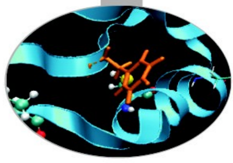
nmics = numer of Intel-Phi devices per node

PBS job script template



```
#!/bin/bash
#PBS -l walltime=2:00:00
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=2:mem=14GB
#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>
```

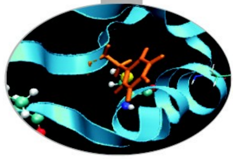
Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

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>
```

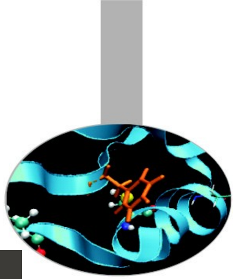
Removes the job from the scheduled jobs by killing it

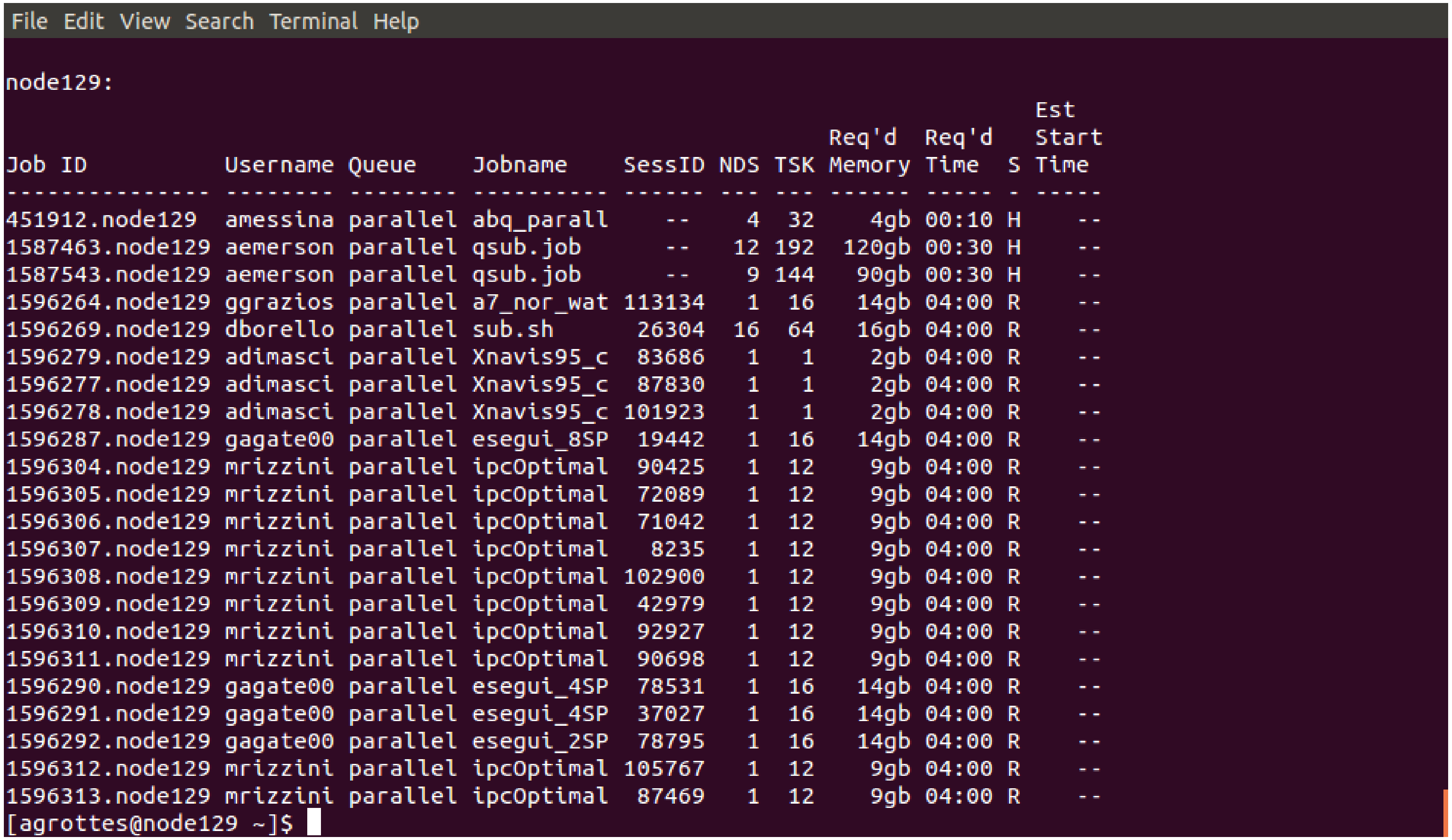
qalter

```
qalter -l <resources> <job_id>
```

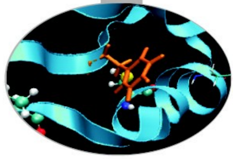
Alter one or more attributes of one or more PBS batch jobs.

PBS commands: qstat



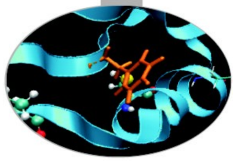


```
node129:
                                                                             Est
                                                            Req'd  Req'd      Start
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time   S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
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 ~]$
```



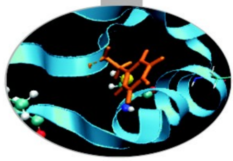
Scripts for running MD codes on Eurora

Gromacs 5.0.4, pure MPI on Eurora



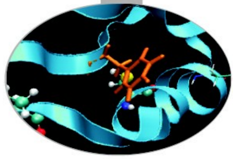
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -q R1660526
#PBS -W group_list=train_cmd32015

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 Eurora



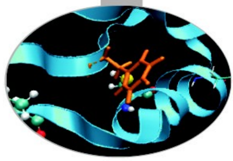
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=2:mpiprocs=2:ngpus=2:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd32015
#PBS -q R1660526
#PBS -W group_list=train_cmd32015

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 2 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



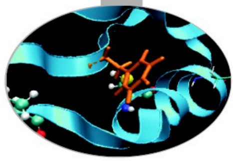
```
#!/bin/bash
#PBS -N MPI-OpenMP-GPU
#PBS -l select=1:ncpus=16:mpiprocs=2:ngpus=2:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd32015
#PBS -q R1660526
#PBS -W group_list=train_cmd32015


# 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 = 2 GPUs
# => set total mpi tasks = 2  (2 per node) and set omp tasks to fill up each node
export OMP_NUM_THREADS=8

mdrun=$(which mdrun_mpi_cuda)
cmd="$mdrun -s topol.tpr -deffnm MPI_OpenMP-GPU -v -maxh 1.0 -gpu_id 01"
mpirun -np 2 $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 <queue>
#PBS -l walltime=1:00:00
#PBS -A train_cmd12015

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
```

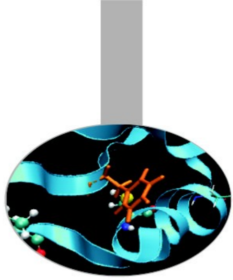
Native execution of codes on Intel Phi



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)

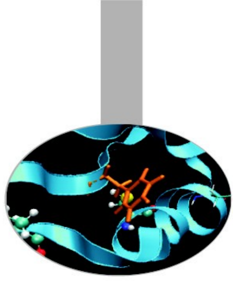
Native execution of codes on Intel Phi



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

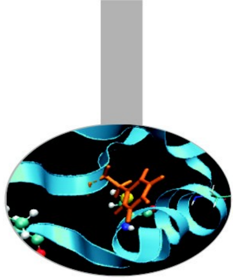


In order to SSH access the mic card you have to create the public key of the Galileo username in your $HOME from login node

http://www.hpc.cineca.it/content/how-connect-public-key

1. ssh-keygen
2. ls /home/<NAME>/.ssh
3. Copy the id_dsa.pub file into the $HOME area of the remote system you want to connect to.
4. cat id_dsa.pub >> $HOME/.ssh/authorized_keys

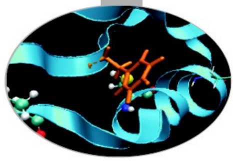
NAMD 2.10 MPI+CUDA on Eurora



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

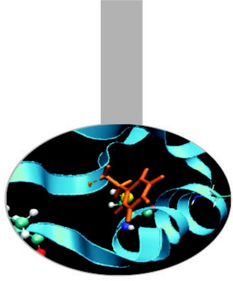
```
#!/bin/bash
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=2:mem=14GB
#PBS -l walltime=0:30:00
#PBS -o namd.out
#PBS -e namd.err
#PBS -A <account_no>
#PBS -q <queue>

cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload namd/2.9

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

mpirun -np 16 $namd +idlepoll md.namd   ==> run CUDA version of NAMD
```

NAMD 2.10 Intel Phi on Eurora



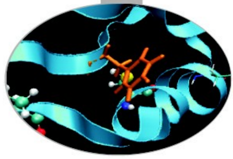
```
>module load autoload namd/2.10
>module help namd/2.10

#!/bin/bash
#PBS -l select=1:ncpus=16:mpiprocs=16:nmics=2:mem=14GB
#PBS -l walltime=0:30:00
#PBS -o namd.out
#PBS -e namd.err
#PBS -A <account_no>
#PBS -q <queue>

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 Eurora (pure MPI version)

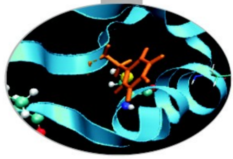


```
#!/bin/bash
#PBS -A <account_no>
#PBS -N amber
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -o job.out
#PBS -q <queue>

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 (Eurora)



```
#!/bin/bash
#PBS -A <account_no>
#PBS -l select=1:ncpus=2:mpiprocs=2:ngpus=2
#PBS -l walltime=1:00:00
#PBS -o job.out
#PBS -q queue

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
```



Scripts for running MD codes on Fermi

Job script: general structure

```
#!/bin/bash
# @ job_name = bgsize.$(jobid)
# @ output = z.$(jobid).out
# @ error  = z.$(jobid).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 02:00:00
# @ notification = never
# @ bg_size = 256
# @ class = keyproject
# @ account_no = cinstaff
# @ restart = no
# @ queue


cd /gpfs/scratch/userinternal/cin0753a/mydir

runjob –ranks-per-node 64 ./program.exe
```

LL keyword

Application block

NAMD template script for Fermi

```
#!/bin/bash
# @ job_name = namd.$(jobid)
# @ output = $(job_name).out
# @ error = $(job_name).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 01:00:00
# @ bg_size = 64
# @ account_no = your_account_name
# @ queue

module load namd/2.9

#launch with 256 processes (grouping 4 processes per node) using multi-thread (16 threads per process)
runjob --rank-per-node 4 : $NAMD_HOME/namd2 +ppn16 input.namd > output.log
```

64*4 = 256 MPI tasks

*16 = 4096 threads

Optimized by IBM namd version: adopts a mixed MPI/OpenMP thread approach for the parallel computation.

The number of MPI process per node are selected with the --ranks-per-node option of LoadLeveler, while the number of OpenMP threads per MPI process with the +ppn flag of namd.

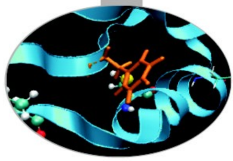
GROMACS template script for Fermi

```
#!/bin/bash
# @ job_name = gromacs.$(jobid)
# @ output = $(jobid).out
# @ error = $(jobid).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 01:00:00
# @ notification = always
# @ bg_size = 64
# @ account_no = <my_account_no>
# @ queue

module load profile/advanced
module load gromacs/5.0.4

# get the path of the mdrun executable for the backend nodes
mdrun=$(which mdrun_bgq)

# add any mdrun options.
# Here we are using 4 OpenMP threads for MPI task.
exe="$mdrun -v -s topol.tpr -ntomp 4"

#launch single precision mdrun on all the back-end nodes
runjob --ranks-per-node 16 --env-all : $exe
```

mixed MPI/OpenMP thread version

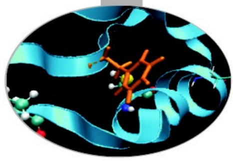
64*16 = 1024 MPI tasks
*4 = 4096 threads

Amber template script for Fermi



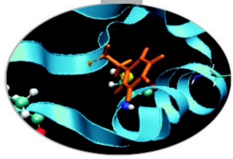
```
#!/bin/bash
# @ job_name = amber.$(jobid)
# @ output = z.$(jobid).out
# @ error = z.$(jobid).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 01:00:00
# @ notification = always
# @ bg_size = 64
# @ account_no = my_account_no
# @ queue

module load amber/12

# get the path of the sander executable for the backend nodes
sander=$(which sander.MPI)

# add any  sander options
exe="$sander -i mdin -o mdout -p prmtop -c inpcrd -r restrt -ref refc -mtmd mtmd -x mdcrd"

#launch sander on all the allocated back-end nodes
#(note the : which must be present with this syntax)
# if you have memory problems reduce the --ranks-per-node option
runjob --ranks-per-node 16 --env-all : $exe
```

Lammps template script for Fermi



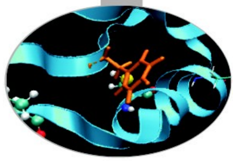
```
#!/bin/bash
# @ job_name = lammps.$(jobid)
# @ output = $(job_name).out
# @ error = $(job_name).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 01:00:00
# @ bg_size = 64
# @ queue
#
module load lammps

# get the path of the lammps executable for the backend nodes
lammps_run=$(which lammps)

# add any lammps command line options
exe="$lammps_run -i inp.file"

#launch lammps with 64 processes on back-end nodes (note the : which must be present)
runjob -n 64 : $exe

# launch MPI+multi-thread lammps version on 64 compute-nodes using 4 threads per process
# lammps_run=$(which lammps_omp)
# exe="$lammps_run -sf omp -i inp.file"
# export OMP_NUM_THREADS=4
# runjob -n 256 -p 4 --exp-env OMP_NUM_THREADS : $exe
```

DL_POLY template script for Fermi



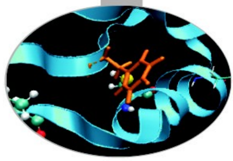
```
#!/bin/bash
# @ job_name = dlpoly.$(jobid)
# @ output = z.$(jobid).out
# @ error = z.$(jobid).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 01:00:00
# @ notification = always
# @ bg_size = 64
# @ account_no = my_account_no
# @ queue


# load module
Module load profile/advanced
module load dl_poly/4.05

# get location of DLPOLY executable
exe=$(which DLPOLY.Z)

# run with 16 tasks/node for a total of 64*16 = 1024 tasks
runjob -n 16 : $exe
```

Tutorial 1: getting familiar with PBS



- Connect to Eurora: ssh username@login.eurora.cineca.it
- Password: prog15ram
- Copy gizipped tar file from here: /gpfs/scratch/userinternal/agrottes/Corsi/November-2015/Tutorial1.tar.gz
- Type: tar zxvf Tutorial1.tar.gz to extract input files and template scripts.
- Run a small run (10000 steps) of a small biological molecule (a 12-mer peptide) using pure MPI, MPI+CUDA and MPI_OpenMP+CUDA scripts provided.
- Compare results with different parallel paradigms (MPI and OpenMP)
- Try optimizing the best combination of MPI/OpenMP ranks to get the best performance.

Tutorial 1: getting familiar with PBS



Run MPI-CUDA and MPI-OpenMP_CUDA jobs on Eurora:

- queue = parallel
- add PBS keyword: #PBS -W group_list=train_cmd22015