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

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 2015), unfortunately EURORA didn't make the Top500.

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

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 parallel
#PBS -A <my_account>
#PBS -W group_list=<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 -e 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)

QUEUING SYSTEM

#PBS -q <queuename>

Specifies the queue requested for the job. Since EURORA is out of production, queues are actually disabled.

You have a special queue reserved for the course. In order to use it, you have to specify the following:

#PBS -q R1661336

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 COURSE

The account provided for this school is "**train_cmpB2015**"

(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_cmpB2015

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

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).

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,…)