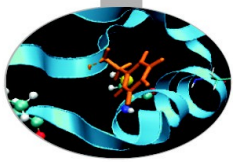
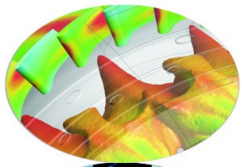
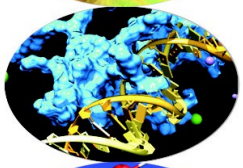
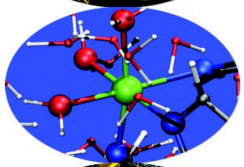
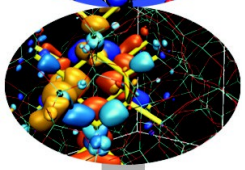
CINECA SCAI
SuperComputing Applications and Innovation



Introduction to PLX working environment

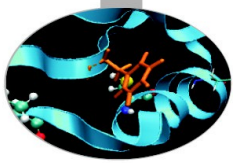





Introduction to Parallel Computing with MPI and OpenMP

9-10-11 December 2014

a.marani@cineca.it

CINECA

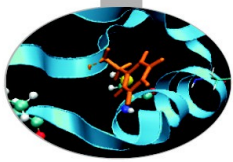


PLX

Architecture: Linux Infiniband Cluster
Processor: Intel Xeon (Esa-Core Westmere) E5645 2.4 GHz
Number of processors (cores): 3288
Number of nodes: 274 (12 cores per node)
RAM: 14 TB (4 GB/core)
Interconnection network: Infiniband
Number of GPUs: 548 (2 per node)
Operative system: Linux
Peak performance: 32 TFlop/s (CPU); 565 TFlop/s (GPU)
Compilers: Fortran, C, C++
Parallel libraries: MPI,OpenMP

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

WORK ENVIRONMENT



Once you're logged on PLX, you are on your **home** space.

It is best suited for **programming** environment (compilation, small debugging sessions…)

Space available: 4 GB (PLX)

Environment variable: $HOME

Another space you can access to is your **scratch** space.

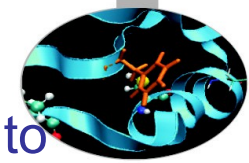
It is best suited for **production** environment (launch your jobs from there)

Space available: 32 TB (PLX)

Environment variable: $CINECA_SCRATCH

Use the command "cindata" for a quick briefing about your space occupancy

ACCOUNTING



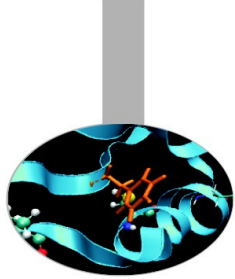
As an user, you have access to a limited number of CPU hours to spend. They are not assigned to users, but to **projects** and are shared between the users who are working on the same project (i.e. your research partners). Such projects are called **accounts** and are a different concept from your username.

You can check the state of your account with the command "*saldo –b*", which tells you how many CPU hours you have already consumed for each account you're assigned at (a more detailed report is provided by "*saldo –r*".

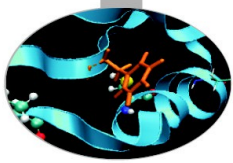
```
[amarani0@fen08 ~]$ saldo -b

--------------------------------------------------------------------------------------------
account            start        end          total       localCluster   totConsumed     totConsumed
                                             (local h)   Consumed(local h)   (local h)          %
--------------------------------------------------------------------------------------------
cin_staff          20110323     20200323     1000000000         30365762     30527993          3.1
cin_totview        20130123     20130213       50000                0            0           0.0
train_sc32013      20130211     20130411     1250000            87458        87458           7.0
train_cn112013     20130311     20130411      100000                0            0           0.0
```

ACCOUNTING



The account provided for this course is "**train_cmpB2014**" (you have to specify it on your job scripts). It expires in one week and is shared between all the students; there are plenty of hours for everybody, but don't waste them!



MODULES

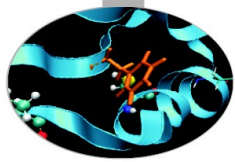
CINECA's work environment is organized with modules, a set of installed tools and applications available for all users.

"loading" a module means defining all the environment variables that point to the path of what you have loaded.

After a module is loaded, the environment variable is set of the form "MODULENAME_HOME"

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

MODULE COMMANDS

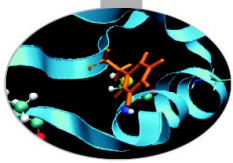


>module available (or just “> module av”)
Shows the full list of the modules available in the profile you’re into, divided by: environment, libraries, compilers, tools, applications

> module load <module_name>
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



COMPILING ON PLX

In 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 12.8

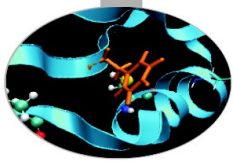
module load intel # loads default intel compilers suite

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

Compiler's name	GNU	INTEL	PGI
Fortran	gfortran	ifortran	pgf77
C	gcc	icc	pgcc
C++	g++	icpc	pgCC

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

PARALLEL COMPILING ON PLX



For parallel programming, two families of compilers are available: **openmpi** (recommended) and **intelMPI** .

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

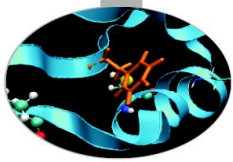
module load openmpi/1.4.5--intel--11.1--binary # loads specific compilers suite

Warning: openmpi needs to be loaded after the corresponding basic compiler suite. You can 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



Compiler's name	**OPENMPI INTELMPI**
Fortran	mpif90
C	mpicc
C++	mpiCC

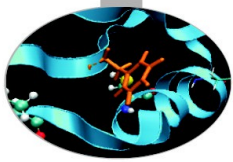
Compiler flags are the same as the basic compiler (since they are basically MPI wrappers of those compilers)

OpenMP is provided with the thread-safe suffix "_r" (ex: mpif90_r) and the following compiler flags:

gnu: -fopenmp

intel : -openmp

pgi: -mp

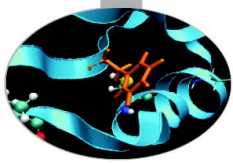


LAUNCHING JOBS

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

PLX uses a completely different scheduler with its own syntax, called **PBS**. The job script scheme remains the same:

- #!/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=1   #resources requested*
#PBS -l walltime=1:00:00       #max 24h, depending on the queue
#PBS -q parallel         #queue desired
#PBS -A <my_account>      #name of the account
```

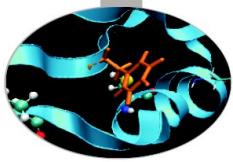
*: select = number of nodes requested

ncpus = number of cpus per node requested

mpiprocs = number of mpi tasks per node

for pure MPI jobs, ncpus =mpiprocs. For OpenMP jobs, mpiprocs < ncpus

LL KEYWORDS SPECIFIC FOR THE COURSE

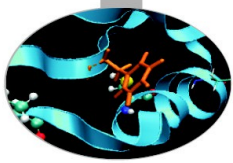


```
#PBS -A train_cmpB2014    # your account name
#PBS -q private     # special queue reserved for you
#PBS -W group_list=train_cmpB2014   # needed for entering in private queue
```

“private” queue is a particular queue composed by 8 nodes reserved for internal staff and course students. Each nodes has no more than 8 CPUs (while regular nodes have 12)

In order to grant fast runs to all the students, we ask you to not launch jobs too big (you won’t need them, anyways). Please don’t request more than 1 node at a time!

ENVIRONMENT SETUP AND EXECUTION LINE



The command used to launch a parallel application is mpirun:

mpirun –n 14 ./myexe

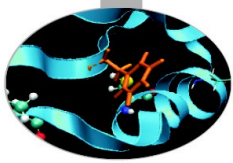
–n is the number of cores you want to use.

The “difficult part” here is setting the environment…

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.

The environment setting usually start with “cd $PBS_O_WORKDIR”. That’s because by default you are launching on your home space and may not find the executable you want to launch.

$PBS_O_WORKDIR points at the folder you’re submitting the job from.



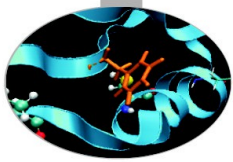
PLX JOB SCRIPT EXAMPLE

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=8:mpiprocs=8
#PBS -o job.out
#PBS -e job.err
#PBS -q private
#PBS -A train_cmpB2014
#PBS –W group_list=train_cmpB2014

cd $PBS_O_WORKDIR
module load autoload openmpi

mpirun ./myprogram
```

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)

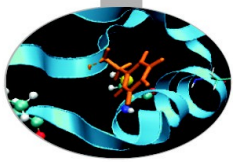
qstat

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 -f <job_id>

Provides a long list of informations for the job requested.

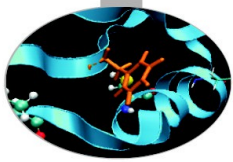
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, you made an error on the job script, you will learn that the job won't ever start

qdel

qdel <job_id>

Removes the job from the scheduler, killing it

JOB CLASSES



Let's suppose you are now a regular HPC user. You won't have access to the "private" queue: how can you launch jobs then?

You have to modify your jobscript by changing the "PBS –q private" keyword with something else: you will be able to submit your jobs, but as a regular user (so expect long waiting times)

The queue you're going into is the one you ask (it has to be specified!):

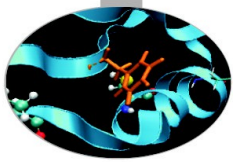
debug: max nodes= 2, wall_clock_time <= 00:30:00

parallel: max nodes=44, wall_clock_time <= 06:00:00

longpar: max nodes=22, wall_clock_time <=24:00:00

You don't need the PBS –W keyword anymore

USEFUL 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