Cineca
TRAINING
High Performance
Computing 2017

Introduction to MARCONI

Parallel & production environment

Mirko Cestari – m.cestari@cineca.it
Alessandro Marani – a.marani@cineca.it
Domenico Guida – d.guida@cineca.it
SuperComputing Applications and Innovation Department

CINECA

May 16, 2017

GOALS

You will learn:

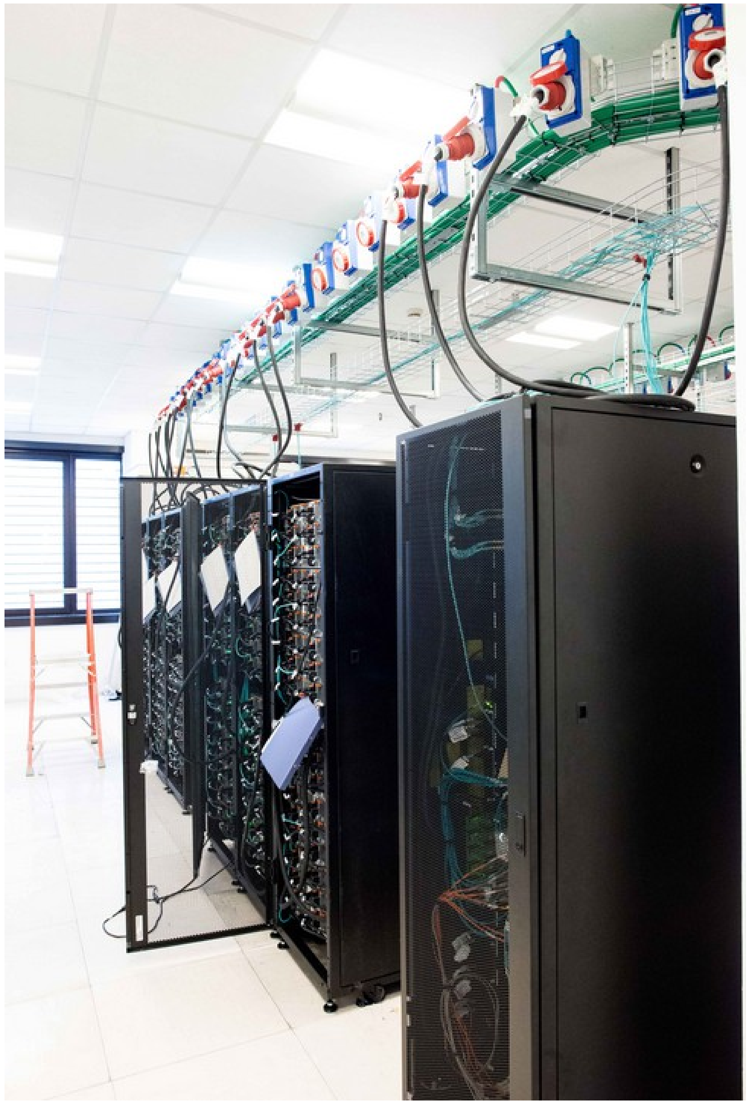
- basic concepts of the system architecture that directly affects your work during the school
- how to explore and interact with the software installed on the system
- how to launch a simulation exploiting the computing resources provided by the MARCONI system

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
- KNL Environment
- For further info...
  - Useful links and documentation

MARCONI CHARACTERISTICS: A1

Model: Lenovo NeXtScale
Architecture: Intel Omnipath Cluster
Processors Type: 18-cores Intel Xeon E5-2697 v4 (Broadwell) 2.30 Ghz (2 per node)
Number of nodes: 1512 Compute
Number of cores: 54432
RAM: 128 GB/node, 3.5 GB/core
Internal Network: Intel Omnipath Architecture 2:1
Peak Performance: 2 Pflop/s
OS: RedHat CentOS release 7.2, 64 bit



MARCONI CHARACTERISTICS: A2

Model: Lenovo Adam Pass

Architecture: Intel Omnipath Cluster

Processors Type: 68-cores Intel Xeon Phi 7250 CPU (Knights Landing) 1.40 Ghz

Number of nodes: 3600 Compute

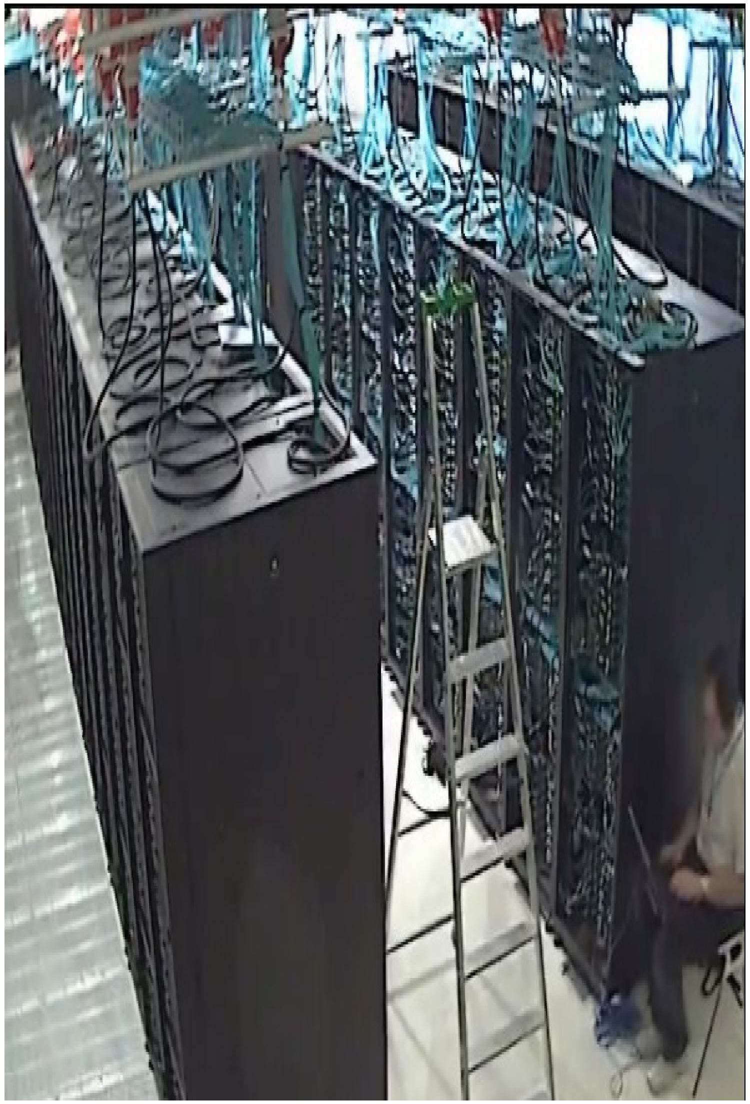
Number of cores: 244800

RAM: 108 GB/node, 96 of DDR4 and 16 of MCDRAM

Internal Network: Intel Omnipath Architecture 2:1

Peak Performance: 11 Pflop/s

OS: RedHat CentOS release 7.2, 64 bit



MARCONI CHARACTERISTICS

- **Compute Nodes:** 1512 36-core compute nodes for A1, 3600 68-core compute nodes for A2.
  - The nodes have 128GB of memory, but the allocatable memory on the node is 118 GB for A1, and 96 GB for A2
  - Not all nodes are available for all the users. A partition of the cluster is reserved for EUROFusion community, and the rest is available for academical users

- **Login nodes:** 8 Login (3 available for regular users) & 12 service nodes for cluster management, each one contains 2 x Intel Xeon Processor E5-2697 v4 with a clock of 2.30GHz and 128 GB of memory. The login nodes are shared between A1 and A2, while the service nodes are splitten among the partitions (6 for A1 and 6 for A2).
- **Network**: all the nodes are interconnected through a custom Intel Omnipath network that can go up to 100Gb/s, making MARCONI the largest Omnipath cluster in the World.

A LOOK AT THE (NEAR) FUTURE: PHASE A3

Model: Lenovo Stark

Architecture: Intel Omnipath Cluster

Processors Type: 40-cores Intel Skylake, 2.30 Ghz

Number of nodes: 1512 Compute

Number of cores: 60480

RAM: 192 GB/node

Internal Network: Intel Omnipath Architecture 2:1

Peak Performance: 4,5 Pflop/s (presumed)

OS: RedHat CentOS release 7.2, 64 bit

HOW TO LOG IN

- Establish a ssh connection

ssh <username>@login.marconi.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**!

https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.1%3A+MARCONI+UserGuide

- When we login, our default environment is set to work with A1 partition. This is where we will handle the first part of our hands-on.

WORK ENVIRONMENT

$HOME:

Permanent, backed-up, and local to MARCONI.

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

$CINECA_SCRATCH:

Large, parallel filesystem (GPFS).

No quota. Run your simulations and calculations here.

A cleaning procedure automatically deletes every file untouched since 50 days

$WORK:

Similar to $CINECA_SCRATCH, but the content is shared among all the users of the same account.

1 Tb of quota. Stick to $CINECA_SCRATCH for the school exercises!

use the command cindata to get info on your disk occupation

https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.4%3A+Data+storage+and+FileSystem

OUTLINE

LAUNCHING JOBS

As in every HPC cluster, MARCONI 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 MARCONI is **PBS**

PBS JOB SCRIPT SCHEME

The scheme of a PBS job script is as follows:

#!/bin/bash

#PBS keywords

variables environment

execution line

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=36:mpiprocs=18:mem=10GB
#PBS -A <my_account>

echo “I’m working on MARCONI!”
```

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=36:mpiprocs=18:mem=10GB

Specifies the resources needed for the simulation.

select – number of compute nodes (“chunks”)

ncpus – number of cpus per node (max. 36)

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

mem – memory allocated for each node (default=3.5 GB)

Cineca TRAINING High Performance Computing 2017

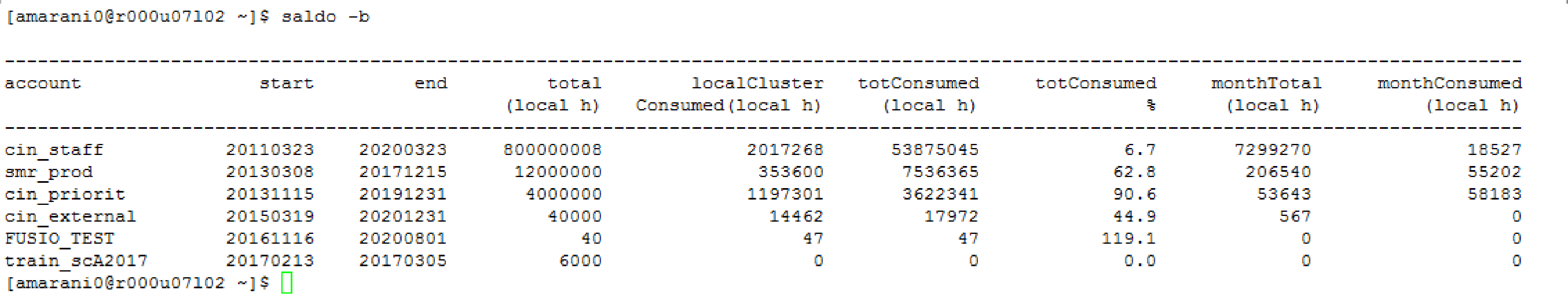
ACCOUNTING SYSTEM

#PBS -A <my_account>

Specifies the account that consumes the CPU hours allocated.

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@r000u07102 ~]$ saldo -b

--------------------------------------------------------------------------------------------------------------------------------------------------
account              start        end        total       localCluster  totConsumed     totConsumed   monthTotal     monthConsumed
                                            (local h)  Consumed(local h)  (local h)            %        (local h)          (local h)
--------------------------------------------------------------------------------------------------------------------------------------------------
cin_staff          20110323    20200323    800000008         2017268     53875045           6.7       7299270              18527
smr_prod           20130308    20171215     12000000          353600      7536365          62.8        206540              55202
cin_priorit        20131115    20191231      4000000         1197301      3622341          90.6         53643              58183
cin_external       20150319    20201231        40000           14462        17972          44.9           567                  0
FUSIO_TEST         20161116    20200801           40              47           47         119.1             0                  0
train_scA2017      20170213    20170305         6000               0            0           0.0             0                  0
[amarani0@r000u07102 ~]$
```

ACCOUNT FOR THE SCHOOL

The account provided for this school (on Broadwell) is “**train_scB2017_0**”

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

It will expire two months 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_scB2017_0

RESERVATION FOR THE SCHOOL

For this afternoon, 4 Broadwell nodes are reserved. The reservation will let you bypass the regular queue and let your jobs run immediately.

The “#PBS -q” keyword regulates the queue selection. If you omit it (regular user behaviour), your job will be processed among the queue suited for the resources you asked (debug, prod, bigprod). To use the reservation, add this to your jobscript:

```
#PBS -q R641493
#PBS -W group_list=train_scB2017_0
```

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 -u

```
qstat -u <username>
```

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.
Hint: add the flag "-w" for an extended vision and the full name of your jobid

PBS COMMANDS

qstat -f

```
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

EXERCISES

GitLab.hpc.cineca.it

Inside your MARCONI $HOME directory, clone the repository:

git clone https://gitlab.hpc.cineca.it/training/summer-school-2017.git

You'll find the exercises in the “first_day” subfolder.

Other teachers may update the repository during the school. use the command:

git pull

To update your work folder.

EXERCISE 01

1) Write a job script with "walltime" of 3 minutes that asks for 1 node, 1 core and 128 GB of memory.

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) The job should start almost immediately, so why isn't it starting? Check with "qstat -f jobid" and use the MARCONI UserGuide (see link in the last slide) to spot the error

5) Fix the script, delete the previous job with qdel and relaunch.

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

...

sleep 60

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**
- KNL Environment
- 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=36:mpiprocs=18
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR     # points to the folder you are actually working into
module load autoload intelmpi
mpirun –np 32 ./myprogram
```

MODULE SYSTEM

- All the optional software on the system is made available through the **"module" system**. It provides a way to rationalize software and its environment variables.
- Modules are divided in several *profiles*:

  •**profile/base** default - stable and tested compilers, libraries, tools

  •**profile/advanced** libraries and tools compiled with different setups that the default

  •**profile/chem** (phys, bioinf, astro,...) “domain” profiles with the application softwares specific for each field of research

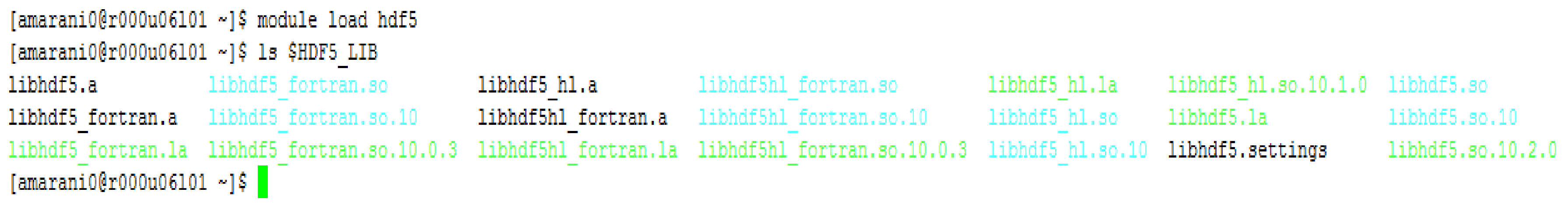
  •**profile/archive** old or outdated versions of our module (we don't throw away anything!)
- Each profile is divided in 4 categories

  compilers (GNU, intel, openmpi) **tools** (e.g. Scalasca, GNU make, VNC, ...)

  libraries (e.g. LAPACK, BLAS, FFTW, …) **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@r000u06101 ~]$ module load hdf5
[amarani0@r000u06101 ~]$ ls $HDF5_LIB
libhdf5.a          libhdf5_fortran.so          libhdf5_hl.a          libhdf5hl_fortran.so          libhdf5_hl.la     libhdf5_hl.so.10.1.0  libhdf5.so
libhdf5_fortran.a  libhdf5_fortran.so.10       libhdf5hl_fortran.a   libhdf5hl_fortran.so.10       libhdf5_hl.so     libhdf5.la            libhdf5.so.10
libhdf5_fortran.la libhdf5_fortran.so.10.0.3   libhdf5hl_fortran.la  libhdf5hl_fortran.so.10.0.3   libhdf5_hl.so.10  libhdf5.settings      libhdf5.so.10.2.0
[amarani0@r000u06101 ~]$
```

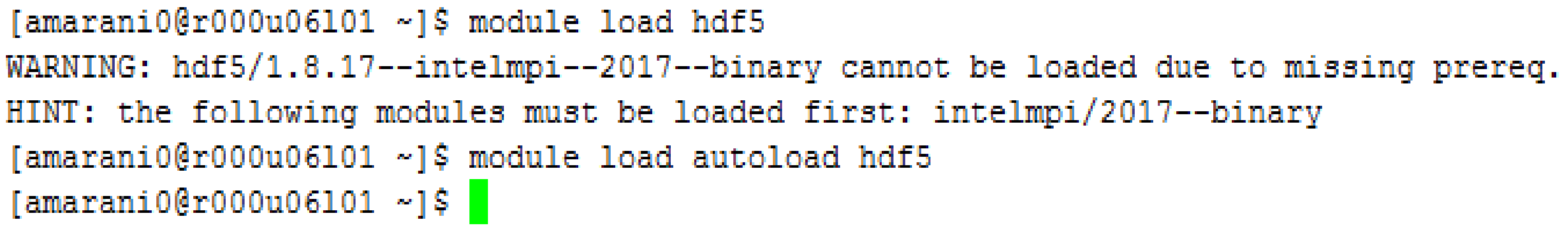
- For certain modules, a specific profile must be loaded before (“module load profile/...”). Use the “modmap” command to understand which module is in which profile (try “modmap -h”)

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”



```
[amarani0@r000u06101 ~]$ module load hdf5
WARNING: hdf5/1.8.17--intelmpi--2017--binary cannot be loaded due to missing prereq.
HINT: the following modules must be loaded first: intelmpi/2017--binary
[amarani0@r000u06101 ~]$ module load autoload hdf5
[amarani0@r000u06101 ~]$
```

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 MARCONI

- On MARCONI 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/pe-xe-2017--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 MARCONI

MPI libraries available: **OpenMPI/IntelMPI**

The library and special wrappers to compile and link the personal programs are contained in several modules, one for each supported suite of compilers

Load a version of **OpenMPI**:

```
module av openmpi
openmpi/1-10.3--gnu--6.1.0 (profile/base)
openmpi/1-10.3--intel--pe-xe-2017--binary
(profile/advanced)
module load autoload openmpi/1-10.3--gnu--6.1.0
```

Load a version of **IntelMPI**:

```
module av intelmpi
intelmpi/5.1--binary     (intel 2016)
intelmpi/2017--binary    (intel 2017)
module load autoload intelmpi/2017--binary
```

PARALLEL COMPILING ON MARCONI

	OPENMPI/INTELMPI
Fortran90	mpif90/mpiifort
C	mpicc/mpiicc
C++	mpiCC/mpiicpc

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

OpenMP is provided with the following compiler flags:

gnu: -fopenmp

intel : -qopenmp

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** inside the job script:

```
module load autoload intelmpi
```

Be sure to load the same version of the compiler that you used to compile your code!!

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,…)

Keep in mind that you are using computing nodes, and by consequence you are consuming

computing hours!

To exit from an interactive session, just type “exit”

EXERCISE 02

1) Compile "test.c" with the compiler (mpiicc) in the module intelmpi/2017--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=3gb
#PBS -l select=2:ncpus=16:mpiprocs=1:mem=3gb
Run first 32 processes and then 2 processes for each select.

Optional: if using Intelmpi (recommended), inside the job export the environment variable:
I_MPI_DEBUG=5
And see what information it prompts you on the standard error.

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

KNL PARTITION

The most important thing to remember is that the two partitions of MARCONI can be imagined as two separate HPC clusters, that are sharing the same front-end environment

Thus, trying to work with KNL has different rules from the regular Broadwell environment, even in terms of strict operativity

Since we are basically working with a different PBS, that reorders job in a different list, some environment needs to be set before switching from A1 to A2

MODULE ENV-KNL

The easiest way to do this is a simple load of the module **env-knl** that transports you from the Broadwell environment to the KNL environment

Exercise: the command "qstat -Q" shows all the queues that are defined in an environment (most of them can't be used by you). Try to launch this command, then load env-knl and finally launch it again and see the differences

To return to Broadwell, you can either unload the module or load **env-bdw**, that sets up the Broadwell environment (and automatically unloads env-knl)

COMPILING FOR KNL

Applications compiled for Broadwell can be used in KNL as well. However, one of the main features of Knights Landing is the capability of exploiting AVX-512 instructions for improved performance in terms of vectorization

To your (**Intel**) compilation, add the flag **-xMIC-AVX512** to generate AVX-512 instructions and make your code faster on KNL

WARNING: this flag will cause your executable to **not** run on Broadwell! The flag **-axMIC-AVX512** makes the compilation both portable and optimized.

SUBMIT JOBS FOR KNL - 1

There are a few things that you have to remember when you want to submit a job for KNL. Namely:

1. Remember to have loaded the “env-knl” module in order to submit on KNL queues (you can see it because the prompt begins with (KNL)).
2. While Broadwell has a core-based granularity, and single cores can be asked from PBS, KNL has a node-based granularity, as in, you can't ask for less than one node and its multiples. So, even when you ask for less, at least one full node is always reserved to your job, and in a PBS job only the value of “select” (i.e. The number of nodes) is important.
3. All KNL nodes of MARCONI are set to NUMA quadrant and MCDRAM cache mode (for production reasons): thus, the maximum memory requirement can't go further than 93GB per node (as 16GB is for MCDRAM-cache and what remains is for the OS). This is also not needed to specify because of point 2.

SUBMIT JOBS FOR KNL - 2

4. On KNL nodes the physical limit is 68 cores per node. However, Hyperthreading is activated, meaning that each physical core can be treated as up to 4 virtual cores. So a line like this:

```
#PBS -l select=1:ncpus=68:mpiprocs=272
```

is perfectly allowed.

5. (school only) For working on KNL, a 10 nodes reservation is up and running for the two weeks of the school. Suggestion: stay low with the walltime (5 minutes is usually more than enough), so you will enter faster in execution and let your fellow students use the reserved nodes as well.

```
#PBS -q R168296
```

6. (school only) Your account changes as well. Submit your KNL jobs by specifying "#PBS -A train_scB2017" and (for the reservation) "#PBS -W group_list=train_scB2017".

You can see it with "saldo -b --knl".

EXERCISE 04

1) Compile "vector.f90"* with the compiler (mpiifort) in the module intelmpi/2017--binary. For each rank, it does a sum of the elements of a large array and it prints the sum of some elements and the time spent in the loop. It's a good code for vector optimization.

2) Load the module env-knl to switch to KNL environment, then submit the job asking for 1 node, 64 ncpus and 64 mpiprocs.

3) Compile the code again by adding the -xMIC-AVX512 flag, then submit again. Compare the times obtained with those from the previous run.

Optional: Experiment with the requests, for example by enabling hyperthreading (ncpus=64, mpiprocs=256). Switch back to Broadwell environment and submit the job for the non-vector optimized executable. What simulation has the best time overall?

*credits:
http://www.nersc.gov/users/computational-systems/edison/programming/vectorization/#toc-anchor-2

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
- KNL Environment
- **For further info...**
  - **Useful links and documentation**

Useful links and documentation

- **Reference guide:**
  https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.1%3A+MARCONI+UserGuide
  https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.5.2%3A+Batch+Scheduler+PBS
  https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.4%3A+Data+storage+and+FileSystem

- **About vector optimization:**
  https://wiki.u-gov.it/confluence/display/SCAIUS/How+to+Improve+Code+Vectorization

- **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**