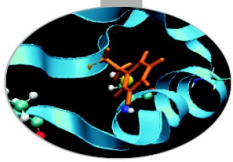
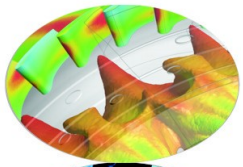
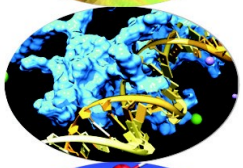
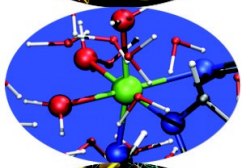
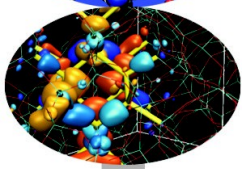
CINECA SCAI
SuperComputing Applications and Innovation



Introduction to CINECA HPC Environment

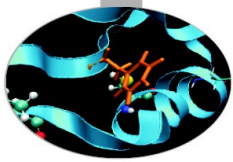
22nd Summer School on Parallel Computing

July 15th 2013

i.baccarelli@cineca.it, m.cestari@cineca.it

CINECA

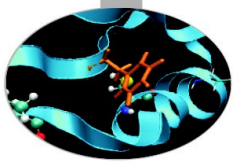
Goals



You will learn:

- after an overview of CINECA HPC platforms
- how to login to the school HPC platform (PLX)
- basic concepts of the system architecture and user environment that directly affects your work
- how to explore and interact with the software installed on the system
- how to submit a job to the system queues

Contents



Systems overview (Fermi, Eurora, PLX)

A first step
- login
- file transfer

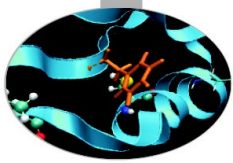
Introduction to the environment
- accounting
- disk systems
- module system

Programming environment
- compilation
- compiling/linking issues

Production environment
- job script
- PBS commands

For further info...
- useful links and documentation

Fermi characteristics



Model: IBM-BlueGene /Q

Architecture: 10 BGQ Frame with 2 MidPlanes each

Front-end Nodes OS: Red-Hat EL 6.2

Compute Node Kernel: lightweight Linux-like kernel

Processor Type: IBM PowerA2, 1.6 GHz

Computing Nodes: 10.240 with 16 cores each

Computing Cores: 163.840

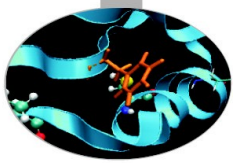
RAM: 16GB / node; 1GB/core

Internal Network: Network interface with 11 links ->5D Torus

Disk Space: more than 2PB of scratch space

Peak Performance: 2.1 PFlop/s

ranked #7 in the top 500 supercomputer sites list (June 2012)
#12 in the top 500 (June 2013)



EURORA characteristics

Model: Eurora prototype

Architecture: Linux Infiniband Cluster

Processors Type:

- Intel Xeon (Eight-Core SandyBridge) E5-2658 2.10 GHz (Compute)
- Intel Xeon (Eight-Core SandyBridge) E5-2687W 3.10 GHz (Compute)
- Intel Xeon (Esa-Core Westmere) E5645 2.4 GHz (Login)

Number of nodes: 64 Compute + 1 Login

Number of cores: 1024 (compute) + 12 (login)

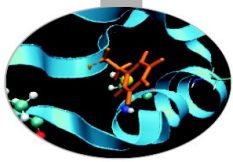
Number of accelerators: 114 nVIDIA Tesla K20 (Kepler) + 14 Intel Xeon Phi (MIC)

RAM: 1.1 TB (16 GB/Compute node + 32GB/Fat node)

OS: RedHat CentOS release 6.3, 64 bit

ranked #1 in the Green 500 chart – The world's most energy-efficient supercomputers (June 2013), 3210 MFlops/s per Watt

PLX characteristics



Model: IBM iDataPlex DX360M3

Architecture: Linux Infiniband Cluster

Processor Type:

- Intel Xeon (Esa-Core Westmere) E5645 2.4 GHz (Compute)
- Intel Xeon (Quad-Core Nehalem) E5530 2.66 GHz (Service and Login)

Number of nodes: 274 Compute + 1 Login + 1 Service + 8 Fat + 6 RVN + 8 Storage + 2 Management

Number of cores: 3288 (Compute)

Number of GPUs: 548 nVIDIA Tesla M2070 + 20 nVIDIA Tesla M2070Q

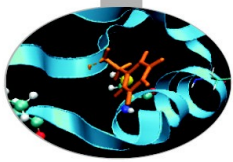
RAM: 14 TB (48 GB/Compute node + 128GB/Fat node)

PLX system performance

Peak performance: **32 Tflops (3288 cores at 2.40GHz)**

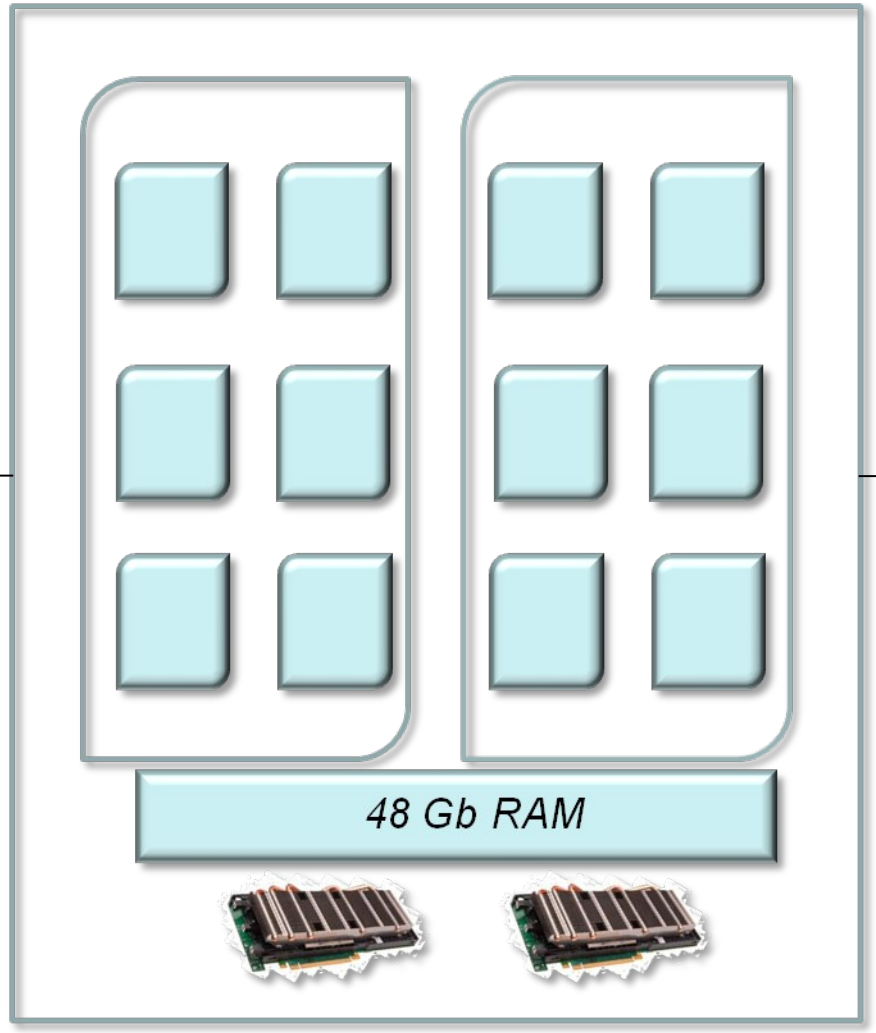
Peak performance: **565 TFlops SP or 283 TFlops DP (548 Nvidia M2070)**

Compute nodes

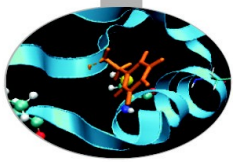


Infiniband connection

Note: the nodes you will use in the queue *private* have 8 cores!



Access credentials



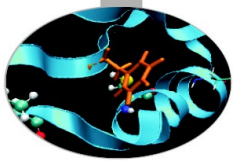
- At the registration desk, you received USERNAME (tyrtytXX) and PASSWORD to log into the laptop in front of you

- You also received the credentials (USERNAME a06trnXX) to log into PLX front-end, login.plx.cineca.it: open a shell (or a putty session on Windows) and establish a secure (ssh) connection to PLX front-end:

```
ssh –X a06trnXX@login.plx.cineca.it
```

This account will grant you access to PLX from July the 15th to September the 15th. After such date the account will expire and the password will be reset.

PLX: how to log in



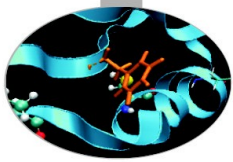
- Establish a ssh connection

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

- Remarks:
  - **ssh** available on all linux distros
  - **Putty** (free) ssh on Windows
  - important messages can be found in the *message of the day*

Check the **user guide**! http://www.hpc.cineca.it/content/ibm-plx-gpu-user-guide-0

PLX: file transfer



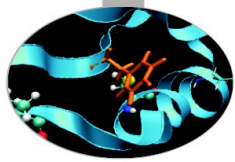
- **sftp / scp** (always available if sshd is running)

```
$ sftp -r <my_dir>  <user>@login.plx.cineca.it:/path/to/
$ scp -r <my_dir>  <user>@login.plx.cineca.it:/path/to/
```

- **rsync**: allows incremental transfer

```
$ rsync -avzr --progress <my_dir> <user>@login.plx.cineca.it:
```

Remark: who uses PLX?

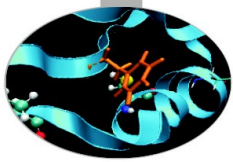


- 8 nodes are **dedicated** to the Summer School via the “private” queue

- But: PLX is a resource **shared** between different type users: academic, industrial, and special agreement users

  ➔ Please be responsible when you use it: if you crash the login node all the users will be affected

Work Environment



$HOME:

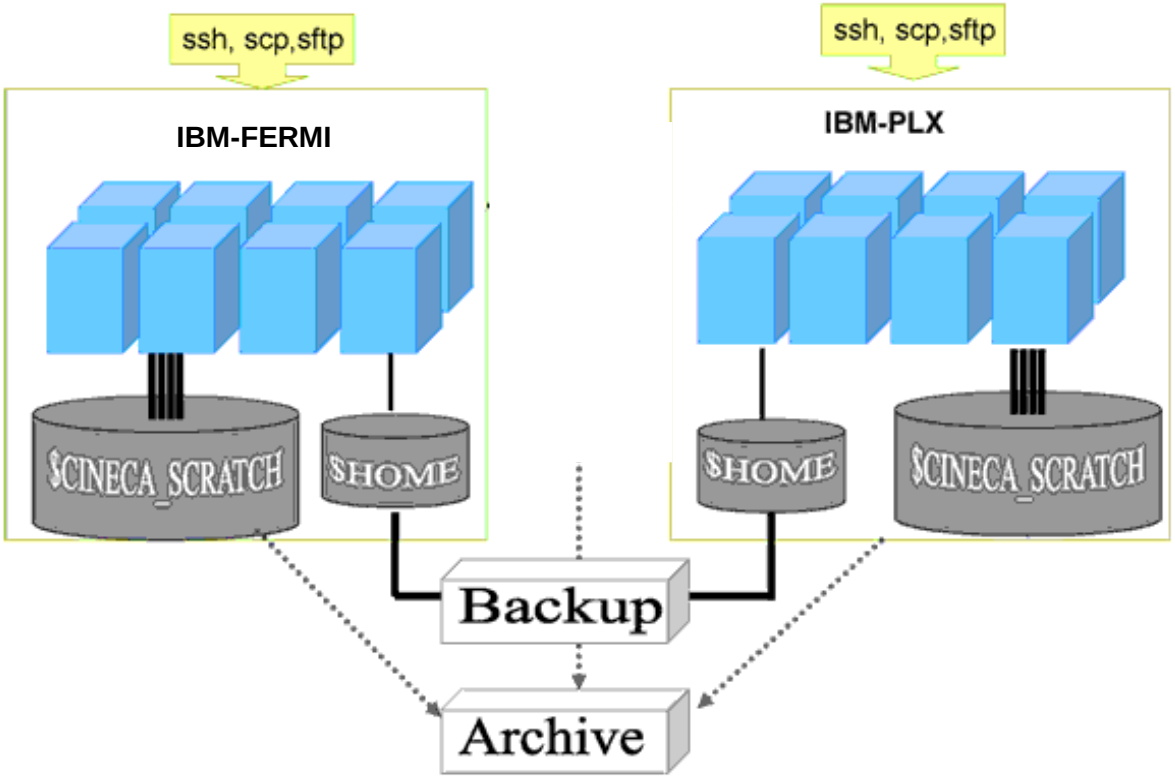
- Permanent, backed-up,and local to PLX.
- Quota = 4GB.
- For source code or important input files.

$CINECA_SCRATCH:

- Large, parallel filesystem (GPFS).
- Temporary (files older than 30 days automatically deleted), no backup.
- No quota. Run your simulations and calculations here.

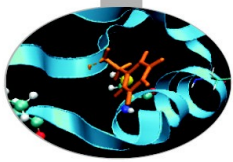
Disks and filesystems

Standard CINECA environment



please use “cindata” command to get info on your disk occupation

Accounting: saldo



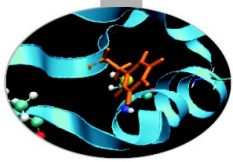
```
[ibaccare@node342~]$ saldo -b

--------------------------------------------------------------------------------------------
account              start         end         total        localCluster   totConsumed     totConsumed
                                              (local h)   Consumed(local h)   (local h)              %
--------------------------------------------------------------------------------------------
cin_staff         20110323    20200323     200000000              172764       7259253            3.6
train_scR2013     20130715    20130914         20000                   0             0            0.0
```

The accounting mechanism 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



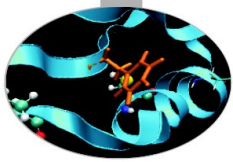
- CINECA's work environment is organized in modules, a set of installed libs, tools and applications available for all users.

- 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/advanced** (software not yet tested or not well optimized)
  - ➔ **profile/engineering** (for industrial users)

- each profile is divided in 4 categories
  - ➔ **compilers** (Intel, GNU, PGI, OpenMPI, IntelMPI, CUDA)
  - ➔ **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - ➔ **tools** (e.g. Totalview, Valgrind, cmake, VNC, python,...)
  - ➔ **applications** (software for chemistry, physics, ...)

Modules

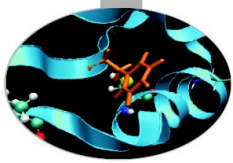


- “loading” a module means that a series of (useful) shell environment variables wil be set
- e.g. after a module is loaded, an environment variable of the form “<MODULENAME>_HOME” is set

```
[ibaccare@node342 ~]$ module load nwchem
WARNING: nwchem/6.1.1 cannot be loaded due to missing prereq
HINT: the following modules must be loaded first: IntelMPI/4.0--binary
[ibaccare@node342 ~]$ module load IntelMPI/4.0--binary
[ibaccare@node342 ~]$ module load nwchem
[ibaccare@node342 ~]$ echo $NWCHEM_HOME
/cineca/prod/applications/nwchem/6.1.1/IntelMPI—4.0—binary/
```

- you can make your life easier by loading the “autoload” module, which takes charge of loading the requested modules

```
[ibaccare@node342 ~]$ module load autoload nwchem
### auto-loading IntelMPI/4.0--binary
[ibaccare@node342 ~]$ echo $NWCHEM_HOME
/cineca/prod/applications/nwchem/6.1.1/IntelMPI—4.0—binary/
```



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

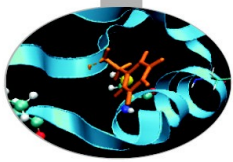
> module list

List all modules you have loaded

> module purge

Gets rid of all the loaded modules

Compiling on PLX



On PLX you can choose between three different compiler families: **gnu**, **intel** and **pgi**

You can take a look at the versions available with "*module av*" and then load the module you want. Defaults are: gnu 4.1.2, intel 11.1, pgi 11.1

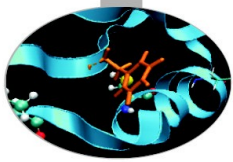
module load intel # loads default intel compilers suite

module load intel/co-2011.6.233--binary # loads specific compilers suite

	GNU	INTEL	PGI
Fortran	gfortran	ifort	pgf77
C	gcc	icc	pgcc
C++	g++	icpc	pgCC

Get a list of the compilers flags with the command *man*

Parallel compiling on PLX



Two families of MPI libraries are available: **openmpi** and **intelmpi**. They provide also the parallel compiler wrappers

There are different versions of openmpi, depending on which compiler has been used for creating them. Default is openmpi/1.4.5--gnu--4.1.2

module load openmpi # loads default openmpi compilers suite

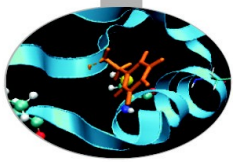
module load openmpi/1.4.5--intel--11.1--binary # loads specific compilers suite

Warning: openmpi needs to be loaded after the corresponding basic compiler suite: you can use “autoload” to load both compilers at the same time

```
[ibaccare@node342 ~]$ module load openmpi
WARNING: openmpi/1.4.5–gnu–4.1.2 cannot be loaded due to missing prereq
HINT: the following modules must be loaded first: gnu/4.1.2
[ibaccare@node342 ~]$ module load autoload openmpi
### auto-loading modules gnu/4.1.2
```

If another type of compiler was previously loaded, you may get a “**conflict error**”. Unload the previous module with “module unload”

Parallel compiling on PLX



	OPENMPI INTELMPI
Fortran	mpif90
C	mpicc
C++	mpiCC

Compiler flags are the same of the basic compiler (since they are basically MPI wrappers of those compilers)

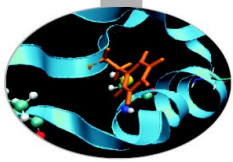
OpenMP is provided with following compiler flags:

gnu: -fopenmp

intel : -openmp

pgi: -mp

Undefined references



- Many compilation errors are due to wrong or incomplete library linking
- (**undefined reference**): don't panic!

- Remember to load your modules (module avail, module load):

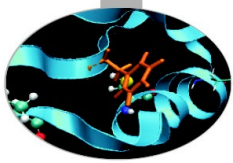
```
module load library/version
```

(fftw/3.2.2—gnu--4.5.2, lapack/3.3.1—intel--co-2011.6.233--binary, ecc.)

- all library paths are in the form $LIBRARY_**LIB** (e.g., $LAPACK_**LIB**…) ;
- include paths are in the form $LIBRARY_**INC**

```
$ module load hdf5
$ ls $HDF5_LIB
libhdf5.a      libhdf5_cpp.la     libhdf5_fortran.la  libhdf5_hl_cpp.a
   libhdf5hl_fortran.a   libhdf5_hl.la  libhdf5.settings libhdf5_cpp.a
libhdf5_fortran.a  libhdf5_hl.a  libhdf5_hl_cpp.la  libhdf5hl_fortran.la
libhdf5.la
```

Undefined references



Use the command "nm" to find the reference and the right library to link:

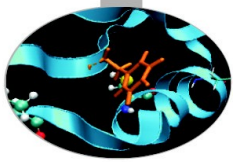
```
$ for i in `ls $HDF5_LIB/*.a` ; do echo $i ; nm $i | grep H5AC_dxpl_id ; done

/cineca/prod/libraries/hdf5/1.8.7_ser/intel--co-2011.6.233--binary/lib/libhdf5.a
                 U H5AC_dxpl_id
                 U H5AC_dxpl_id
000000000000009c D H5AC_dxpl_id
```

2 ways to link a library:

`-L$LIBRARY_LIB -lname` --- or --- `$LIBRARY_LIB/libname.a`

```
1) mpicc_r -I$HDF5_INC input.c -L$HDF5_LIB -lhdf5 -L$SZIP_LIB -lsz -L$ZLIB_LIB -lz
2) mpicc_r -I$HDF5_INC input.c $HDF5_LIB/libhdf5.a $SZiP_LIB/libsz.a $ZLIB_LIB/libz.a
```

Static/Dynamic Linking



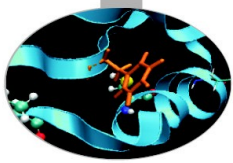
On PLX you can choose between dynamic and static linking (**dynamic** is the **default)**.

- **Static linking** means that the library references are resolved at **compile time**, so the necessary functions and variables are already contained in the executable produced. It means a bigger executable but no need for linking the library paths at runtime.

- **Dynamic linking** means that the library references are resolved at **run time**, so the executable searches for them in the paths provided. It means a lighter executable and no need to recompile the program after every library update, but environment variables have to be defined at runtime (i.e. LD_LIBRARY_PATH)

To enable static linking: -static (gnu), -Bstatic (intel, pgi)

Launching jobs



Now that we have our executable, it's time to learn how to prepare a job for its execution

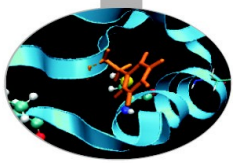
- PLX uses **PBS** scheduler.
- Jobs are submitted with `qsub`
- Submit an "interactive" job via command line:

qsub -I –l select=1:ncpus=1,walltime=00:10:00 -A **train_scR2013** –q **private**

- Submit a job to batch: qsub script.sh

The job script scheme is:

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



PBS keywords

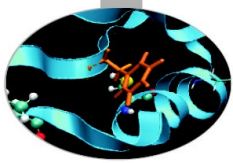
```
#PBS –N jobname   # name of the job
#PBS -o job.out    # output file
#PBS -e job.err    # error file
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=24gb   # resources
#PBS -l walltime=1:00:00       # hh:mm:ss, max 144 hours in queue “private”
#PBS -q private         # chosen queue
#PBS -A <my_account>      # name of the account: train_scR2013
```

select = number of chunk requested

ncpus = number of cpus per chunk requested

mpiprocs = number of mpi tasks per chunk

mem = RAM memory per chunk



PBS Keywords specific for the school

#PBS -A **train_scR2013** # your “account” name

#PBS -q **private** # special queue used in this school

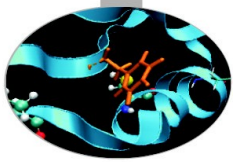
private is a particular queue defined on 8 compute nodes (8 cores each)

train_scR2013 is the cpu-hours budget that you need to set

Remember the how to submit an “interactive” job via command line using qsub flags:

qsub -I –l select=1:ncpus=1,walltime=00:10:00 -A train_scR2013 –q private

Environment setup and execution line



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

mpirun –n 24 ./myexe arg_1 arg_2

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

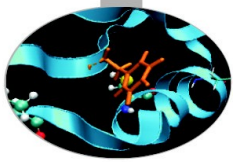
arg_1 arg_2 are the normal arguments of myexe

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.

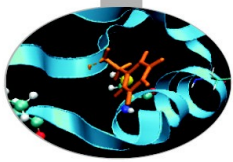
$PBS_O_WORKDIR points to the directory from where you're submitting the job .

PLX job script example



```
#!/bin/bash
##PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=2:mpiprocs=2:mem=12gb,walltime=1:00:00
#PBS -o job.out
#PBS -e job.err
#PBS -q private
#PBS -A train_scR2013
#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 openmpi
module load somelibrary

mpirun ./myprogram < myinput
```

PBS commands



qsub

qsub <job_script>

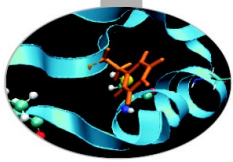
Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

qstat

qstat

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>

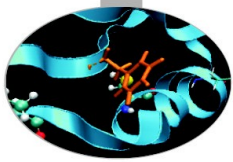
Provides a long list of informations for the job requested.
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

qdel

qdel <job_id>

Removes the job from the scheduled jobs by killing it

PRIVATE queue



$ qstat -Qf private

Queue: private

queue_type = Execution

total_jobs = 0

resources_max.ncpus = 64

resources_max.ngpus = 8

resources_max.walltime = 144:00:00

resources_default.ncpus = 8

resources_default.ngpus = 0

resources_default.place = free:shared

acl_group_enable = True

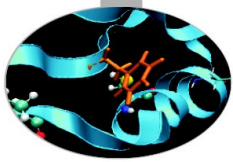
acl_groups = -,a06trn00,+cin_staff,+cinstaff,+corsi,+train_scR2013

enabled = True

started = True

…….

Documentation



Check out the User Guides on our website www.hpc.cineca.it

PLX:

Advanced topic

http://www.hpc.cineca.it/sites/default/files/PBSProUserGuide10.0.pdf