UG3.4.x: FERMI UserGuide

(updated: Unknown macro: 'page-info')

Attention

FERMI is no more available at CINECA

In this page:

hostname:	login.fermi.cineca.it
early availability:	July 30th, 2012
start of production:	Aug 8th, 2012

Technical sheet	
Model: IBM-BlueGene /Q **Architecture: 10 BGQ Frame with 2 MidPlanes each** **Front-end Nodes OS: Red-Hat EL 6.2** **Compute Node Kernel: lightweight Linux-like kernel** **Processor Type: IBM PowerA2, 1.6 GHz** **Computing Nodes: 10.240 with 16 cores each** **Computing Cores: 163.840** **RAM: 16GB / node; 1GB/core** **Internal Network: Network interface with 11 links ->5D Torus** **Disk Space: more than 2PB of scratch space** **Peak Performance: 2.1 PFlop/s**	

System Architecture

FERMI is a BlueGene/Q system and has a massively parallel architecture. In the following picture we show how the system is built up.

Each Compute Card (which we call a **"compute node"**) features an IBM PowerA2 chip with 16 cores working at a frequency of 1.6 GHz, 16 GB of RAM and the network connections. A total of 32 compute nodes are plugged into a so-called Node Card. Then 16 Node Cards are assembled in one midplane which is combined with another midplane and two I/O drawers to give a rack with a total of 32x32x16 = 16K cores for each rack.

In our BG/Q configuration there are **10 racks for a total of 160 K cores**.

In addition to the compute nodes there are also front-end nodes (or login nodes) running Linux for interactive access and the submission of batch jobs. Parallel applications have to be cross-compiled on the front-end nodes and can only be executed on the partition defined on the compute nodes. Access to the compute nodes are mediated by I/O nodes, since only them are able to interact with the file systems.

On the **login nodes** there is a complete Red-Hat Enterprise Linux (6.2) distribution. On the **compute nodes** instead there is a light Linux-like kernel called Compute Node Kernel (CNK). Compute nodes are diskless and I/O functionalities are provided by dedicated I/O nodes. **I/O Nodes** run Linux and provide a more complete range of OS services, e.g.: files, sockets, process launch, signaling, debugging, and termination. **Service Nodes** perform system management services (e.g., partitioning, heart beating, monitoring errors) and can be used only by system administrators.

The CINECA configuration is made of 10 racks as follows:

- 2 racks: 16 I/O nodes per rack, implying a minimum job allocation of 64 nodes (1024 cores).
- 8 racks: 8 I/O nodes per rack, implying a minimum job allocation of 128 nodes (2048 cores).

The minimum allocations result from the fact that a job must have at least one I/O node allocated to it. Therefore, even if your job asks for an arbitrary number of nodes or cores, a minimum partition of 64 or 128 nodes (or multiples) will be allocated and accounted!

Access

There are several login nodes (4 to 8) used for interactive access, data movement and batch submission. They have an identical environment and can be reached with **SSH (Secure Shell)** protocol using the "collective" hostname:

```
login.fermi.cineca.it
```

wich establishes a connection to one of the available login nodes.

Files from other computers can be moved to FERMI using **sftp** or **scp**:

```
sftp login.fermi.cineca.it
> user:
> passw:

scp pippo.dat user@login.fermi.cineca.it:pippo.dat
> passw:
```

Disk and Filesystems

FERMI is completely consistent with the general CINECA infrastructure (see Sect "Data Storage and File Systems").

	Tot Dimension (GB)	**Quota (GB)**
$HOME	100 TB	50 GB

$CINECA_SCRATCH	900 TB	no quota
$WORK	1.800 TB	1 TB (or as required by the project)

At present the $WORK space is defined only on FERMI.

Only the $CINECA_SCRATCH and $WORK areas should be used for running programs.

Since these filesystems are all based on GPFS (General Parallel File System), the usual unix command "quota" will not work. Instead use the local command "cindata" to query for disk usage and quota ("cindata -h" for help):

```
  > cindata
```

In addition to these filesystems, a long term archive area on tape, $TAPE, is available upon request. Finally, a new storage resource, $DRES, was added on the 1st October 2014. It is intended as a medium/long term repository and as a shared area within the project tema and across HPC platforms. Also this resource is created upon request, please, refer to the relevant page in the General Information section.

Production Environment

The FERMI production environment is completely consistent with the CINECA infrastructure described in the "Production Environment and tools" section. It consists of scientific applications available from the Software catalogue, a batch scheduler for submitting jobs, the "module" environment and tools for data management.

The scheduler of the FERMI system is **LoadLeveler (LL)**. The scheduler is responsible for managing jobs on the machine by allocating partitions for the user on the compute nodes, returning job output and error files to the users. LL with special emphasis on the BG/Q keywords, is discussed in the "Batch scheduler LoadLeveler" Section.

In order to make correct requests to the LL, just remember that the FERMI compute nodes are not shareable, i.e. a compute node cannot run multiple jobs at the same time. Moreover, compute nodes are allocated in groups called *partitions* of fixed size. You can ask for an arbitrary number of compute nodes, but the LL will allocate them for you as fixed (an maybe larger) partitions. The smallest allocation is 64 (or 128) nodes, for a total of 1024 (or 2048) cores depending on the job class (see below). A batch job is charged for the number of nodes allocated multiplied by the number of cores per node (16) and the wall clock time used.

At present, on the FERMI machine, the following *public* classes are defined. Please note this schema is not final and some changes could occur. In this case users will be notified via the HPC-news mailing list.

- **debug**: for testing jobs, short time and few ComputeNodes.
- **longdebug:** for testing jobs and for production extra small size jobs, longer time and minimum parallelism.
- **smallpar:** for production small size jobs, longer time and small parallelism.
- **parallel**: for production medium size jobs, longer time and medium parallelism.
- **bigpar:** for production large size jobs, longer time and high parallelism.
- **keyproject**: for production extra large jobs. Normally available only for selected users and under request
- **serial**: for jobs of post-processing, compilation and for moving data to/from CINECA_DATA/PROJECT and the cartridge system.

queue name	ComputeNodes	max wall time	defined on	notes	priority
debug	64	00:30:00	2 rack with 16 I/O nodes	Maxqueued(*)=1	high
longdebug	64	24:00:00	2 rack with 16 I/O nodes	Maxqueued(*)=2	low
smallpar	128	24:00:00	8 rack with 16 I/O nodes	Maxqueued(*)=4	high
parallel	256 - 512	24:00:00	2 rack with 8 I/O nodes	Maxqueued(*)=2	high
bigpar	1024 - 2048 (1-2 racks)	24:00:00	6 rack with 8 I/O nodes	Maxqueued(*)=2	very high
keyproject	1024 - 6*1024 (1-6 racks)	24:00:00	8 rack with 8 I/O nodes	Ask to superc@cineca.it Due to the geometry of the system, the max number of usable racks is 6	very high
serial	1 core	06:00:00	front-end node	to be requested with @#class=serial	high

(*) For a user's job to be allowed into the job queue and then dispatched the total of other jobs (in the Idle, Pending, and Running states) for that user must be less than the maxqueued value for that user. If it is more than the maxqueued value the submitted job will still be accepted as "not-queued" (NQ) and its priority will be assigned only when it passes from the NQ to the queueing state. i.e. when some of the queued jobs are completed or canceled from the queue. This is also valid for multi-step jobs.

For more information about LoadLeveler and about the syntax of the batch script, please see the specific chapter of the Guide (Batch Scheduler LoadLeveler).

Budget linearization policy

Starting on May the 5th, 2014, a new policy on the use of project budgets on FERMI has been implemented.

For each project/account_no, a monthly quota is defined as (total_budget / total_no_of_months). Each month, starting from the first day of the month, the collaborators of that project are allowed to use the quota at full priority; after that, their jobs will still be considered for execution, but with a lower priority.

This policy is similar to those already applied by other important HPC centers in Europe and worldwide. The aim is to improve the response time, giving users opportunities related to the actual size (total amount of core-hours) of their projects.

Please note that, in this way, you will be forced into applying a sort of "linearization" of your project budget. Each month a given percentage of your budget is guaranteed, but non-linear usage is discouraged.

Programming Environment

This section is relevant for those interested in writing their own codes and running them on the FERMI system. The environment is consistent with the general HPC programming envirnoment in CINECA as discussed in the section "Programming Environment" and is made of compilers, mathematical libraries, debuggers, profilers and general tools for helping users in writing and executing their own codes.

Is my application “suitable” to BGQ architecture?

The first thing is considering if your code is a good candidate for the FERMI system. Its main features are: (i) the high number of cores that impose a very high level of parallelisation (starting from 1024 cores) and (ii) the small memory allocated to each MPI task (depending on applications in the range 256MB - 16GB).

You should try to answer the 4 questions reported below in order to understand if this architecture is suitable for your code:

1. Does the code use the Message Passing Interface MPI? (needed to scale up to 1024 processes).
2. Is the memory requirement per MPI task less than 1 GB (pure MPI) or 16 GB (MPI+OpenMP)? Is it possible to exploit SMT technology?
3. Is the code computationally intensive? That is, is there a small amount of I/O compared to computation? Is the code floating-point intensive?
4. Does the algorithm allow for distributing the work to a large number of cores? How does it scale up to with thousands of threads/tasks?

If your answers are all "YES", well your code is a very good candidate for FERMI. If not, you should consider to modify the code, in order to best adapt it to the BG/Q architecture.

Compilers

On BlueGene systems you generally connect to the login node and use the compiler to build the executable for the compute nodes. Since the Login and the Compute nodes have different architectures, you need the so-called **cross-compilation**, that means to instruct the compiler to compile for a different platform. This is done by using specific "wrappers" of the compilers containing the correct options for cross-compilation.

Two main families of compilers are available: **XL** (IBM native) and **GNU**. Both of them can compile for the Login (front-end) nodes and the Compute (back-end) nodes.

To cross-compile with XL or GNU load, respectively, the module:

```
module load bgq-xl
module load bgq-gnu
```

The compilers/wrappers for cross-compilation connected with the XL compiler are:

```
bgxlf,bgxlf90:      IBM Fortran compiler
mpixlf90:            (MPI wrapper)
bgxlc,bgcc:         IBM C compiler
mpixlc:               (MPI wrapper)
bgxlc++,bgxlC:   IBM C ++ compiler
mpixlcxx:            (MPI wrapper)
```

(add the "_r" suffix for thread-safe compiling)

Key options for XL compilers include:

```
-qarch=qp: Produces object code for the BG/Q platform
-qtune=qp: Optimizes for the BG/Q platform
```

These options are the defaults but can be added anyway to emphasise that the executables are being created for BG/Q compute nodes. Note also that it is not possible to compile in 32 bit mode on FERMI, i.e. the -q32 flag is not supported.

Other useful options include:

```
-qnostaticlink: Create executable with dynamic linking.
                         By default static executables are created (for performance).
```

The compilers/wrappers for cross-compilation connected with the GNU compiler are:

```
gfortran: GNU Fortran compiler
mpif77,mpif90: (MPI wrapper)
gcc: GNU C compiler
mpicc: (MPI wrapper)
g++: GNU C++ compiler
mpicxx: (MPI wrapper)
```

Since it is not possible to compile 32 bit executables the -m32 flag of the GNU compilers is not supported. Just as for the XL compilers the default linking mode is static. If you wish to create dynamic executables then use the -dynamic option.

Some useful remarks:

- “bg” compilers are for cross-compilation
- “mpi” are MPI wrappers for cross-compilation
- gcc compiler for login and compute nodes has the same name. Before using it be carefull to load the correct module.
- The _r-suffixed invocations allow for thread-safe compilation
    - use them if you want to create threaded applications
    - The -qsmp option must be used together with thread-safe compiler invocation modes

For example, to compile an MPI c program for the compute nodes:

```
 mpixlc_r  -O3 -qarch=qp -qtune=qp myprog.c -o prog
```

In case you need to compile a program (small uitlity) for the login node you have to use different compiler and the -q64 flag:

```
xlf, xlc, xlC               XL compilers for the Login node
gfortran, gcc, g++    GNU compilers for the Login node
```

Remember to load the front-end-xl or front-end-gnu module before using the XL or GNU compilers for the login nodes.

Mathematical libraries

A number of mathematical and I/O libraries are available on FERMI. Remember to load the module, before to use them.

- **Mathematical Libraries** (ESSL, BLAS, Lapack, Scalapack, blacs, PETSc, fftw, ...)
- **I/O Libraries** (HDF5, NetCDF)

ESSL libraries are collections of state-of-the-art mathematical subroutines specifically designed to improve the performance of engineering and scientific applications on the IBM systems. To compile with them you should load the module and add the reference when linking your code:

```
> module load essl
> mpixlf ... -L$ESSL_LIB -lesslbg
```

Debugging & Profiling

Three main tools are available for debugging purposes:

- addr2line (unix command, translates program addresses into file names and line numbers. Given an address and an executable, it uses the debugging information in the executable to figure out which file name and line number are associated with a given address)
- gdb (general debugger - GNU)
- totalview (very efficient debugger for massively parallel programs)

You can find a detailed guide about how to use these tools on FERMI here:

debug guide on FERMI

Among the "Performance Analysis tools", the most useful are:

- Automatically Available Performance Counters
- SCALASCA
- High Performance Computing Toolkit
    - MPI Profiler and Tracer
    - CPU profiler Xprofiler
    - Hardware Performance Monitoring
    - I/O Performance

References

- IBM Redbook – Application Development
- IBM Redbook – System Administration
- IBM – Compilers:
    - xl c/c++ for IBM Blue Gene/Q, V12.1
    - xl Fortran for IBM Blue Gene/Q, V14.1
- GNU - Compilers: http://gcc.gnu.org/

Outgoing links: Unknown macro: 'outgoing-links'