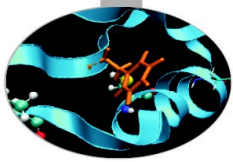
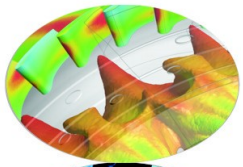
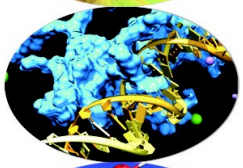
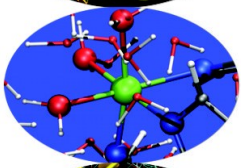
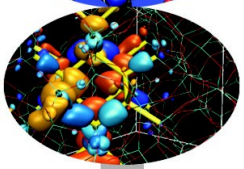
CINECA SCAI
SuperComputing Applications and Innovation



Introduction to HPC Numerical libraries at CINECA

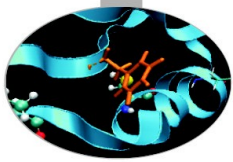





HPC Numerical Libraries

10-11-12 March 2014

n.spallanzani@cineca.it

CINECA



WELCOME!!

The goal of this course is to show you how to get advantage of some of the most important numerical libraries for improving the performance of your HPC applications. We will focus on:

FFTW

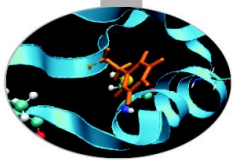
FFTW, a subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST)

ScaLAPACK

A good number of libraries for Linear Algebra operations, including BLAS, LAPACK, SCALAPACK and MAGMA

PETSc

PETSc, a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations



ABOUT THIS LECTURE

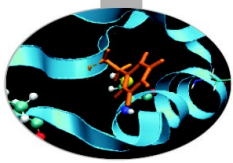
This first lecture won't be about numerical libraries…

Its purpose is to teach you the very basics of how to interact with CINECA's HPC cluster, where exercises will take place.

You will learn how to access to our system, how to compile, how to launch batch jobs, and everything you need in order to complete the exercises succesfully

…don't worry, it won't last long!! ;-)

WORK ENVIRONMENT



Workstation: User → corsi Password → corsi_2013!

ssh username@login.plx.cineca.it

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

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

Environment variable: $HOME

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

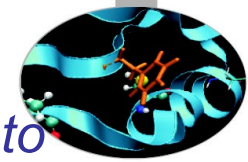
*It is best suited for **production** environment (launch your jobs from there)*

Environment variable: $CINECA_SCRATCH

*WARNING: is active a **cleaning procedure**, that deletes your files older than 30 days!*

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.*

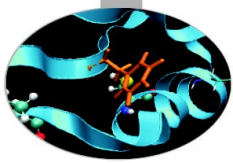
You can check the state of your account with the command "*saldo –b*", which tells you how many CPU hours you have already consumed for each account you're assigned at

(a more detailed report is provided by "*saldo –r*").

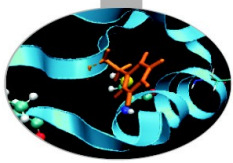
```
[amarani0@fen08 ~]$ saldo -b

--------------------------------------------------------------------------------------------
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_cnl12013     20130311     20130411        100000                 0            0          0.0
```

ACCOUNTING



The account provided for this course is “**train_cnl2014**” (you have to specify it on your job scripts). It expires in one month and is shared between all the students; there are plenty of hours for everybody, but don’t waste them!



MODULES

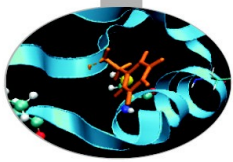
CINECA's work environment is organized with modules, a set of installed tools and applications available for all users.

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

After a module is loaded, an 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

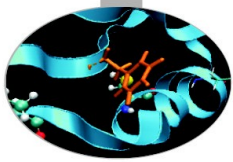
> module show <module_name>

Shows the environment variables set by a specific module

> module help <module_name>

Gets all informations about how to use a specific module

LIBRARY MODULES



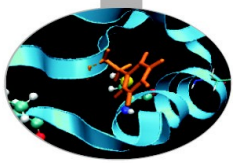
The Numerical Libraries you will learn about and use during the course are also available via module system

```
------------ /cineca/prod/modulefiles/base/libraries ------------
blas/2007--bgq-xl--1.0(default)       libjpeg/8d--bgq-gnu--4.4.6
essl/5.1                              mass/7.3--bgq-xl--1.0
fftw/2.1.5--bgq-xl--1.0               mpi4py/1.3--bgq-gnu--4.4.6
fftw/3.3.2--bgq-xl--1.0               netcdf/4.1.3--bgq-xl--1.0
fftw/3.3.3--bgq-xl--1.0(default)      numpy/1.6.2--bgq-gnu--4.4.6
gsl/1.15--bgq-xl--1.0                 papi/4.4.0--bgq-gnu--4.4.6
hdf5/1.8.9_par--bgq-xl--1.0           petsc/3.3-p2--bgq-xl--1.0
hdf5/1.8.9_ser--bgq-xl--1.0           scalapack/2.0.2--bgq-xl--1.0(default)
lapack/3.4.1--bgq-xl--1.0(default)    szip/2.1--bgq-xl--1.0
libint/2.0--bgq-xl--1.0(default)      zlib/1.2.7--bgq-gnu--4.4.6
```

Once loaded, they set the environment variable LIBRARYNAME_LIB .

If needed, there is also LIBRARYNAME_INC for the header files.

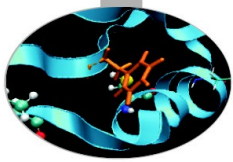
More on that during the course…



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



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

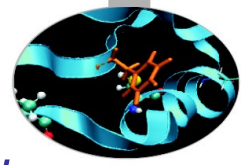
module load intel # loads default intel compilers suite

module load intel/cs-xe-2012--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 flags 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/1.4.4--gnu--4.5.2

module load autoload openmpi # loads default openmpi compilers suite

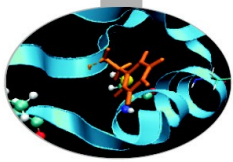
module load openmpi/1.4.5--gnu--4.1.2 # loads specific compilers suite

Warning: mpi compiler 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

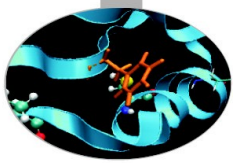
Compiler flags are the same as the basic compiler (since they are basically MPI wrappers of those compilers)

OpenMP is provided with the following compiler flags:

gnu: -fopenmp

intel : -openmp

pgi: -mp



COMPILING WITH LIBRARIES

Once you have loaded the proper library module, specify its linking by adding a reference in the compiling command.

Two ways to link a library:

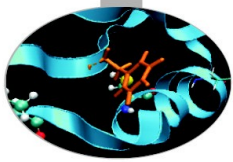
-L$LIBRARY_LIB **-l**name *– or –* **-L**$LIBRARY_LIB/**lib**name**.a**

For some libraries, it may be necessary to include the header path

-I$LIBRARY_INC

```
$ mpicc -I$HDF5_INC input.c -L$HDF5_LIB -lhdf5 \
        -L$SZIP_LIB -lsz -LZLIB_LIB -lz
$ mpicc -I$HDF5_INC input.c -L$HDF5_LIB/libhdf5.a \
        -L$SZIP_LIB/libsz.a -L$ZLIB_LIB/libz.a
```

COMPILING WITH LIBRARIES

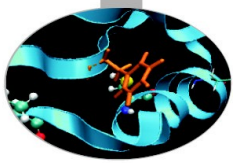


*PLX lets you choose between **static** and **dynamic linking**, with the latter one as a default.*

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 need a lot of environment variables to define at runtime.

For enabling static linking: -static (gnu), -intel-static (intel), -Bstatic (pgi)



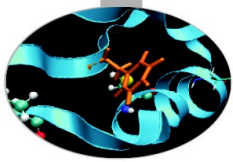
LAUNCHING JOBS

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

PLX uses a scheduler called **PBS**.

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=2   #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
```

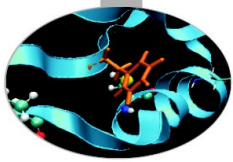
* select = number of chunks (not exactly the nodes) requested

ncpus = number of cpus per chunk requested

mpiprocs = number of mpi tasks per chunk

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

KEYWORDS SPECIFIC FOR THE COURSE



#PBS -A train_cnl2014 # your account name

#PBS -q private # special queue reserved for you

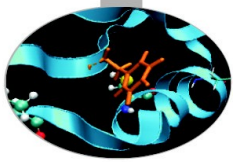
#PBS -W group_list=train_cnl2014 # needed for entering in private queue

"private" queue is a particular queue composed by 8 nodes reserved for internal staff and course students.

"private" nodes have only 8 cores per node.

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

ENVIRONMENT SETUP AND EXECUTION LINE



The command that "split" the executable on the processes is mpirun:

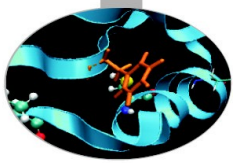
```
mpirun -n 8 ./myexe arg_1 arg_2
```

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

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.



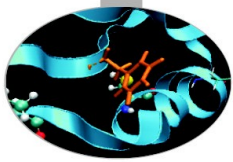
PLX JOB SCRIPT EXAMPLE

```
#!/bin/bash
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -o job.out
#PBS -e job.err
#PBS -q private
#PBS -A train_cnl2014
#PBS –W group_list=train_cnl2014

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)

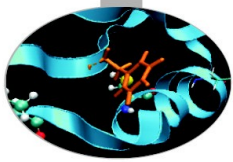
qstat

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 -f <job_id>

Provides a long list of informations for the job requested.

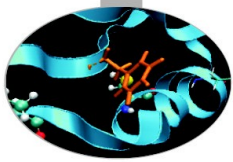
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

qdel

qdel <job_id>

Removes the job from the scheduler, killing it

JOB CLASSES



After the end of the course, you won't be able to use the private queue anymore: how can you launch jobs then?

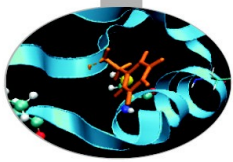
You have to modify your jobscript by changing the "#PBS –q private" keyword with something else: you will be able to submit your jobs, but as a regular user (so expect long waiting times)

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

FERMI:

http://www.hpc.cineca.it/content/ibm-fermi-user-guide

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

EURORA:

http://www.hpc.cineca.it/content/eurora-user-guide

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