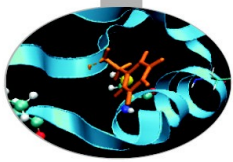
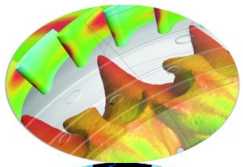
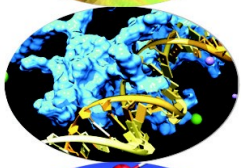
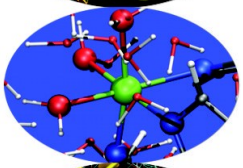
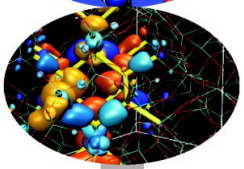
CINECA SCAI
SuperComputing Applications and Innovation



Introduction to HPC Numerical libraries on FERMI and PLX

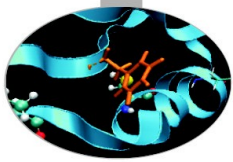





HPC Numerical Libraries

11-12-13 March 2013

a.marani@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

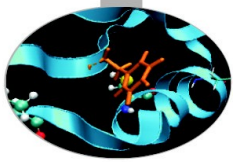
FFTW, a subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST)

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



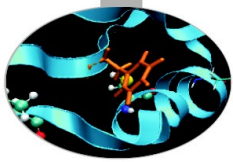
ABOUT THIS LECTURE

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 clusters, 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!! ;-)



WE APOLOGIZE…

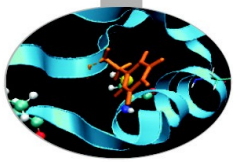
Some weeks ago we asked you to complete our UserDb registration form in order to grant you access to **FERMI**, our main HPC cluster…

…however, within the last week some urgent massive jobs made the cluster really hard to be used for exercises. Your jobs may need to remain in idle (waiting) state for a lot of time before their actual start!!

So we decided to let you have access to **PLX**, a smaller but faster cluster in terms of waiting time. FFTW exercises must be conducted on FERMI, so we are going to learn how to use **both** clusters (maybe in two different lectures)

So let's start with some concept that is common to both systems, and then we will move to more cluster specific informations!

WORK ENVIRONMENT



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

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

Space available: 50 GB (FERMI) – 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)

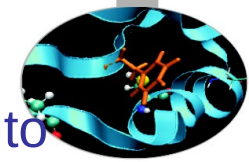
Space available: UNLIMITED (FERMI) – 32 TB (PLX)

Environment variable: $CINECA_SCRATCH

WARNING: On PLX 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



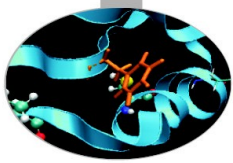
As an user, you have access to a limited number of CPU hours to spend. They are not assigned to users, but to **projects** and are shared between the users who are working on the same project (i.e. your research partners). Such projects are called **accounts** and are a different concept from your username.

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*").

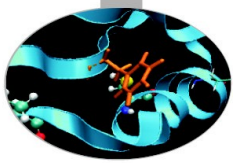
```
[amarani0@fen08 ~]$ saldo -b

--------------------------------------------------------------------------------------------
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



Between PLX and FERMI there is a ratio of 1:5 CPU hours, which means that every hour you spend on PLX is equal to 5 hours spent on FERMI (that's because of the different architecture of the two systems)

The account provided for this course is "**train_cnl12013**" (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

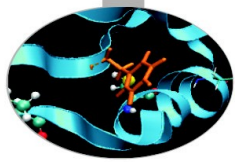
CINECA's work environment is organized with modules, a set of installed tools and applications available for all users.

"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



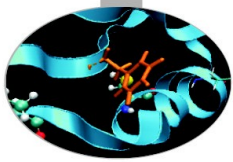
>module available (or just “> module av”)
Shows the full list of the modules available in the profile you’re into, divided by: environment, libraries, compilers, tools, applications

> 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



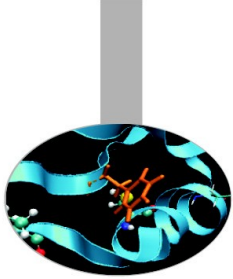
The Numerical Libraries you will learn about and use during the course are also available via module system

```
------------ /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…

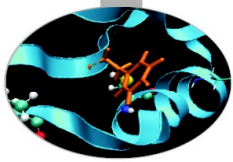


FERMI

Architecture: BlueGene/Q
Processor: IBM PowerA2, 1.6 GHz
Number of processors (cores): 163840
Number of nodes: 10240 (16 cores per node)
RAM: 160 TB (16 GB/core)
Interconnection network: Internal (5D torus)
Disk space: 2 PB
Power consumption: ~1 MW
Operative system: Linux (on surface)
Peak performance: 2 PFlop/s
Compilers: Fortran, C, C++
Parallel libraries: MPI,OpenMP

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

FERMI IN TOP500



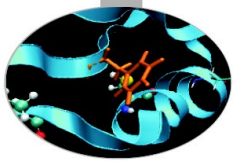
TOP500 SUPERCOMPUTER SITES®

Top500 is a ranking of the most powerful HPC clusters of the World, updated twice a year

www.top500.org

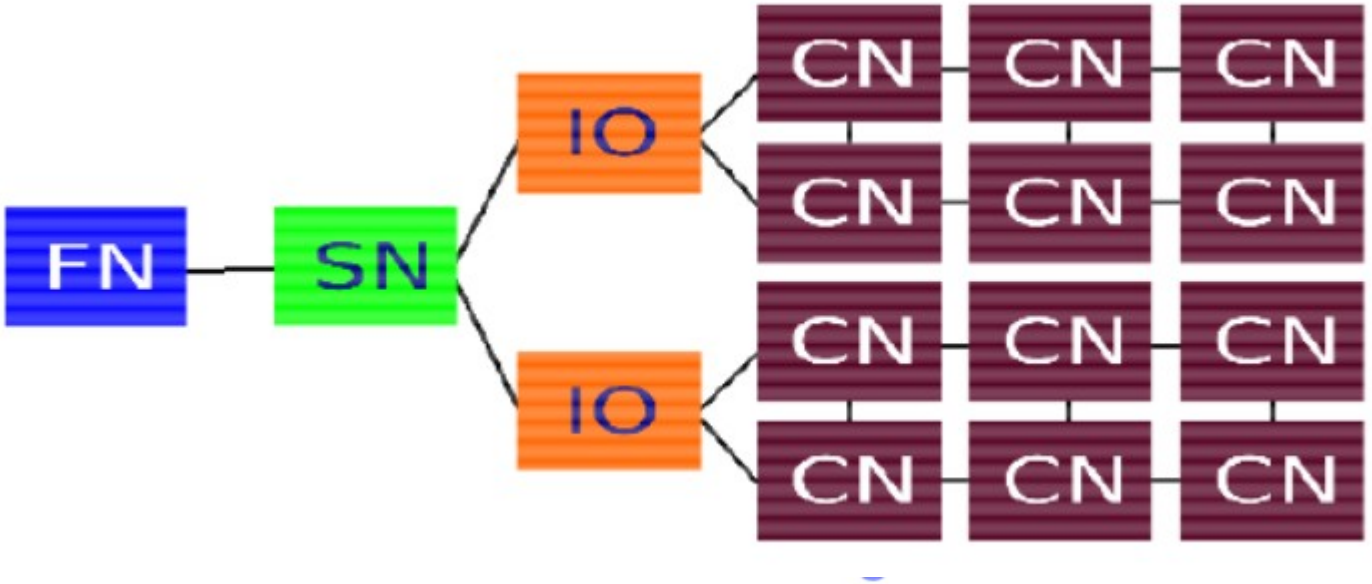
Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
2	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	16324.8	20132.7	7890
3	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
		...				
8	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT	186368	2566.0	4701.0	4040
9	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	163840	1725.5	2097.2	822
10	IBM Development Engineering United States	**DARPA Trial Subset** - Power 775, POWER7 8C 3.836GHz, Custom Interconnect IBM	63360	1515.0	1944.4	3576

FERMI is the 9° most powerful supercomputer in the World! (3° in Europe)

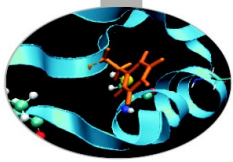


FERMI ARCHITECTURE

This is how FERMI is hierarchically organized:

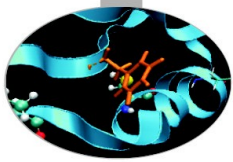


- **Front-end nodes (FN)**, dedicated for user's to login, compile programs, submit jobs, query job status, debug applications
- **Service nodes (SN)**, perform system management services, create and monitoring processes, initialize and monitor hardware, configure partitions, control jobs, store statistics
- **I/O nodes (IO)**, provide a number of OS services, such as files, sockets, process management, debugging
- **Compute nodes (CN)**, run user application, limited OS services



SOME CONSEQUENCES…

- Front-end nodes and Compute nodes are basically of a different architecture, they even have different OS
- You can ssh only on front-end nodes, it's impossible to access directly to compute nodes
- Applications have to be compiled differently depending on which type of nodes you want to execute your program (*cross-compiling*)
- When you want to launch an application on compiling nodes, you have to allocate some resources via batch submission
- Since there is 1 I/O node each 64 or 128 compute nodes, there is a minimum number of nodes you can allocate at a time



COMPILING ON FERMI

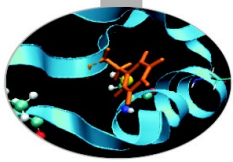
In order to compile parallel programs (as in our case) we need to use compiler developed for back-end (compute) nodes.

You can get advantage of those compilers by loading the proper module:

module load bgq-xl or *module load bgq-gnu*

	Back-end Compilers	
	XL family	GNU family
C	bgxlc, mpixlc_r	gcc, mpicc
C++	bgxlc++, mpixlcxx	g++. mpicxx
Fortran	bgxlf,bgxlf90,... mpixlf90,...	gfortran, mpif90

XL compilers family is recommended on FERMI, since it is developed specifically for IBM architectures, like BG/Q



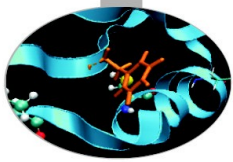
COMPILING ON FERMI

Serial compiling: use the compilers prefixed with “bg”
(*bgxlf, bgxlc, bgxlc++*)

Parallel compiling: use the compilers prefixed with “mp”
(*mpixlf77, mpixlc, mpixlcxx*)

OpenMP compiling: use the thread-safe compilers suffixed with “_r”
(*mpixlf77_r, bgxlc_r, …*) and the optimization flag
–qsmp=omp:noauto

Get a full list of the compiler flags (optimization, debugging, profiling,…) by typing:
man <compiler name>
With the compiler in serial version (es: *man bgxlc*)



COMPILING WITH LIBRARIES

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

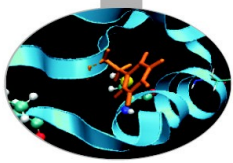
2 ways to link a library:

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

```
1) mpixlc_r -I$HDF5_INC input.c -L$HDF5_LIB -lhdf5 -L$SZIP_LIB -lsz -L$ZLIB_LIB -lz
2) mpixlc_r -I$HDF5_INC input.c $HDF5_LIB/libhdfc5.a $SZiP_LIB/libsz.a
```

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

-I$LIBRARY_INC



UNDEFINED REFERENCES

Sometimes your compilation goes wrong because of a similar error:

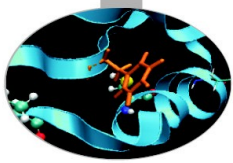
test.o:(.text+0xb8): undefined reference to `H5Z_xform_copy'

It means that you are not linking the correct library. Luckily, there is a "magic formula" for finding it:

> for i in `ls $HDF5_LIB/*.a` ; do echo $i & nm $i | grep H5Z_xform_copy ; done

```
/cineca/prod/libraries/hdf5/1.8.9_ser/bgq-xl--1.0/lib/libhdf5.a
                 U H5Z_xform_copy
0000000000000168 D H5Z_xform_copy
0000000000000150 d H5Z_xform_copy_tree
[1]+  Done                    echo $i
```

Now you know what is the library to link!



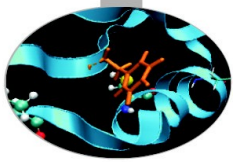
LAUNCHING JOBS

So let's say you have compiled your executable and you want to launch it…

The question is…**HOW TO DO THAT????**

Since we want to execute parallel programs, we have to learn how to get access to back-end nodes

This can be done by writing a small batch script that will be Submitted to a scheduler called **LoadLeveler**



LOADLEVELER BATCH SCRIPT

A LoadLeveler batch script is composed by **four** parts:

1) **Bash interpreter**

#!/bin/bash

2) **LoadLeveler keywords** (more on that later)

 # @ …

 # @ …

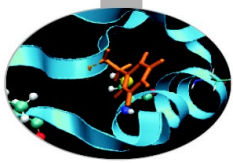
3) **Variables initialization**

export WORK_DIR=…

module load somelibrary

4) **Execution line** (more on that later)

runjob <runjob_options> : <executable> <arguments>

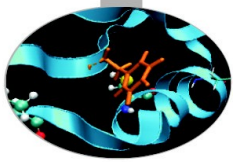


LL KEYWORDS

```
# @ job_name = check
# @ output = $(job_name).$(jobid).out
# @ error = $(job_name).$(jobid).err
# @ environment = COPY_ALL  #export all variables from your submission shell
# @ job_type = bluegene
# @ wall_clock_limit = 10:00:00      #execution time h:m:s, up to 24h
# @ bg_size =  64      # compute nodes number
# @ notification = always|never|start|complete|error
# @ notify_user = <email_address>
# @ account_no = <budget_name>      #saldo –b
# @ queue
```

Highlighted are the mandatory keywords, the others are highly suggested

LL KEYWORDS SPECIFIC FOR THE COURSE



@ **wall_clock_limit** = 00:10:00 #exercises are short, and the lower time you ask, the sooner your job starts

@ **bg_size** = 128 # no less (it won't start), no more (waste of nodes)!

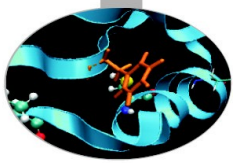
@ **account_no** = train_cnl12013 #your account for the course

@ **class** = training #special high priority class reserved for you

@ **queue**

With great power comes great responsability!!!

FERMI is overbooked at the moment…please use the training class **only** for jobs related to exercises!!!



EXECUTION LINE

Your executable is launched on the compute nodes via the command “runjob”, that you can set in two ways:

1) Use “:” and provide executable infos how you’re used to
runjob : ./exe_name arg_1 arg_2

2) Use specific runjob flags

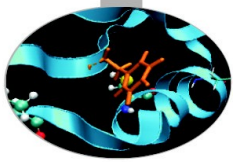
--exe Path name for the executable to run

runjob --exe ./exe_name

--args Arguments for the executable specified by --exe

runjob --exe ./exe_name --args arg_1 --args arg_2

EXECUTION LINE: MPI TASKS SETTING



--ranks-per-node (-p) Number of ranks (MPI task) per compute node. Valid values are 1, 2, 4, 8, 16, 32 and 64 (default=depending on the tasks requested)

bg_size = 64

runjob --ranks-per-node 1 : ./exe <options> #64 nodes used, 1 task per node

runjob --ranks-per-node 4 : ./exe <options> #64 nodes used, 4 tasks per node

--np (-n) Number of ranks (MPI task) in the entire job (default=max)

bg_size = 64

runjob --np 64 -- ranks-per-node 1: ./exe <options> #64 tasks, 1 per node

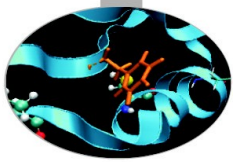
runjob --np 256 -- ranks-per-node 4: ./exe <options> #256 tasks, 4 per node

runjob --np 200 -- ranks-per-node 4: ./exe <options> #200 tasks, 4 per node until all tasks are allocated

runjob --np 1 --ranks-per-node 1: ./exe <options> # serial job

Formula: np <= bg_size*ranks-per-node

EXECUTION LINE: ENVIRONMENT VARIABLES

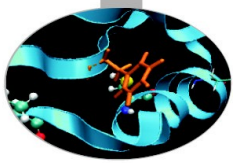


--envs Sets the environment variables for exporting them on the compute nodes

```
#MPI/OpenMP job (16 threads for each MPI task)
runjob -n 64 --ranks-per-node 1 --envs OMP_NUM_THREADS = 16 : ./exe
```

--exp-env Exports an environment variable from the current environment to the job

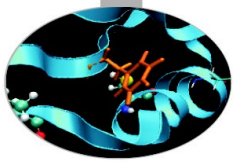
```
export OMP_NUM_THREADS = 16
runjob -n 64 --ranks-per-node 1  --exp-env OMP_NUM_THREADS : ./exe
```



FERMI JOB SCRIPT EXAMPLE

```
#!/bin/bash
# @ job_type = bluegene
# @ job_name = example
# @ comment = "BGQ Example Job"
# @ output = $(job_name).$(jobid).out
# @ error = $(job_name).$(jobid).out
# @ environment = COPY_ALL
# @ wall_clock_limit = 00:10:00
# @ bg_size = 128
# @ account_no = train_cnl12013
# @ class = training
# @ queue

export EXE=$CINECA_SCRATCH/.../my_program

runjob --np 512 --ranks-per-node 16 --exe $EXE --args input.inp
```

LOADLEVELER COMMANDS



Your job script is ready! How to launch it?

llsubmit

llsubmit <job_script>

Your job will be submitted to the LL scheduler and executed when there will be nodes availble (according to your priority)

llq

llq -u $USER

Shows the list of all your scheduled jobs, along with their status (idle, running, closing,…)

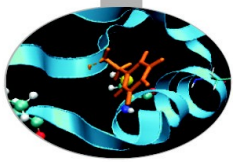
Also, shows you the job id required for other llq options

llq -s <job_id>

Provides information on why a selected list of jobs remain in the

NotQueued, Idle, or Deferred state.

LOADLEVELER COMMANDS



llq -l <job_id>

Provides a long list of informations for the job requested.

In particular you'll be notified about the bgsize you requested and the real bgsize allocated:

.....

.....

BG Size Requested: 1024
BG Size Allocated: 1024
BG Shape Requested:
BG Shape Allocated: 1x1x1x2
BG Connectivity Requested: Mesh
BG Connectivity Allocated: Torus Torus Torus Torus

.....

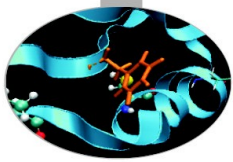
.....

llcancel

llcancel <job_id>

Removes the job from the scheduler, killing it

JOB CLASSES



After the end of the course, class training will be disabled: how can you launch jobs then?

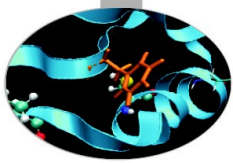
You have to modify your jobscript by removing the “class = training” keyword: you will be able to submit your jobs, but as a regular user (so expect long waiting times)

The class you’re going into depends on the resources you asked:

debug: bg_size=64, wall_clock_time <= 00:30:00

longdebug: bg_size=64, wall_clock_time > 00:30:00 (up to 24h)

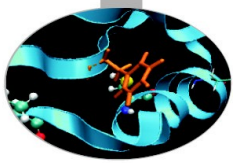
parallel: bg_size>64 (valid values: 128,256,512,1024,2048. The bigger the number, the longer the waiting time)



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



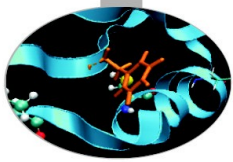
PLX vs. FERMI

The architecture of PLX is pretty different from FERMI because it is designed for commercial users with not-so-scalable applications that need a better performance of the single CPU and a bigger RAM memory.

Accessing on PLX compute nodes is less restrictive and not as complicated as FERMI, but it still has to be scheduled by a batch script. However, there is no need for cross-compiling (applications can be run both interactively and via batch job)

Being an older cluster, it has a larger number of modules installed, thus providing a more complete (but more confusing) work environment

COMPILING ON PLX



In PLX there are no XL compilers, but 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

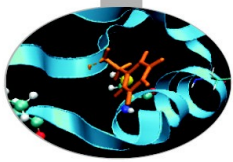
module load intel # loads default intel compilers suite

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

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 openmpi # loads default openmpi compilers suite

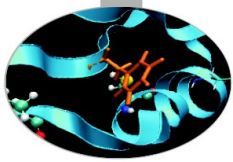
module load openmpi/1.4.5--intel--11.1--binary # loads specific compilers suite

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)

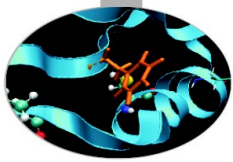
OpenMP is provided with the thread-safe suffix "_r" (ex: mpif90_r) and the following compiler flags:

gnu: -fopenmp

intel : -openmp

pgi: -mp

COMPILING WITH LIBRARIES

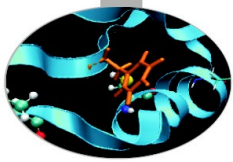


Linking libraries on PLX works exactly as on FERMI (load the library module, use –I and –L, watch out for undefined references…). However, where FERMI allowed only **static linking**, PLX lets you choose between static and **dynamic linking**, with the latter one as a default.

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 a lot of environment variables to define at runtime.

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

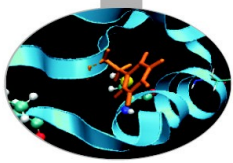


LAUNCHING JOBS

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

PLX uses a 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
```

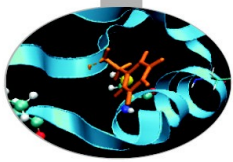
*: select = number of nodes requested

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

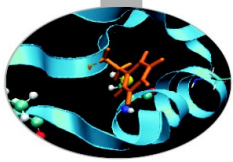


```
#PBS -A train_cnl12013    # your account name
#PBS -q private     # special queue reserved for you
#PBS -W group_list=train_cnl12013   # needed for entering in private queue
```

“private” queue is a particular queue composed by 4 nodes reserved for internal staff and course students

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 runjob is here replaced by mpirun:

mpirun –n 14 ./myexe arg_1 arg_2

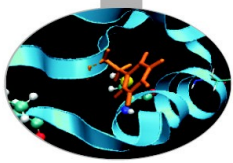
–n is the number of cores you want to use.

It is way easier to setup than runjob! 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.

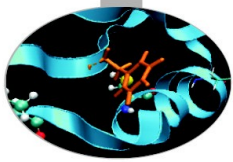
$PBS_O_WORKDIR points at the folder you’re submitting the job from.



PLX JOB SCRIPT EXAMPLE

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=12:mpiprocs=12
#PBS -o job.out
#PBS -e job.err
#PBS -q private
#PBS -A train_cnl12013
#PBS –W group_list=train_cnl12013

cd $PBS_O_WORKDIR
module load autoload openmpi
module load somelibrary

mpirun ./myprogram < myinput
```

PBS COMMANDS



Being a different scheduler, of course the commands for operating with PBS are different than LL…

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)

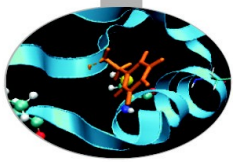
qstat

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.

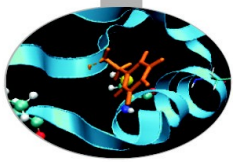
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, you made an error on the job script, you will learn that the job won't ever start

qdel

qdel <job_id>

Removes the job from the scheduler, killing it

JOB CLASSES



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!):

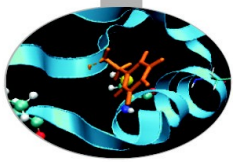
debug: max nodes= 2, wall_clock_time <= 00:30:00

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

FERMI:

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

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

PLX:

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

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