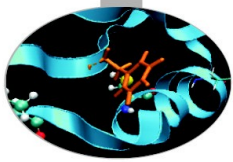
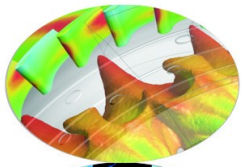
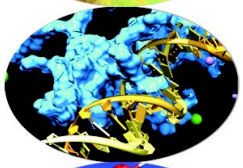
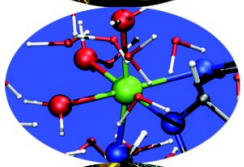
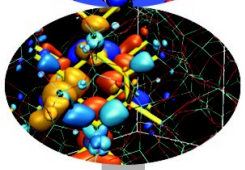
CINECA SCAI
SuperComputing Applications and Innovation



Production environment on FERMI

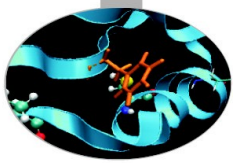





Introduction to the FERMI Blue Gene/Q, for users and developers

18 March 2013

a.marani@cineca.it
silvia.giuliani@cineca.it

CINECA



PRODUCTION TIME!!

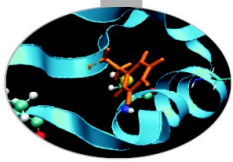
So let's say you have compiled your executable and you want to launch it…

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

Before you do that, let's take a look at your operational space…

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

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.

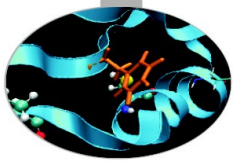
It is best suited for **production** environment (launch your jobs from there)

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



MODULE PROFILES

> module load <profile_name>

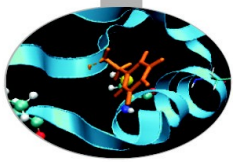
Available modules (“module av profile”):

- **profile/base (default)**: it contains the application modules compiled for back-end nodes and ready to be used

- **profile/front-end**: it contains the applications modules compiled for front-end nodes and ready to be used

- **profile/advanced**. Testing profile. It contains the applicantions modules that have to be tested yet. Usable but not guaranteed

APPLICATION MODULES

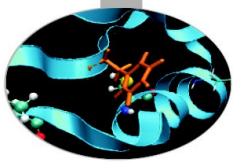


>module available (or just "> module av")
Shows the full list of the modules available in the profile you're into, divided by: environment, libraries, compilers, tools, applications

Below is the list of the application modules available on profile/base, updated to 17th march, 2013

---------- /cineca/prod/modulefiles/base/applications --------

abinit/6.12.3	crystal09/1.01	pluto/4.0
amber/12(default)	crystal09/2.0.1 (default)	qe/5.0bgq
bigdft/1.6.0	dl_poly/4.03(default)	siesta/3.1
cp2k/2.3(default)	gromacs/4.5.5 (default)	siesta/3.1-TS
cpmd/3.15.3_hfx(default)	lammps/20120816	vasp/5.2.12
cpmd/v3.15.3	namd/2.9	vasp/5.3.2



MODULE COMMANDS

> module load <module_name>

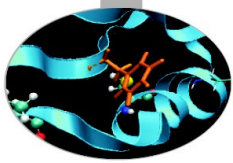
Loads a specific module

> 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



EXECUTION MODALITIES

- Via command line

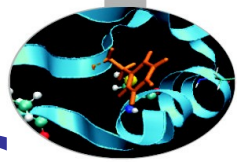
  >./myexe

On Front-end nodes only

- Via batch

  >llsubmit job.cmd

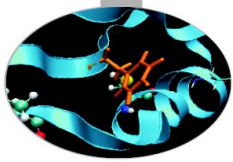
On Front-end and Back-end nodes



EXECUTION: FRONT-END NODES

- **Pre and Post** processing
- **Data transfer**
- **Serial** execution (1 core)
- Executables compiled with serial **FE compilers**
  - >front-end-gnu/4.4.6
  - >front-end-xl/1.0
- **Command line** execution (10 min)
- **Batch execution** (up to 6 h)

BATCH EXECUTABLE: FRONT-END NODES

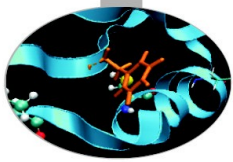


USER EXECUTABLE

>edit job.cmd

- **Shell** interpreter path
  - #!/bin/bash
- **Load Leveler Scheduler** Keywords (we'll check them later ;-))
  - # @
  - # @
  - # @
  - ……
- **Variables** initialization
- **Execution** line
  - ./myexe <options>

BATCH EXECUTABLE: FRONT-END NODES



MODULE EXECUTABLE

- **Shell** interpreter path

  #!/bin/bash

- **Load Leveler Scheduler** Keywords (we'll check them later ;-))

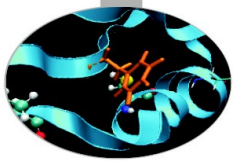
  # @
  # @
  # @
  .....

- **Variables** initialization

module load profile/front-end
module load <module_name>

Execution line

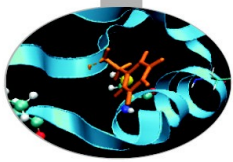
  myexe <options>



BATCH EXECUTABLE: FRONT-END NODES

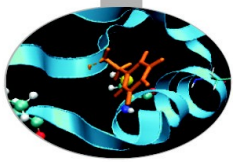
LL KEYWORDS

```
# @ job_name = serial.$(jobid)
# @ output = $(job_name).out
# @ error = $(job_name).err
# @ wall_clock_limit = 0:10:00 # h:m:s
execution time up to 6 hours
# @ class = serial
# @ queue
```



EXECUTION: BACK-END NODES

- **Serial** (WARNING: 64 compute nodes are still required) and **Parallel** execution
- Executable compiled with serial and parallel **BE compilers**
  - >bgq-gnu/4.4.6
  - >bgq-xl/1.0
- NO **command line** execution
- **Batch** execution (from 64 compute nodes up to 2048 compute nodes, wall clock time up to 24 h)
- **Runjob** command
  - >runjob <options>
  - >man runjob



BATCH EXECUTABLE: BACK-END NODES

- USER EXECUTABLE
  - **Shell** interpreter path

    #!/bin/bash
  - **Load Leveler Scheduler** Keywords

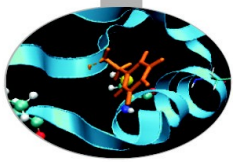
    # @

    # @

    # @

    ……
  - **Variables** inizialization
  - **Execution** line

    >runjob <runjob_options> : ./myexe <myexe_options>



BATCH EXECUTABLE: BACK-END NODES

- MODULE EXECUTABLE
  - **Shell** interpreter path

    #!/bin/bash
  - **Load Leveler Scheduler** Keywords

    # @

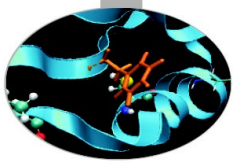
    # @

    # @

    .....
  - **Variables** inizialization

    module load <module_name>
  - **Execution** line

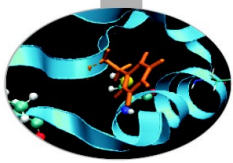
    >runjob <runjob_options> : $MODULE_HOME/bin/exe <exe_options>



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



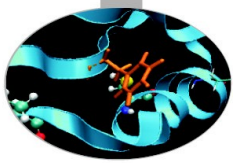
LL KEYWORDS - TOPOLOGY

#@ bg_shape = MD(A)xMD(B)xMD(C)xMD(D)
#midplanes number in the A,B,C,D dimensions

@ bg_rotate = **true**|false

@ bg_connectivity = torus|**mesh**|either|

Xa Xb Xc Xd #type of connectivity



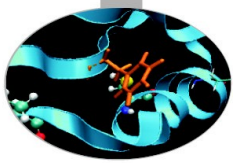
LL KEYWORDS - BG_SIZE

@ bg_connectivity = Mesh #default

@ bg_size = number of compute nodes

- **for requests <= 1midplane** (512 compute nodes)
  bg_size = 64| 128| 256| 512

- **for requests > 1midplane**
  bg_size = (512)X2 | (512)X3 | (512)X4 | (512)X5 | (512)X6| (512)X8 | (512)X10 | (512)X12 | (512)X16



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

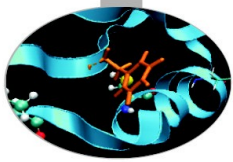
--exe Path name for the executable to run

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

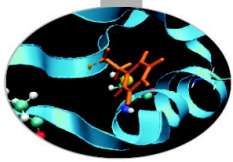
runjob --np 64 -- ranks-per-node 1: ./exe <options> #64 tasks, 1 per node

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

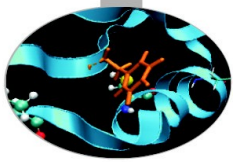
--envs Sets the environment variables for exporting them on the compute nodes

```
#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
```

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,…)

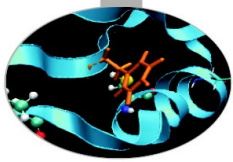
Also, shows you the job id required for other llq options

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

.....

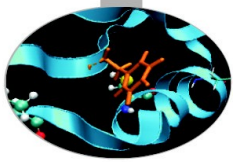
.....

llcancel

llcancel <job_id>

Removes the job from the scheduler, killing it

JOB CLASSES



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)

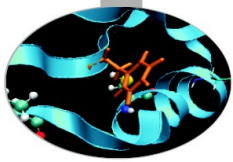
parallel: bg_size>64 (valid values: 128,256,512,1024,2048)

There are some classes that you can specify:

special: bg_size>64 (up to 512), @ class = special
this class allows you to run in 16 I/O nodes racks with bigger jobs

keyproject: for bigger jobs (> 2048 nodes). You have to be an authorized user (write to **superc@cineca.it**)

MODULE «SUPERC»



>module load superc

jobtyp (provides useful information about job in the LL queues - user, tasks, times, ...)

- For using

> jobtyp <job_id>

sstat/sstat2 (provides useful information about the system status - jobs in the LL queues, allocated nodes, Midplane status,...

- For using

> sstat

> sstat2

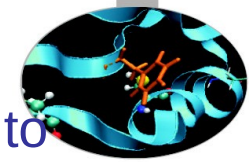
bgtop (draws a full-terminal display of nodeboards and jobs)

>bgtop

loadHPC (calculates aggregate statistics of LL jobs)

>loadHPC

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.

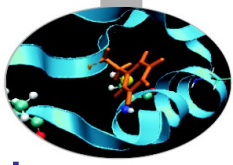
You can check the state of your account with the command "*saldo –b*", which tells you how many CPU hours you have already consumed for each account you're assigned at

(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
```

SMT

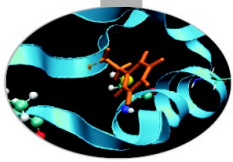


It is possible to improve the efficiency of every single CPU by activating **S**imultaneous **M**ulti **T**hreading (SMT)

Each CPU is divided into threads that act as separated tasks, sharing the CPU resources to work simultaneously (with some loss because of latency)

On FERMI, you can activate 2 or 4 simultaneous threads per CPU, meaning for example that you can launch a job with 2048 or 4096 tasks asking only for 1024 cores!

This is achieved by asking for ranks-per-node = 32 (2*16) or ranks-per-node = 64 (4*16)



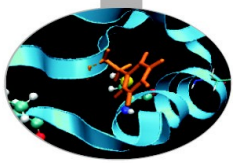
SUB-BLOCKING

Remember that you are consuming the ALLOCATED resources and not necessarily the REQUESTED resources!!

(allocated compute nodes)*(16cores)*(exec. time)

There is, however, a technique that allows to launch multiple executables on a single 64 nodes allocation, partitioning it in sub-groups of nodes called **sub-blocks**

With sub-blocking, you can get advantage of the full number of resources you have to allocate, even with smaller or not very scalable applications. Nothing is wasted!



HOW TO USE SUB-BLOCKING

Some environment variables have to be set for sub-blocking usage:

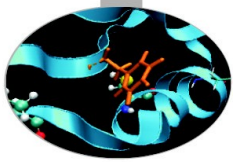
> module load subblock

You can find a complete jobscript in our LL User Guide (link at final slide)

Jobscript USER SECTION:

```
export N_SUBBLOCK=4              ### No. of sub-block you're asking (2,4,8,16,32,64)
export RANK_PER_NODE=16          ### No. of MPI tasks in each node.
### module load <your applications>
export WDR=$PWD
export EXE_1=$WDR/executable_1.exe
export EXE_2=$WDR/executable_2.exe
…
export EXECUTABLES="$EXE_1 $EXE_2 $EXE_3 $EXE_4"
```

USEFUL DOCUMENTATION



FERMI USER GUIDE:

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

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

Job command file keyword descriptions IBM

http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.loadl.v5r1.load100.doc/am2ug_sbmbgjbs.htm