9th Advanced School on PARALLEL COMPUTING

Programming Envinronment on FERMI-BGQ

Mirko Cestari – m.cestari@cineca.it
SuperComputing Applications and Innovation Department

CINECA

February 11 - 15, 2013

Outline

- A first step
  - ssh
  - file transfer
- Introduction to the environment
  - module command
  - module usage
- Programming environment
  - cross - compilation
  - available compilers
  - examples
  - libraries, compiling/linking issues
  - optimization with XL
- Code tuning
  - case study
- For further info...
  - useful links and documentation

FERMI: how to login

- Establish a ssh connection

  ssh <username>@login.fermi.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/hpc-user-guide-2012

FERMI: File transfer

- **sftp / scp** (always available if sshd is running)

```
$ sftp -r <my_dir>  <user>@login.fermi.cineca.it:/path/to/
$ scp -r <my_dir>  <user>@login.fermi.cineca.it:/path/to/
```

- **rsync**: allows incremental transfer

```
$ rsync -avzr --progress <my_dir> <user>@login.fermi.cineca.it:
```

- **gridftp**: allows for stream transfer and much more
  (~10x transfer! - available soon)

```
$ globus-url-copy -vb -r -p 16 -sync -sync-level 2 \
file:/path/to/files/ \
sshftp://user@login.fermi.cineca.it/path/to/
```

"module", my best friend

- all the optional software on the system is made available through the **"module" system**
  - provides a way to rationalize software and its env variables
- modules are divided in 3 *profiles*
  - **profile/base** (stable and tested modules)
  - **profile/front-end** (contains all the base modules plus front-end libs and apps)
  - **profile/advanced** (software not yet tested or not well optimized)
- each profile is divided in 4 categories
  - **compilers** (IBM-xl, GNU)
  - **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - **tools** (e.g. Scalasca, GNU make, VNC, ...)
  - **applications** (software for chemistry, physics, ...)

module env

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

NB: some modules rely on other modules

```
$ module load boost
WARNING: boost/1.51.0--bgq-xl--1.0 cannot be loaded due to missing prereq.
HINT: the following modules must be loaded first: bgq-xl/1.0
```

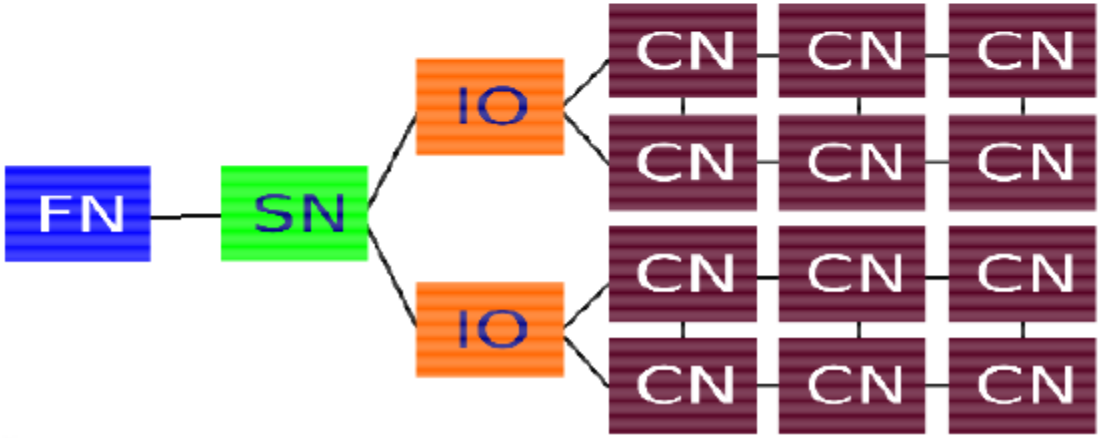
Load the "autoload" module which checks the dependencies and load all the modules needed:

```
$ module load autoload boost
```

So far so good, BUT...

Blue Gene Blocks Hierarchical Organization

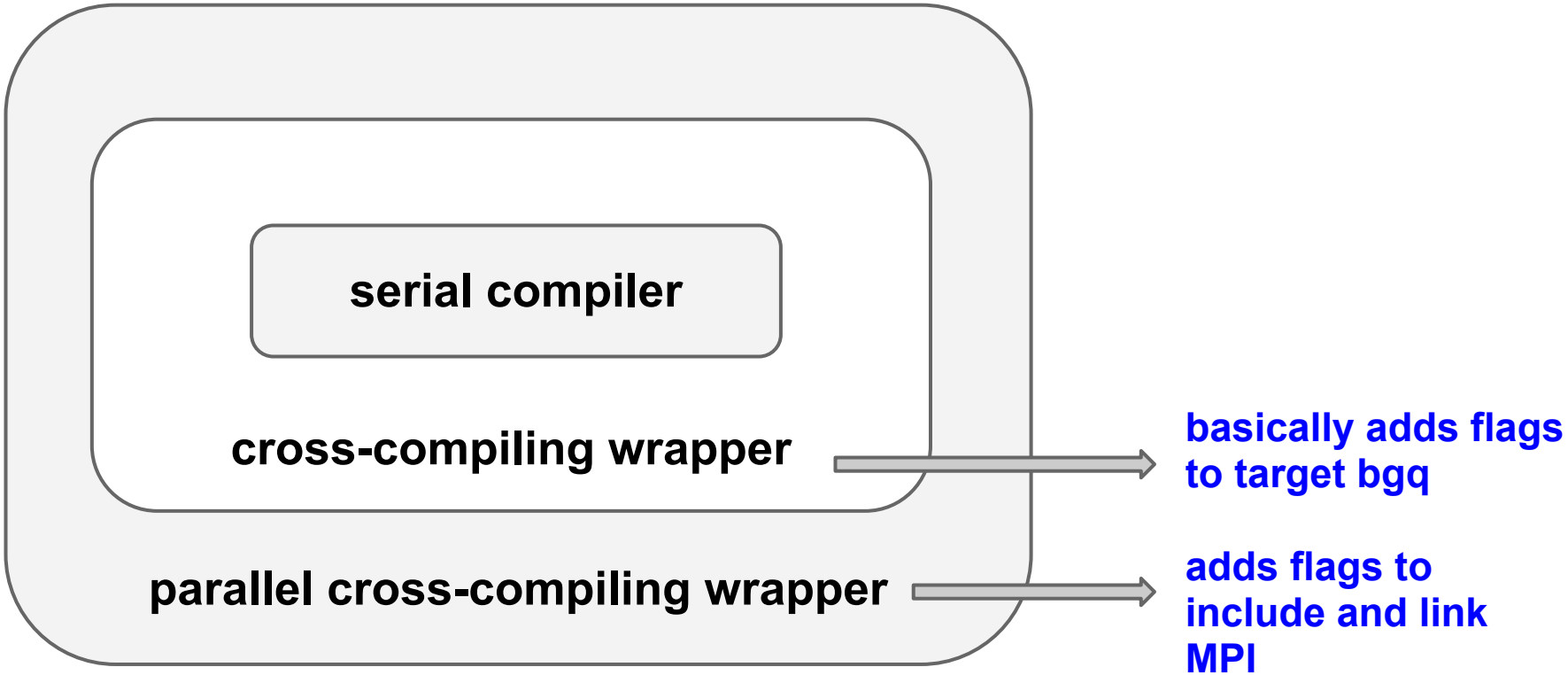
- **Front-end nodes (FN)**, dedicated for user's to login, compile programs, submit jobs, query job status, debug applications
- **Service nodes (SN)**, perform system management services, create and monitoring processes, initialize and monitor hardware, configure partitions, control jobs, store statistics
- **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



Cross-Compilation

- Architectures of **compute nodes** and **login nodes** are different (both cpu and O.S.)
- ... and you cannot log into the compute nodes
- you need to **compile on the login nodes targeting the compute node** architecture (***cross-compiling***)
- you can rely on the available **wrappers** (they do all the dirty work for you)
  - it's only matter of **selecting the right wrapper**

Compiler Hierarchy



Compiler Families

Two Different compilers family for both front-end and back-end nodes

- **IBM Compilers**
- **GNU Compilers**

	Back-end Compilers		Front-end Compilers	
	XL family	GNU family	XL family	GNU family
C	bgxlc, mpixlc_r	gcc, mpicc	xlc	gcc
C++	bgxlc++, mpixlcxx	g++. mpicxx	xlc++, xlC	g++
Fortran	bgxlf,bgxlf90,… mpixlf90,…	gfortran, mpif90	xlf, xlf90,…	gfortran

Cross compilation:
mpixlc –O3 **–qarch=qp –qtune=qp** myprog.c

Compilation:
xlc –q64 myprog.c

Tasks / Threads

Parallel programming is a programming technique that involves the use of *multiple processors* working together *on a single problem*. The global problem is split in different sub-problems, each of which is performed by a different processor in parallel. The code needs a programming language that allows to formally describe non-sequential algorithms. ... And of course you need the right machine architecture!

Parallel Program: different tasks communicate with each other to achieve an overall computational target.

Process

- Algorithm : the sequence of logical steps that must be followed to solve a given problem.
- Program : implementation of the algorithm, by means of a suitable formalism (programming language) so that it can be executed on a specific computer.
- Sequential process : sequence of events (execution of operations) which gives place the computer when operates under the control of a particular program.

A **task** is a Unix process.

TASKS...

- ... have their own memory space
- ... run a single instance of a serial application or a *Message Passing* Interface (MPI) application.
- ... can belong to different users
- ... can be different programs that a single user is running concurrently

A **thread** is an independent instruction stream, but as part of a Unix process.

THREADS...

- ... run in a *Shared Memory* space
- ... can have private data, but they can collaborate on the same data.
- ... are part of **one process** and therefore share each other's data.

IBM XL common options

Option	Meaning
-qarch=qp	Produces object code for the BGQ platform and: Enables BGQ vector data type Sets the –qsimd=auto option
-qtune=qp	Default with –qarch=qp or without –qarch, -qtune options and bg-prefixed compilers. Also set if specifying –q64 or –O4,-O5
-q64	Sets 64-bit compiler mode
-qstaticlink	The compiler links only static libraries with the object file being produced. (Enabled by default; specify -qnostaticlink to dynamically link your programs).
-qsimd	-qsimd=auto enables automatic generation of QPX vector instructions. Enabled by default at all optimization levels. To disable automatic generation of QPX instructions, use -qsimd=noauto.
-qsmp	Enables parallelization of program code. -qsmp=omp enables strict openMP compliance. The -qsmp option must be used together with thread-safe compiler invocation modes (_r-suffixed)

Example

COMPILING...

```
$ module purge
$ module load bgq-xl/1.0 lapack/3.4.1--bgq-xl--1.0
$ module list
Currently Loaded Modulefiles:
 1) profile/base                3) lapack/3.4.1--bgq-xl--1.0
 2) bgq-xl/1.0
$ mpixlf90_r -o check.x check.f90 -L$LAPACK_LIB -llapack
** checkmpi    === End of Compilation 1 ===
1501-510  Compilation successful for file check.f90.
```

... WITH OPTIONS

```
$ module purge
$ module load bgq-xl/1.0 lapack/3.4.1--bgq-xl--1.0
$ module list
Currently Loaded Modulefiles:
 1) profile/base                3) lapack/3.4.1--bgq-xl--1.0
 2) bgq-xl/1.0
$ mpixlf90_r -O2 -qlistopt -qreport -qsmp=omp -o check.x check.f90 -
L$LAPACK_LIB -llapack
** checkmpi    === End of Compilation 1 ===
1501-510  Compilation successful for file check.f90.
```

Libraries: Use the right **LINK**!

- Many compilation errors are due to wrong or incomplete library linking (**undefined reference**): don't panic!
- Remember to load your modules (module avail, module load):

```
module load library/version
```

(fftw/2.1.5--bgq-xl--1.0, lapack/3.4.1--bgq-xl--1.0... ecc.)

- all library paths are in the form $LIBRARY**_LIB** ($FFTW**_LIB**, $LAPACK**_LIB** ecc.) ; include paths are in the form $LIBRARY**_INC**

```
$ module load hdf5/1.8.9_ser--bgq-xl--1.0
$ ls $HDF5_LIB
libhdf5.a      libhdf5_cpp.la     libhdf5_fortran.la  libhdf5_hl_cpp.a
libhdf5hl_fortran.a   libhdf5_hl.la  libhdf5.settings libhdf5_cpp.a
libhdf5_fortran.a  libhdf5_hl.a  libhdf5_hl_cpp.la  libhdf5hl_fortran.la
libhdf5.la
```

How to find which library I need?

Use the command "nm" to find the reference and the right library to link:

```
$ 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
```

2 ways to link a library:

-L$LIBRARY_LIB **-l**name --- or --- $LIBRARY_LIB/**lib**name**.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/libhdf5.a $SZiP_LIB/libsz.a $ZLIB_LIB/bibz
a
```

Example:

```
$ module load bgq-xl
$ module load hdf5/1.8.9_ser--bgq-xl--1.0
$ module load szip zlib
$ mpixlc_r -I$HDF5_INC input.c -L$HDF5_LIB -lhdf5 -L$SZIP_LIB -lsz -L$ZLIB_LIB -lz
```

Optimization with IBM XL

- Add **-qlistopt -qreport** to generate a file.lst containing the optimization flags

- **SIMD** (**S**ingle **I**nstruction **M**ultiple **D**ata) vectorization activated by default where possible (deactivate it with **-qsimd**=noauto)

  NB: loops are not SIMD vectorized when step "i" depends on step "i+1" or "i-1"

- **Unrolling** loops: **-qunroll**=yes (default with -O3 optimization, deactivate it with **-qnounroll)**

- -O3 optimization uses -qnostrict: the semantic of your code could be altered. If you want to keep the semantic use -qstrict option

- **Threading**: **-qsmp[**=auto|omp|...]
  N.B "auto" means automatic parallelization of loop. If you want the OpenMP directives to be interpreted specify "-qsmp=omp" (noauto is implied)
  -qsmp without suboptions means:

  -qsmp=auto:opt:noomp:norec_locks:nonested_par:schedule=auto:nostackcheck:threshold=100:ostls:speculative

for further info: http://pic.dhe.ibm.com/infocenter/compbg/v121v141/index.jsp

Problem definition

- User asks about performance issues of his own code
- Performance seems rather bad on FERMI
- User claims the code on his desktop runs much faster
  - on his **desktop**:
    - 2 processes,
    - number of iterations (each): **20480**
    - **164 sec**
  - on **FERMI**:
    - 1024 processes,
    - number of iterations (each): **40**
    - **7.6 sec**
- 20480/40 = **512;** 164/7.6 = **21.6** (**unacceptable!!**) Let's see what is going on...

Profiling of the code

1) std compilation w/ **-pg -g**
2) $ /bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gprof ..
/hmdrabgp-m2l50eo-nomp.exe gmon.out.0 >gprof.out.0

Flat profile:
Each sample counts as 0.01 seconds.
% cumulative self self total
time seconds seconds calls s/call s/call name
82.78 234.64 234.64 7040 0.03 0.04 .buildyx1
14.61 276.05 41.41 42240 0.00 0.00 .yanna1
0.45 277.32 1.27 7040 0.00 0.04 .doallnop
0.15 277.75 0.43 .LDScan
0.12 278.10 0.35 11 0.03 25.26 .buildho
0.10 278.38 0.28 .zslvlvf
0.10 278.66 0.28 .zslvuvf
0.09 278.91 0.25 .zlaswp

83% of time is spent in this function

Performance on FERMI (1)

4 compilations

- mpixlf77 -q64 **-O0 -qlistopt -qreport -qsource** (**user compilation**)
- mpixlf77 -q64 **-O3 -qstrict -qlistopt -qreport -qsource**
- mpixlf77 -q64 **-O3 -qhot -qlistopt -qreport -qsource**
- mpixlf77_**r** -q64 **-O3 -qhot -qsmp -qlistopt -qreport -qsource**

Check .lst files to see the improvement on **buildyx1** function

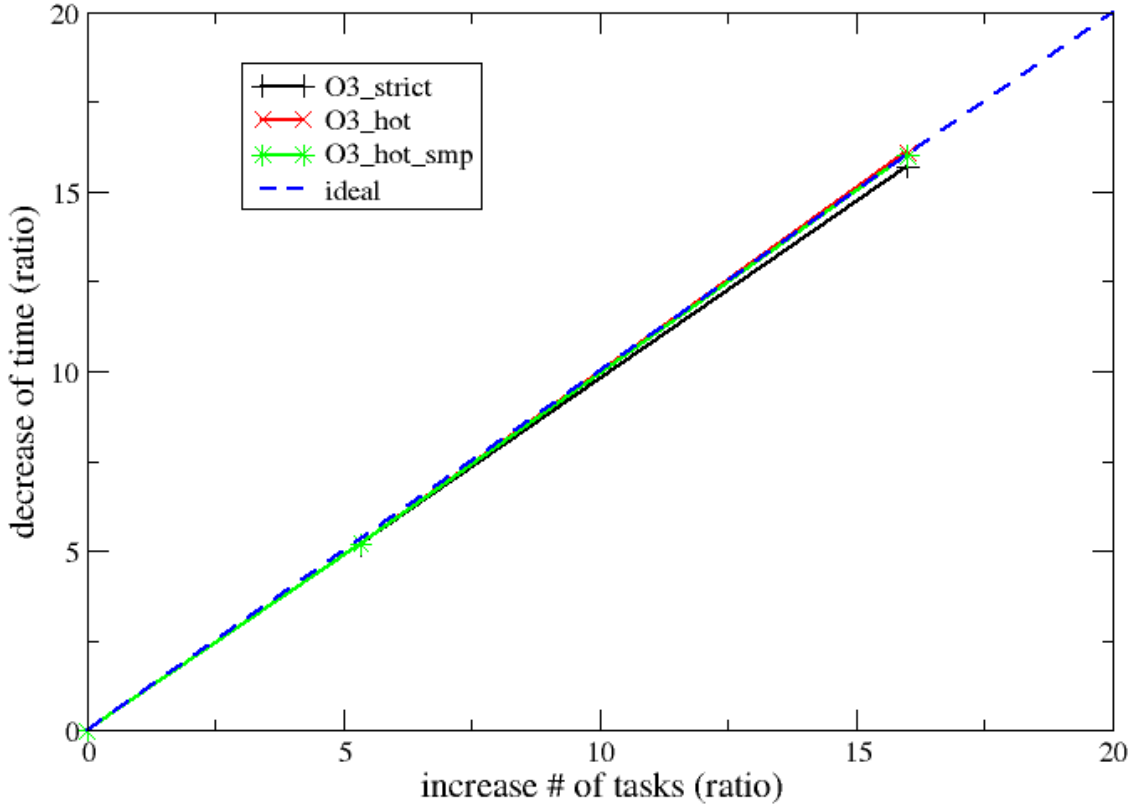
Performance on FERMI (2)

# tasks	-O0	-O3 -strict	-O3 -qhot	-O3 -qhot -qsmp
2			697.0	
12	646.0	154.8	135.0	**118.8**
64	122.0	29.9	25.8	**22.8**
1024	**7.6**	1.9	1.6	**1.4**

results are in seconds

Performance on FERMI (3)

Scale-up (strong scaling)



Can we squeeze a little bit more?

- Changing the code: **dynamic allocation** allows for using SMT!
- one can "feed" a single processor w/ 1, 2, 4 processes (exploiting 2-way concurrent issue, 1 int + 1 float, and decreasing cpu-memory latency).

# tasks	ranks-per-node	compute nodes	O3 -qhot -qsmp
64	16	**4**	**26.7**
128	32	**4**	**17.5**
256	64	**4**	**13.1**
1024	16	**64**	**1.7**
2056	32	**64**	**1.0**
4096	64	**64**	**0.8**

PLX (**Intel CPU**) 64 tasks: **3.8** sec **~ 3 times faster**

Before I came here I was confused about this subject.
Having listened to your lecture I am still confused.
But on a higher level.

E. Fermi

- **FERMI reference guide**:

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

- **IBM compilers guide**:

  http://pic.dhe.ibm.com/infocenter/compbg/v121v141/index.jsp

- **Stay tuned with the HPC news:**

  http://www.hpc.cineca.it/content/stay-tuned

- **"man" command:** man bgxlf90, man bgxlc, man rsync ...

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

- ... And if you are highly motivated to better understand HPC... (Or just curious!)

  V. Eijkhout, *Introduction to High Performance Scientific Computing* (2011)