11th Advanced School on PARALLEL COMPUTING

Introduction to EURORA

Parallel & production environment

Mirko Cestari - m.cestari@cineca.it
Alessandro Marani - a.marani@cineca.it
SuperComputing Applications and Innovation Department

CINECA

February 09, 2015

GOALS

You will learn:

- basic concepts of the system architecture that directly affects your work during the school
- how to explore and interact with the software installed on the system
- how to compile a parallel code and/or a code that involves the usage of accelerators (MICs & GPUs)
- how to launch a simulation exploiting the computing resources provided by the EURORA system

OUTLINE

- **A first step:**
  - **System overview**
  - **Login**
  - **Work environment**
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- Programming environment
  - Module system
  - Serial and parallel compilation
  - Interactive session
- Dealing with accelerators
  - Compiling for GPUs and MICs
  - Accelerator job submission
- For further info…
  - Useful links and documentation

EURORA CHARACTERISTICS

Model: Eurora prototype

Architecture: Linux Infiniband Cluster

Processors Type:

- Intel Xeon (Eight-Core SandyBridge) E5-2658 2.10 GHz (Compute)
- Intel Xeon (Eight-Core SandyBridge) E5-2687W 3.10 GHz (Compute)
- Intel Xeon (Esa-Core Westmere) E5645 2.4 GHz (Login)

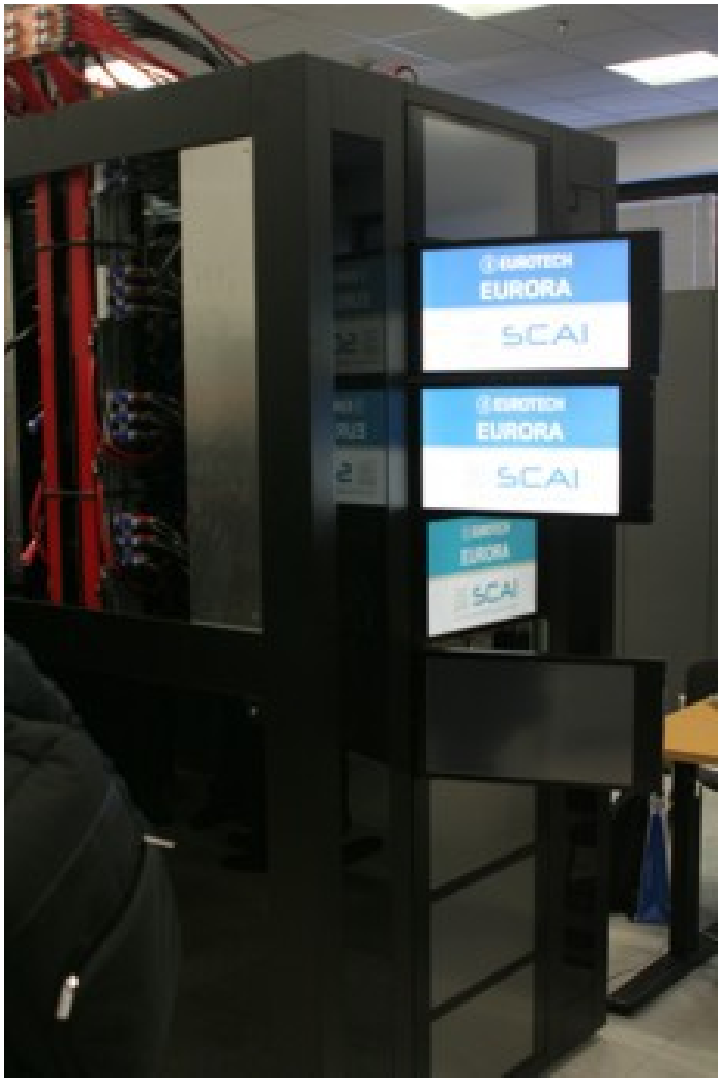
Number of nodes: 64 Compute + 1 Login

Number of cores: 1024 (compute) + 12 (login)

accelerators: 64 nVIDIA Tesla K20 (Kepler) + 64 Intel Xeon Phi (MIC)

RAM: 1.1 TB (16 GB/Compute node + 32GB/Fat node)

OS: RedHat CentOS release 6.3, 64 bit



EURORA CHARACTERISTICS

- **Compute Nodes:** 64 16-core compute cards (nodes).
  - 32 nodes contain 2 Intel(R) Xeon(R) SandyBridge 8-core E5-2658 processors, with a clock rate of about 2 GHz,
  - 32 nodes contain 2 Intel(R) Xeon(R) SandyBridge 8-core E5-2687W processors, with a clock rate of about 3 GHz.
  - 58 compute nodes have 16GB of memory, but the allocatable memory on the node is 14 GB. The remaining 6 nodes (with processors at 3 GHz clock rate) have 32 GB RAM.
  - The Eurora cores are capable of 8 floating point operations per cycle. Half of the compute cards (the ones with a 3GHz clock rate) have two nVIDIAK20 (Kepler) GPU cards installed. The other half (the 2GHz ones) have two Intel Xeon Phi accelerators installed.
- **Login node:** 2 Intel(R) Xeon(R) 6-core Westmere E5645 processors at 2.4 GHz.
- **Network**: all the nodes are interconnected through a custom Infiniband network, allowing for a low latency/high bandwidth interconnection.

EURORA IN GREEN500

THE GREEN 500™

The Green500 is a ranking that classifies the Top500 supercomputers in terms of "energy efficiency"

(best ratio performance/power consumption)

Listed below are the June 2013 The Green500's energy-efficient supercomputers ranked from 1 to 10.

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	3,208.83	CINECA	Eurora - Eurotech Aurora HPC 10-20, Xeon E5-2687W 8C 3.100GHz, Infiniband QDR, NVIDIA K20	30.70
2	3,179.88	Selex ES Chieti	Aurora Tigon - Eurotech Aurora HPC 10-20, Xeon E5-2687W 8C 3.100GHz, Infiniband QDR, NVIDIA K20	31.02
3	2,449.57	National Institute for Computational Sciences/University of Tennessee	Beacon - Appro GreenBlade GB824M, Xeon E5-2670 8C 2.600GHz, Infiniband FDR, Intel Xeon Phi 5110P	45.11

In June 2013 ranking, EURORA has been proclaimed the greenest supercomputer in the world!!

In the last ranking (Nov 2013), unfortunately EURORA didn't make the Top500.

It would have been ranked #4 in the Green500

A LOOK AT THE FUTURE: GALILEO

Model: IBM NeXtScale

Architecture: Linux Infiniband Cluster

Processors Type: 8-cores Intel Haswell 2.40 GHz (2 per node)

Number of nodes: 516 Compute

Number of cores: 8256

Accelerators: 2 Intel Phi 7120p per node on 384 nodes (768 in total)

RAM: 128 GB/node, 8 GB/core

OS: RedHat CentOS release 7.0, 64 bit

The production and programming environment of GALILEO are really similar to the ones we are using on EURORA

How to log in

- Establish a ssh connection

ssh <username>@login.eurora.cineca.it

- Remarks:
  - **ssh** available on all linux distros
  - **Putty** (free) or **Tectia** ssh on Windows
  - *secure shell plugin* for Google Chrome!
  - login nodes are swapped to keep the load balanced
  - important messages can be found in the *message of the day*

- Check the **user guide**!

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

WORK ENVIRONMENT

$HOME:

Permanent, backed-up, and local to EURORA.

5 Gb of quota. For source code or important input files.

$CINECA_SCRATCH:

Large, parallel filesystem (GPFS).

No quota. Run your simulations and calculations here.

use the command cindata to get info on your disk occupation

http://www.hpc.cineca.it/content/data-storage-and-filesystems-0

OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- **Production environment**
  - **Our first job!!**
  - **Creating a job script**
  - **Accounting and queue system**
  - **PBS commands**
- Programming environment
  - Module system
  - Serial and parallel compilation
- Dealing with accelerators
  - Compiling for GPUs and MICs
  - Interactive session
  - Accelerator job submission
- For further info…
  - Useful links and documentation

LAUNCHING JOBS

As in every HPC cluster, EURORA allows you to run your simulations by submitting **“jobs”** to the compute nodes

Your job is then taken in consideration by a **scheduler**, that adds it to a queuing line and allows its execution when the resources required are available

The operative scheduler in EURORA is **PBS**

PBS JOB SCRIPT SCHEME

The scheme of a PBS job script is as follows:

#!/bin/bash

#PBS keywords

variables environment

PBS JOB SCRIPT EXAMPLE

```
#!/bin/bash
#PBS -N myname
#PBS -o job.out
#PBS -e job.err
#PBS -m abe
#PBS -M user@email.com
#PBS -l walltime=00:30:00
#PBS -l select=1:ncpus=16:mpiprocs=8:mem=10GB
#PBS -q debug
#PBS -A <my_account>

echo “I’m working on EURORA!”
```

PBS KEYWORD ANALYSIS - 1

#PBS -N myname

Defines the name of your job

#PBS -o job.out

Specifies the file where the standard output is directed (default=jobname.o<jobID>)

#PBS -o job.err

Specifies the file where the standard error is directed (default=jobname.e<jobID>)

#PBS -m abe (optional)

Specifies e-mail notification. An e-mail will be sent to you when something happens to your job, according to the keywords you specified (a=aborted, b=begin, e=end, n=no email)

#PBS -M user@email.com (optional)

Specifies the e-mail address for the keyword above

PBS KEYWORD ANALYSIS - 2

#PBS -l walltime=00:30:00

Specifies the maximum duration of the job. The maximum time allowed depends on the queue used

(more about this later)

#PBS -l select=1:ncpus=16:mpiprocs=8:mem=10GB

Specifies the resources needed for the simulation.

select – number of compute nodes (“chunks”)

ncpus – number of cpus per node (max. 16)

mpiprocs – number of MPI tasks per node (max=ncpus)

mem – memory allocated for each node (default=850MB, max.=14 GB)

You can require up to 32GB but have to wait more because you will be directed on the special high memory nodes

NEVER ask for 15GB, or the job won’t run properly!

QUEUING SYSTEM

#PBS -q debug

Specifies the queue requested for the job. The job will be put by PBS on the waiting list depending on the queue specified. For each queue, there is a set limit of walltime and resources that can be asked.

Job type	Max nodes	Time slot	Max wall time
debug	2	Always	0:30:00
parallel	32	Always	4:00:00
np_longpar	9	Non primetime*	8:00:00
p_devel	2	Primetime*	1:00:00

The queue parameter is actually optional and doesn't need to be specified, as it depends from the resources asked. The only exception is p_devel.

* Primetime = 10am – 6pm weekdays. Non primetime = (6 pm - 10 am weekdays, Friday 6 pm - Monday 10 am)

ACCOUNTING SYSTEM

#PBS -A <my_account>

Specifies the account to use the CPU hours from.
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 status 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
```

ACCOUNT FOR THE SCHOOL

The account provided for this school is "**train_scA2015**"

(you have to specify it on your job scripts).

It will expire two weeks after the end of the school and is shared between all the students; there are plenty of hours for everybody, but don't waste them!

#PBS -A train_scA2015

RESERVATION FOR THE SCHOOL

For the days of the school, 4 nodes equipped with 2 Xeon Phi each are reserved for the students

In order to use it, you have to specify a special queue parameter:

#PBS -q R1603381

PBS COMMANDS

After the job script is ready, all there is left to do is to submit it:

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 PBS commands

PBS COMMANDS

qstat

qstat -f <job_id>

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

qdel

qdel <job_id>

Removes the job from the scheduled jobs by killing it

EXERCISE 01

1) Write a job script with "walltime" of 3 minutes that asks for 1 node and 1 core. Copy-paste the following in the execution section

hostname

echo 'Hello World'

sleep 4

Now add the automatic sending of the email in case of ending and abort of the job.

2) Launch the job with qsub

3) Check its state with qstat

4) Check its state again with "qstat -f jobid" after having increased the sleep to 60, namely:

hostname

echo 'Hello World'

sleep 60

5) Add a memory request to the "select" line in the job script (rember that each processor has a quota of 850 MB of memory). Please check the new requirements with "qstat -f jobid"

OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- **Programming environment**
  - **Module system**
  - **Serial and parallel compilation**
  - **Interactive session**
- Dealing with accelerators
  - Compiling for GPUs and MICs
  - Accelerator job submission
- For further info…
  - Useful links and documentation

AN EXAMPLE OF A PARALLEL JOB

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=2:ncpus=16:mpiprocs=4
#PBS -o job.out
#PBS -e job.err
#PBS -q parallel
#PBS -A <my_account>

cd $PBS_O_WORKDIR    # points to the folder you are actually working into
module load autoload openmpi

mpirun –np 8 ./myprogram
```

MODULE SYSTEM

- All the optional software on the system is made available through the **"module" system**
  - provides a way to rationalize software and its environment variables

- Modules are divided in 2 *profiles*
  - **profile/base** (default - stable and tested modules)
  - **profile/advanced** (software not yet tested or not well optimized)

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

MODULE SYSTEM

- CINECA's work environment is organized in modules, a set of installed libraries, tools and applications available for all users.
- "loading" a module means that a series of (useful) shell environment variables will be set
- E.g. after a module is loaded, an environment variable of the form "<MODULENAME>_HOME" is set

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

MODULE COMMANDS

COMMAND	DESCRIPTION
module av	list all the available modules
module load <module_name(s)>	load module <module_name>
module list	list currently loaded modules
module purge	unload all the loaded modules
module unload <module_name>	unload module <module_name>
module help <module_name>	print out the help (hints)
module show <module_name>	print the env. variables set when loading the module

MODULE PREREQS AND CONFLICTS

Some modules need to be loaded after other modules they depend from (e.g.: parallel compiler depends from basic compiler). 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
```

You may also get a “conflict error” if you load a module not suited for working together with other modules you already loaded (e.g. different compilers). Unload the previous module with “module unload”

COMPILING ON EURORA

- On EURORA 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.

module load intel *# loads default intel compilers suite*

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

	GNU	INTEL	PGI
Fortran	gfortran	ifort	pgf77
C	gcc	icc	pgcc
C++	g++	icpc	pgcc

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

PARALLEL COMPILING ON EURORA

- MPI libraries available: **OpenMPI/IntelMPI**
  - The library and special wrappers to compile and link the personal programs are contained in several "openmpi" modules, one for each supported suite of compilers
- Load a version of OpenMPI:

```
module av openmpi
        openmpi/1.6.4--pgi--12.10
        openmpi/1.6.5--gnu--4.6.3
        openmpi/1.6.5--intel--cs-xe-2013--binary
        openmpi/1.6.5--pgi--12.10
        openmpi/1.6.5--pgi--14.1
module load autoload openmpi/1.6.4--gnu--4.6.3
```

- Load a version of IntelMPI:

```
module av intelmpi
          intelmpi/4.1.0--binary
          intelmpi/4.1.1--binary
module load autoload intelmpi/4.1.1--binary
```

PARALLEL COMPILING ON EURORA

	OPENMPI/INTELMPI
Fortran90	mpif90
C	mpicc
C++	mpiCC

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

OpenMP is provided with following compiler flags:

gnu: -fopenmp

intel : -openmp

pgi: -mp

JOB SCRIPT FOR PARALLEL EXECUTION

Let's take a step back…

#PBS -l select=2:ncpus=16:mpiprocs=4

This example line means "allocate 2 nodes with 16 CPUs each, and 4 of them should be considered as MPI tasks"

So a total of 32 CPUs will be available. 8 of them will be MPI tasks, the others will be OpenMP threads (4 threads for each task).

In order to run a pure MPI job, ncpus must be equal to mpiprocs.

EXECUTION LINE IN JOB SCRIPT

mpirun –np 8 ./myprogram

Your parallel executable is launched on the compute nodes via the command

“mpirun”.

With the “–np” flag you can set the number of MPI tasks used for the execution.

The default is the maximum number allowed by the resources requested.

WARNING:

In order to use mpirun, **openmpi-intelmpi has to be loaded.**

module load autoload openmpi

DEVELOPING IN COMPUTE NODES: INTERACTIVE SESSION

It may be easier to compile and develop directly in the compute nodes, without recurring to a batch job.

For this purpose, you can launch an interactive job to enter inside a compute node by using PBS.

The node will be reserved to you as it was requested by a regular batch job

Basic interactive submission line:

```
qsub -I -l select=1 -A <account_name> (-q <queue_name>)
```

Other PBS keyword can be added to the line as well (walltime, resources,…)

EXERCISE 02

1) Compile "test.c" with the compiler (mpicc) in the module intelmpi/4.1.1--binary

2) Check with:

$ ldd <executable>

the list of required dynamic libraries.

3) Write "job.sh" (you can copy it from exercise 1), modifying the "select" line with the following requests:

#PBS -l select=2:ncpus=16:mpiprocs=16:mem=12gb

#PBS -l select=2:ncpus=16:mpiprocs=1:mem=12gb

Run first 32 processes and then 2 processes for each select.

EXERCISE 03

1) Launch an interactive job. You just need to write the same PBS directives, without "#PBS" and on the same line, as arguments of "qsub -I"

$ qsub -I ... <arguments>

2) Check whether you are on a different node

3) Check that there's an interactive job running

OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- Programming environment
  - Module system
  - Serial and parallel compilation
  - Interactive session
- **Dealing with accelerators**
  - **Compiling for GPUs and MICs**
  - **Accelerator job submission**
- For further info…
  - Useful links and documentation

COMPILING FOR GPUS: CUDA

CUDA is the programming language used for developing HPC applications that involve the usage of GPUs.

For compiling a GPU application, the module “cuda” is available on EURORA:

```
module load cuda/5.0.35
```

The module provides the compilator “nvcc” and optimized GPU-enabled scientific libraries for linear algebra, FFT, random number generators, and basic algorithms:

CUBLAS: GPU-accelerated BLAS library

CUFFT: GPU-accelerated FFT library

CUSPARSE: GPU-accelerated Sparse Matrix library

CURAND: GPU-accelerated RNG library

CUDA NPP: nVidia Performance Primitives

THRUST: a CUDA library of parallel algorithms with an interface resembling the C++ Standard Template Library (STL)

COMPILING WITH CUDA

In order to compile an application with the CUDA module, you need to move your compilation to a

compute node. Thus, you need to submit a batch job:

```
#!/bin/bash
#PBS -l walltime=0:30:00
#PBS -l select=1:ncpus=1
#PBS -o job.out
#PBS -A <myaccount>

cd $PBS_O_WORKDIR
module load gnu   #the modules relative to the non-GPU compilation have to be loaded
                  #before cuda
module load cuda

nvcc –arch=sm_30 –I$CUDA_INC –L$CUDA_LIB –lcublas –o myprog myprog.c
```

CUDA libraries have to be linked in compilation phase

-arch=sm30 is for exploiting at best the characteristics of EURORA's architecture

COMPILING WITH CUDA IN INTERACTIVE

```
qsub -I -l walltime=0:10:00 -l select=1:ncpus=1 -A <myaccount>
```

After a short waiting time, you will be prompted inside a computing node. Now you can compile with CUDA as you would normally do:

```
module load gnu

module load openmpi/1.6.4--gnu--4.6.3

module load cuda

make
```

You can exit the interactive session with “exit” or ^D

COMPILING FOR MICS

The MPSS environment (Intel® *Manycore Platform Software Stack*) for MIC compiling is available also on the Eurora Front-end nodes. Therefore, you do not need to be logged **inside** a compute node to compile a code suited for the MICs. However, you still have to set the proper environment for MIC compilation:

```
module load intel (i.e. compiler suite)
module load mkl (if necessary – i.e. math libraries)
source $INTEL_HOME/bin/compilervars.sh intel64 (to set up
the environment variables)
```

The compilation now differs depending on which you want to compile in **offload** or **native** mode

OFFLOAD AND NATIVE MODE

Offload mode means that the code is run mainly on CPUs but parallel segments are moved to MICs

Offload mode is resolved mainly with pragmas on the source code and thus can be compiled as usual:

```
icpc -openmp hello_offload.cpp -o exe-offload.x
```

Native mode means that the code is run interely inside the MIC cards

For a native compilation, you have to remember to cross-compile by using the –mmic flag:

```
icpc -openmp hello_native.cpp –mmic -o exe-native.x
```

MIC AND MPI COMPILATION

In order to compile parallel programs for MIC cards, some additional setup is required:

```
module load intel (i.e. compiler suite)

module load intelmpi (i.e. mpi library)

module load mkl (if necessary – i.e. math libraries)

source $INTEL_HOME/bin/compilervars.sh intel64 (to set up the

environment variables)
```

Now you can compile as usual. Remember to cross-compile if native!

```
export I_MPI_MIC=enable (to enable mpi on MIC)
```

For Fortran applications, a special “mpifc” compiler has to be used:

```
mpifc -O3 -mmic mpi_code.f
```

SUBMITTING JOBS WITH GPUS

Submitting jobs involving GPUs is the same as submitting regular CPUs jobs. The only difference is that the GPU usage has to be specified:

```
#!/bin/bash

#PBS -l walltime=30:00

#PBS -l select=1:ncpus=1:ngpus=1

#PBS -o job.out

#PBS -e job.err

#PBS -A <my_account>

cd $PBS_O_WORKDIR

./myCUDAprogram
```

GPUs have to be required in the PBS resources keyword. The parameter *ngpus* specifies the number of GPUs per node requested (max. 2)

SUBMITTING JOBS WITH MICS OFFLOAD MODE

Submitting jobs involving MICs (in offload mode) is similar as submitting regular CPUs

or GPUs jobs:

```
#!/bin/bash
#PBS -o job.out
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:nmics=1
#PBS -A <my_account>

module load intel
cd $CINECA_SCRATCH
source $INTEL_HOME/bin/compilervars.sh intel64
./exe-offload.x
```

Like with ngpus, the parameter *nmics* specifies the number of MIC cards allocated for each node and can go up to 2. Notice also the necessity to load the compilervars.sh script.

RUNNING MIC EXECUTABLES NATIVE MODE

MIC-native codes need to be executed inside the MIC card itself. In order to log into a MIC card you have to:

- login to a MIC node with a PBS interactive session requesting at least 1 mic (nmics=1);

- use the "qstat -f <job_id>" command in order to get the name of the specific MIC card assigned to you;

- connect through ssh into the MIC card (in the example node018-mic1)

```
qsub -A <account_name> -I -l select=1:ncpus=1:nmics=1
qsub: waiting for job 31085.node129 to start
qsub: job 31085.node129 ready
...
qstat -f 31085.node129
...
exec_vnode = (node018:mem=1048576kb:ncpus=1+node018-mic1:nmics=1)
...
ssh node018-mic1
$
```

RUNNING MIC EXECUTABLES NATIVE MODE

At this point you will be prompted in the home space of the MIC card you've logged into. Here, the usual environment variables are **not** set, therefore the module command won't work and your scratch space (which is mounted on the MIC card) has to be indicated with the full path instead of $CINECA_SCRATCH.

For executing your native-MIC program, you need to set the LD_LIBRARY_PATH environment variable manually, by adding the path of the intel libraries specific for MIC execution. You may also need to add also path for mkl and/or tbb (Intel® Thread Building Blocks) MIC libraries.

When everything is ready, you can launch your code as usual:

```
cd /gpfs/scratch/userexternal/<myuser>
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2013/binary/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2013/binary/mkl/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2013/binary/tbb/lib/mic:${LD_LIBRARY_PATH}
./exe.native.x
```

MIC NATIVE + MPI

To run MIC native applications that involve MPI, you can stay in the MIC node. Export the I_MPI_MIC environment variable and use the specific command *mpirun.mic*. In the execution line you have to specify the MIC card involved with the flag -host :

```
export I_MPI_MIC=enable
mpirun.mic  -host node018-mic1 -np 30  ./a.out
```

If you want to use two MIC cards you can set the number of tasks per card via the -perhost flag:

```
mpirun.mic  -host node018-mic0,node018-mic1 -perhost 15 -np 30  ./a.out
```

For MPI+OpenMP applications, specify the number of threads involved with the flag *–genv* :

```
mpirun.mic  -host node018-mic0,node018-mic1 -perhost 1 -np 2 -genv OMP_NUM_THREADS
120 ./a.out
```

EXERCISE 04

1) Compile the provided cuda program, after having loaded the required modules

2) Write a job script for a serial execution (1 chunk, 1 cpu) that asks also for a gpu device

3) Run the job script

EXERCISE 05a

1) log into a MIC node with a PBS interactive job requesting at least 1 mic (nmics=1);

2) use the "qstat -f <job_id>" command in order to get the name of the specific MIC card assigned to you;

3) connect through ssh into the MIC card (i.e "ssh node018-mic1")

4) now set LD_LIBRARY_PATH as following:

export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2013/binary/lib/mic:${LD_LIBRARY_PATH}

and "cd" to the directory containing the executable

5) launch the execution

./exe-native.x

Exercise 05b

1) compile the provided cpp code for offload mic usage

2) Write a job script that asks for a mic (nmics=1)

3) launch the job with 10 threads (export the OMP_NUM_THREADS variable)

OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- Programming environment
  - Module system
  - Serial and parallel compilation
  - Interactive session
- Dealing with accelerators
  - Compiling for GPUs and MICs
  - Accelerator job submission
- **For further info…**
  - **Useful links and documentation**

Useful links and documentation

- **Reference guide**:
  http://www.hpc.cineca.it/content/eurora-user-guide
  http://www.hpc.cineca.it/content/eurora-batch-scheduler-pbs
  http://www.hpc.cineca.it/content/gpgpu-general-purpose-graphics-processing-unit
  http://www.hpc.cineca.it/content/quick-guide-intel-mic-usage
  http://www.hpc.cineca.it/content/galileo

- **GPU computing** *http://www.nvidia.com/object/GPU_Computing.html*
  MIC programming http://software.intel.com/en-us/mic-developer

- **Stay tuned with the HPC news: http://www.hpc.cineca.it/content/stay-tuned**

- **HPC CINECA User Support:** mail to superc@cineca.it
- **HPC Courses:** corsi@cineca.it