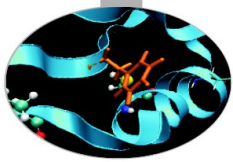
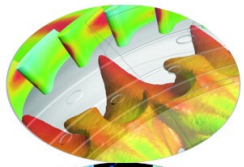
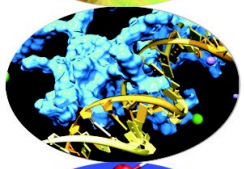
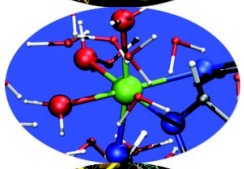
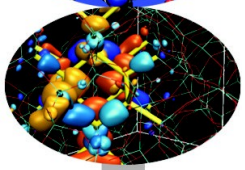
CINECA SCAI
SuperComputing Applications and Innovation



Using the IBM iDataPlex (PLX)

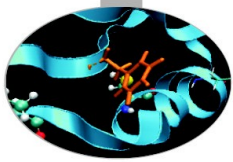
ARPA - Piemonte

02-03 May 2013

m.cestari@cineca.it

CINECA

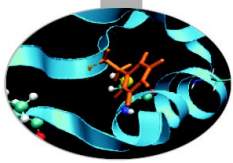
Goals



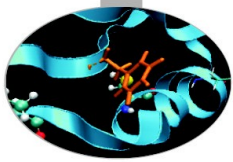
You will learn:

- basic concepts of the system architecture that directly affects your work
- how to explore and interact with the software installed on the system
- how to compile a parallel code and how to fix basic issues
- how to launch a simulation exploiting the computing resources provided by the PLX system

Contents



PLX: get the access credentials



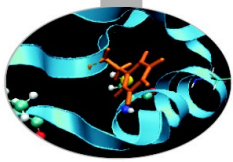
- Please fill out the form on

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

- You'll receive userdb credentials: Then
  - Click on “HPC Access” and follow the 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 ARPA_prod project

- When everything is done an automatic procedure sends you (via 2 separate emails) the username/password to access PLX

PLX: how to log in



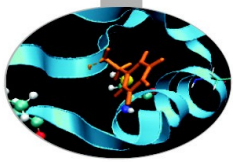
- Establish a ssh connection

  ssh <username>@login.plx.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/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

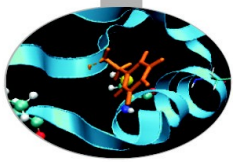
```
$ rsync -avzr --progress <my_dir> <user>@login.plx.cineca.it:
```

- **gridftp**: allows for stream transfer and much more
  (~10x transfer! - available soon)

```
$  globus-url-copy -vb -r -p 16 -sync -sync-level 2
file:/path/to gsiftp://mcestari@gftp-plx.cineca.it:2812/path/to/
```

Check the **user guide**! www.hpc.cineca.it/content/transferdata

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

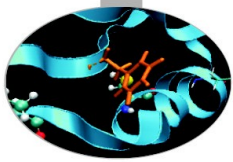
RAM: 14 TB (48 GB/Compute node + 128GB/Fat node)

PLX system performance

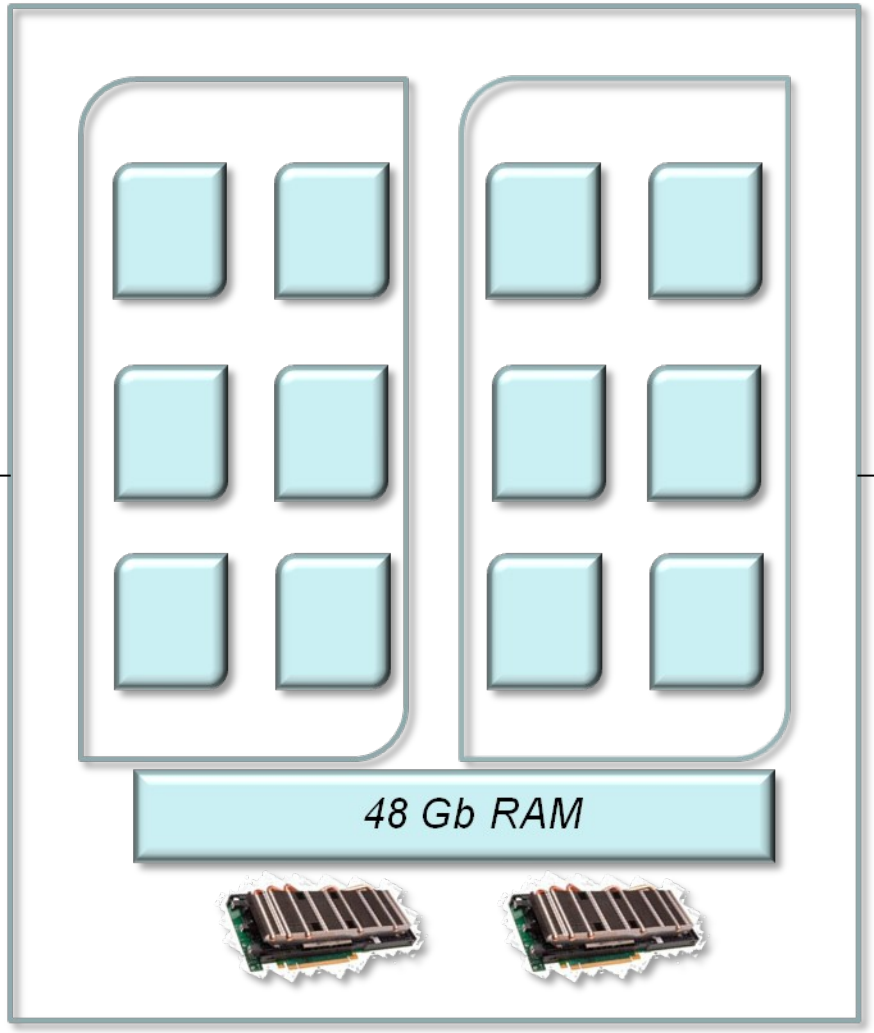
Peak performance: **32 Tflops (3288 cores a 2.40GHz)**

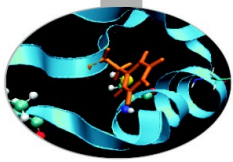
Peak performance: **565 TFlops SP or 283 TFlops DP (548 Nvidia M2070)**

Compute nodes

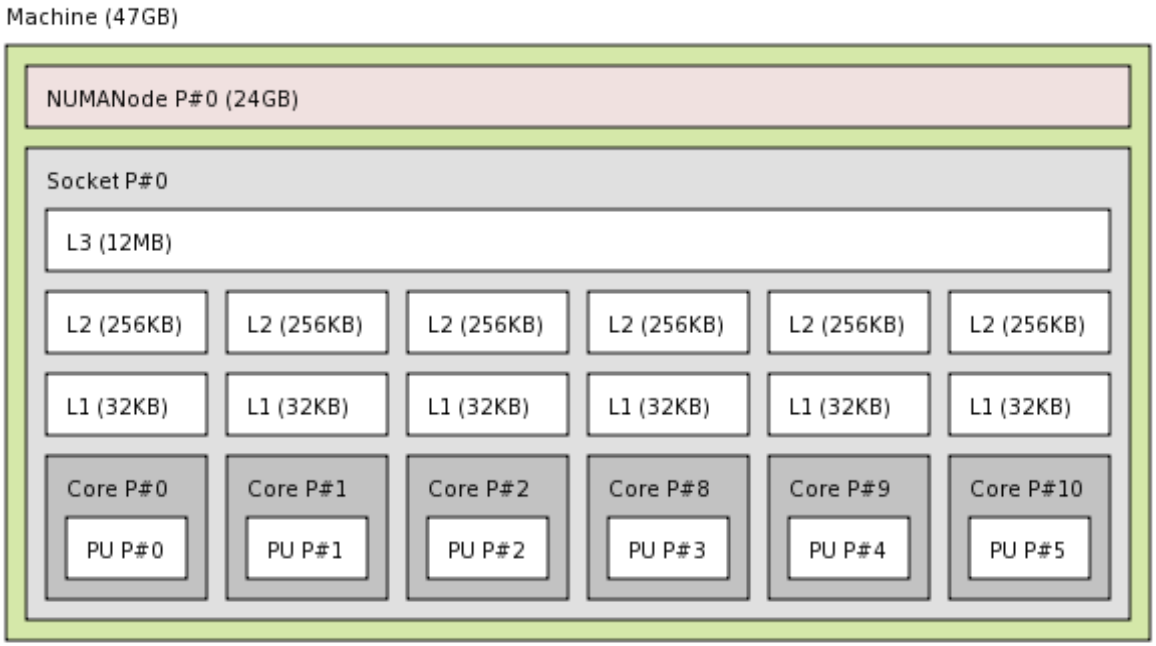


Infiniband connection





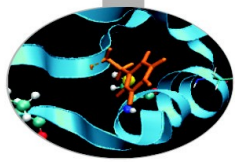
Xeon E5645 – cache hierarchy



To find out **more info** on the cache hierarchy

$ grep . /sys/devices/system/cpu/cpu0/cache/index*/*

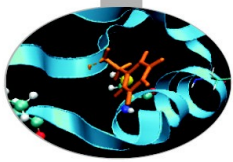
Remark: who uses PLX?



- 7 nodes are **reserved** for ARPA users: none of the other users of the system can launch jobs on those nodes

- PLX is a resource **shared** between different type users: academic, industrial, and special agreement (e.g. ARPA-Piemonte) 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_DATA:

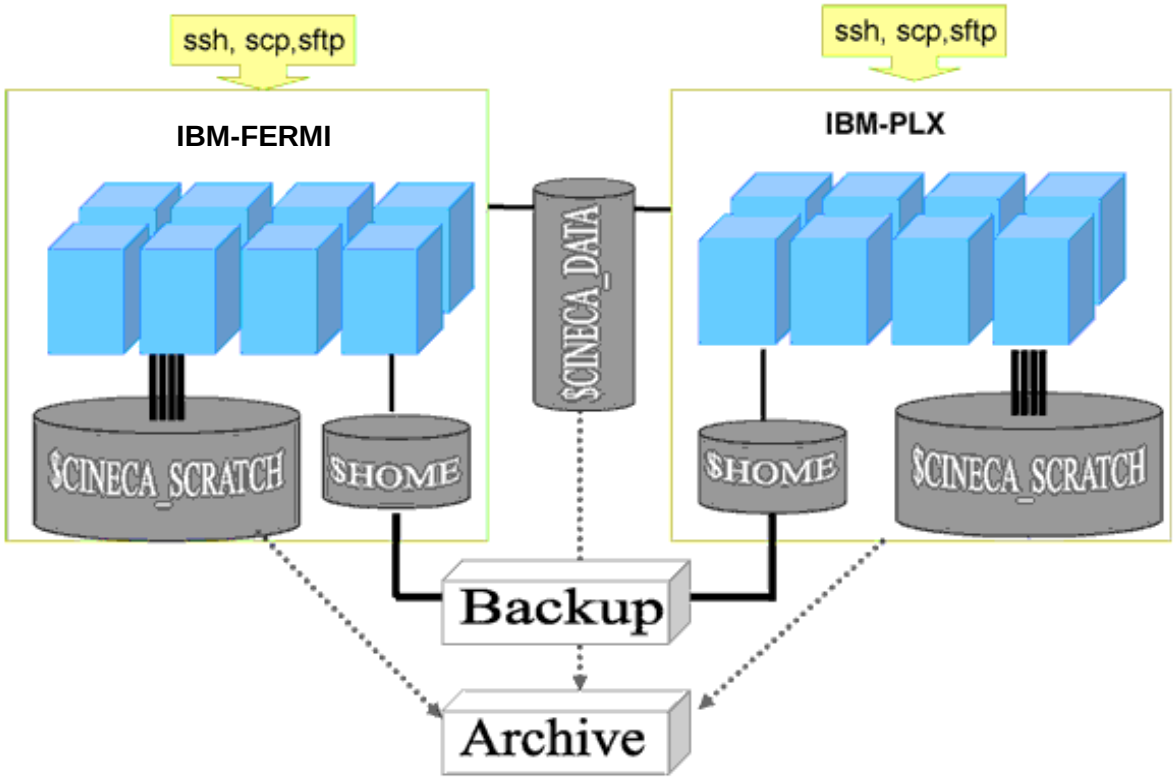
- Permanent, no backup, and shared with other CINECA systems. Mounted only on login nodes (i.e. not visible in normal batch jobs)
- Quota=100Gb can be extended on request.
- Intended as a storage area and file transfer between PLX and other CINECA systems.

$CINECA_SCRATCH:

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

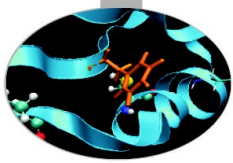
Disks and filesystems

Standard CINECA environment



please use "cindata" command to get info on your disk occupation

Accounting: saldo



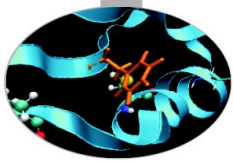
```
[mcestari@node342] (~)
$ saldo -b

--------------------------------------------------------------------------------------------
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.

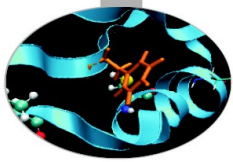
module, my best friend



- 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/front-end** (contains all the base modules plus front-end libs and apps)
  - **profile/advanced** (software not yet tested or not well optimized)

- each profile is divided in 4 categories
  - **compilers** (IBM-xl, GNU)
  - **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - **tools** (e.g. Scalasca, GNU make, VNC, ...)
  - **applications** (software for chemistry, physics, ...)



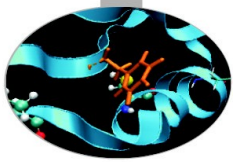
Modules

- 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

```
[amarani0@fen07 ~]$ module load namd
[amarani0@fen07 ~]$ ls $NAMD_HOME
backup  flipbinpdb  flipdcd  namd2  namd2_plumed  namd2_remd  psfgen  sortreplicas
```



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

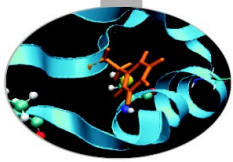
> module (un)load <module_name>
(Un)loads a specific module

> 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

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

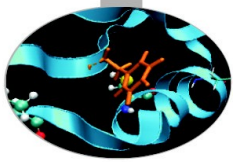
module load intel # loads default intel compilers suite

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/1.4.4--gnu--4.5.2

module load openmpi # loads default openmpi compilers suite

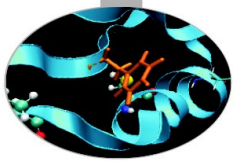
module load openmpi/1.4.5--intel--11.1--binary # loads specific compilers suite

Warning: openmpi needs to be loaded after the corresponding basic compiler suite. You can load both compilers at the same time with "**autoload**"

```
[cin0955a@node342 ~]$ module load openmpi
WARNING: openmpi/1.4.4--gnu--4.5.2 cannot be loaded due to missing prereq.
HINT: the following modules must be loaded first: gnu/4.5.2
[cin0955a@node342 ~]$ module load autoload openmpi
### auto-loading modules gnu/4.5.2
```

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)

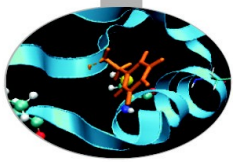
OpenMP is provided with following compiler flags:

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):

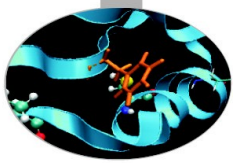
```
module load library/version
```

(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 $LIBRARY_**LIB** ($FFTW_**LIB**, $LAPACK_**LIB** ecc.) ; 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:

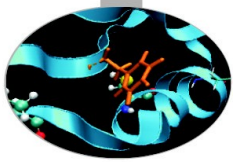
```
$ for i in `ls $HDF5_LIB/*.a` ; do echo $i ; nm $i | grep H5AC_dxpl_id ; done

/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/bibz.a
```

Static/Dynamic Linking



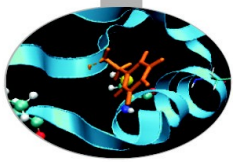
On PLX you can choose between dynamic and static linking (**dynamic** is the **default)**.

- **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)

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 a lot of environment variables to define at runtime.

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



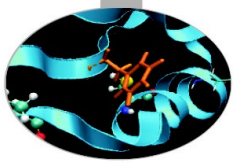
Launching jobs

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

- PLX uses **PBS** scheduler.

- The job script scheme is:

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



PBS keywords

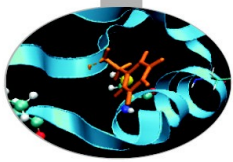
```
#PBS –N jobname   # name of the job
#PBS -o job.out    # output file
#PBS -e job.err    # error file
#PBS -l select=1:ncpus=12:mpiprocs=12:mem=47gb   # resources
#PBS -l walltime=1:00:00       # hh:mm:ss, max 48h for ARPA-P
#PBS -q privarpap         # chosen queue
#PBS -A <my_account>      # name of the account
```

select = number of chunk requested

ncpus = number of cpus per chunk requested

mpiprocs = number of mpi tasks per chunk

mem = RAM memory per chunk



PBS Keywords specific for ARPA

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

#PBS -q **privarpap** # special queue reserved for you

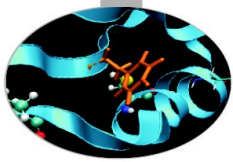
privarpap is a particular queue composed by 8 compute nodes reserved for ARPA Piemonte users

“**privarpap**” queue is a particular queue composed by 8 compute nodes reserved for ARPA Piemonte users

ArpaP_prod is the

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

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

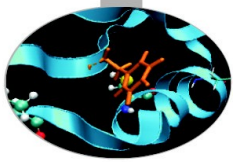
In order to use mpirun, openmpi (or IntelMPI) has to be loaded. Also, if you linked dynamically, you have to remember to load every library module you need.

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.

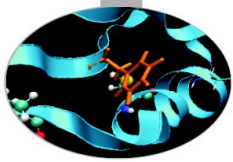
$PBS_O_WORKDIR points to the directory from where you're submitting the job .

PLX job script example



```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=12:mpiprocs=12:mem=46gb
#PBS -o job.out
#PBS -e job.err
#PBS -q privarpap
#PBS -A ArpaP_prod
#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>

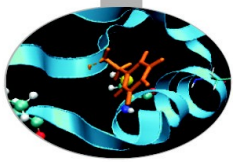
Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

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>

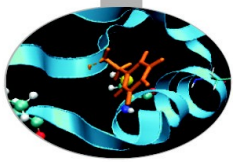
Provides a long list of informations for the job requested.
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

qdel

qdel <job_id>

Removes the job from the scheduled jobs by killing it

ARPA queue

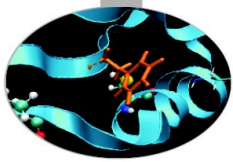


$ qstat -Qf privarpap

Queue: privarpap

queue_type = Execution
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Begun:0
acl_user_enable = False
resources_max.ncpus = 84
resources_max.ngpus = 14
resources_max.Qlist = arpap
resources_max.walltime = 48:00:00
resources_min.Qlist = arpap
resources_default.ncpus = 1
resources_default.ngpus = 0
resources_default.place = free:shared
resources_default.Qlist = arpap
acl_group_enable = True
acl_groups = -,arpap_prod
default_chunk.Qlist = arpap
enabled = True
started = True

Documentation



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

PLX:

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

http://www.hpc.cineca.it/content/batch-scheduler-pbs-0

Advanced topic

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