



## Porting Applications from standard platforms to the IBM Blue Gene platform :: What matters?

Porting an application to new a platform can be anything from a trivial task to notoriously difficult. In this document we will go through some points that should be taken into consideration when porting applications to the IBM Blue Gene platform. Many of porting issues are not particular to the Blue Gene platform, we will try focus only on Blue Gene specific problems. In the first section we give a brief overview of the Blue Gene hardware and software environment. Then we look at important points to take into consideration when porting to the Blue Gene.

### Hardware overview

A good understanding of the underlying hardware architecture is important when porting and tuning applications. The Blue Gene (BG) architecture consists of the following key components:

- Service Nodes which provide control of system.
- A Frontend Node which provides access to the users to submit, compile and build applications.
- Compute Nodes which run applications; users cannot login to these nodes.
- I/O Nodes which provide access to external devices. All I/O requests are routed through these nodes.
- High bandwidth networks for end user application use i.e. point to point & collective communications.
- Special purpose control networks e.g. JTAG.

Each BG/P compute node is based on 4 fully cache coherent 32-bit PowerPC cores with a clock frequency of 850MHz. Each core has a dual pipe double precision floating point unit. The four cores share 2GB of RAM. The BG/L compute node is based on two non-cache coherent 32-bit PowerPC cores with a frequency of 750 MHz. Each

compute node has 1GB of RAM. Both the BG/L & /P contain 1024 compute nodes. The BG/L with a peak performance of 5.73 TFlops and a linpack performance of 4.74 TFlops. The BG/P has a peak performance of 13.93 TFlops and a linpack performance of 11.11 TFlops. We refer the reader to:<http://www.top500.org> for the definition of peak and linpack performance.

### porting to the blue gene

Hardware overview	1
Networks	2
Software overview	2
Cross compilation	2
OpenMP	3
Memory Management	3
System calls	4
LoadLeveler	4
Signal 7	5



For an in depth architectural description of the Blue Genes see the the IBM Redbooks:

- <http://www.redbooks.ibm.com/abstracts/sg246796.html?Open>
- <http://www.redbooks.ibm.com/redpieces/abstracts/sg247287.html>

## Networks

One of the interesting aspects of the BG system is its internal networks. On the BG five networks are used for various tasks.

#1 Three-dimensional torus: The torus network is used for general-purpose, point-to-point message passing. In this configuration each node is connected to its six nearest neighbours. A fully connected torus is only available when using jobs which occupy one or two complete midplanes, i.e. half or all of the system.

#2 Global collective: The global collective network is a high-bandwidth, tree based network that is used for collective communication operations, such as broadcasts and reductions, and to move process and application data from the I/O nodes to the compute nodes.

#3 Global interrupt: The global interrupt network is a separate set of wires based on asynchronous logic, which forms another network that enables fast signalling of global interrupts and barriers (global AND or OR).

#4 10 Gigabit Ethernet: File I/O and host interface. The 10 Gigabit Ethernet (optical) network consists of all I/O Nodes and discrete nodes that are connected to a standard 10 Gigabit Ethernet switch.

#5 Control: The control network consists of a JTAG interface with direct access to shared SRAM in every compute and I/O node. It is used for boot, monitoring, and diagnostics.

This choice of networks is normally transparent to users and developers in the sense that they can not directly decide which networks to use. However, this information can serve as guide to the developers to choose appropriate communications types. For example knowing that global communications will use a dedicated network gives an extra motivation to the developers to favour global communications over point-to-point communications where possible.

## Software overview

For development the IBM XL family of C/C++ and Fortran compilers are recommended. GCC is also available as is the IBM Engineering and Scientific Subroutine Library (ESSL). The implementation of MPI on the BGs is based on MPICH2 developed by Argonne National Labs. One of the aspects of the MPI-2 standard that is not supported by BG is Dynamic Process Management (creating new MPI processes). However, the various threading modes are supported on the BG/P. The XL compiler supports OpenMP based multithreading. Threading can be done automatically by the compiler by using the `-qsmp` option. However this automatic approach can be counter productive and is not recommend. Conventional OpenMP pragma based threading is more likely to be successful.

## Cross compilation

Two different set of wrappers are available to invoke the XL compilers. For C/C++ one has `xlc/xlc++` and `bgxlc/bgxlc++`. For Fortran one has `xlf/xlf90` and `bgxlf/bgxlf90`.

```
$ xlc myprog.c -o myprog
```

Cross compilation is required on the BG, as all compilation is done on the frontend. Wrappers for compiling specifically for the backend nodes have the

```
$ bgxlf90 myprog.f90 -o myprog -V
```

same name as the xl compiler variants but start with the letters "bg" i.e. compiling with `xlc` and `bgxlc` will produce different binaries by default. Rather than using the `bg` wrappers compiler flags can be used with to target a particular platform. For example `-qtune=450d` will instruct the compiler to tune the code for BG/P compute nodes. The thread safe version of these compilers are invoke by adding an `_r` to the names above. That is `xlc_r` is the thread safe version of `xlc`.

Some application's default build process will fail or give incorrect results because the compilation is done on the frontend whereas the code is meant to run on the backend. Similarly some build systems, using config tools for example, perform tests to check whether the compilation has been successful. These tests will be carried out on the frontend nodes even though the binaries are created for the backend nodes and so will likely fail. To run these compilation tests, one should submit them as jobs to the queuing system (loadleveler, discussed later). Users should also note that the default compiler optimization level is `-O0` instead of `-O2` as is the case for many compilers.

## Cross Compilation

Cross compiling means compiling a code on a platform which differs from that on which it is intended to run. i.e. the BG frontend node is not the same architecture as the backend nodes. Cross compilation is a common practice in embedded device development where one might develop on a conventional workstation yet deploy code to something quite different such as a mobile phone.

## Using double floating point unit

### -qarch=440d/450d

By default the XL compiler will try to use both double floating point units available on backend node cores.

The compiler does this by setting the `-qarch=440d/450d` flag (440d on the BG/L or 450d on the BG/P). This option can cause problems for certain applications. If an obscure floating point related error occurs at runtime, one could try to use only one floating point unit by setting `-qarch=440/450`. For example:

```
$ bgxlf90_r -qarch=450 myprog.f90 -o myprog
```

will instruct the compiler to use one floating point unit only.

## Loading modules

In order to use these compilers on the BG systems you need to load the relevant module. Module loading sets the appropriate paths for binaries and libraries and other environment variables. The following command will load the

```
$ module load bgp
```

the BG/P development environment. And all subsequent invocations of an XL compiler will refer to the compilation for the BG/P. As you might expect a `bgp` module also exists.

## Word length 32/64 bit

It is important to bear in mind when porting/developing applications to the BG that the backend processors are 32bit processors. In short, users applications should not require 64-bit mode. In terms of compilation, it means that something like `bgxlf90 -q64` is not valid. Users should also note that The 440/450 processors do not support VMX instructions or vector data types (`-qaltivec`).

```
$ bgxlc myprog.c -o myprog -q64
1506-807 (W) Incompatible
specifications for options -qarch and -
q64 (or environment variable
OBJECT_MODE) 1506-809 (W) Incompatible
specifications for options -qtune and -
q64 (or environment variable
OBJECT_MODE)
...
```

## Multi-threading programming on the BG with OpenMP

The BG/P system supports shared-memory parallelism on single nodes. The XL compilers support the following:

- Full support for OpenMP 2.5 standard.
- Interoperability with MPI
  - ▶ MPI at outer level, across the compute nodes.

```
$ mpixlf90_r -qcclines -qrealsize=8
-qsmpt=omp:auto:noopt:stackcheck
-qarch=450 myprog.F90 -o myprog
```

- ▶ OpenMP at the inner level, within a compute node.
- Autoparallelisation as a loop level optimisation.
- Use of the thread-safe compiler version with any threaded, OpenMP, or SMP application:
  - ▶ `-qsmpt` must be used on OpenMP or SMP applications.
  - ▶ `-qsmpt` by itself automatically parallelises loops.
  - ▶ `-qsmpt=omp` parallelises based on OpenMP directives in the code.

The above command shows the compilation of a program with the option `-qsmpt` followed by sub-options. Note that in general shared memory programming on BG implies hybrid parallelization since the multi-threading is done within a node which is usually running one mpi task. In other words, the BG systems is not meant to run pure OpenMP programs.

## Linking static/shared libraries

On the BG/P both static and dynamic linking are supported while on the BG/L only static linking is supported. Static linking is the default the /P as it gives marginally better performance and dynamic linking is a relatively new feature.

## Memory management

Because of the relatively small amount of physical memory available to a given compute node on a BG system, particular attention should be paid to an application's memory requirement. On the BG/L, the memory requirement per MPI process should be kept below 1GB in coprocessor mode and below 512MB in the virtual node mode. In the case of the BG/P system, if the memory requirement per MPI process is greater than 512MB in virtual node mode or greater than 1GB in dual node mode, then the application will not run correctly. In SMP mode, 2GB of memory can be used. It is therefore very important to be able to reduce the memory footprint.

## Unsupported system calls

The compute node kernel is a single-process operating system. It is designed to provide the services that are needed by most applications that are expected to run on the BG systems, but unfortunately there are exceptions. Users should be aware that there are a number of unsupported system calls. Below we list few of them; again, for complete list refer to the IBM System Blue Gene Solution: Blue Gene/P Application Development guide <http://www.redbooks.ibm.com/redpieces/abstracts/sg247287.html> section 6.6 on page 60 at time of writing. Some common unsupported system calls:

- `system()` function. This call is not supported since it uses `fork()` and `exec()` (via `libc`) which are currently not implemented.
- The BG does not supported `gethostname()` or `getlogin()` as Unix normally provides.
- Calls to `usleep()` are not supported.

## Running Applications, Job Launching

The BG systems use the IBM LoadLeveler batch processing system. Parallel applications are started by invoking `mpirun`. LoadLeveler provides a set of functions for job scheduling and cluster resource management. The most important LoadLeveler's commands are listed in the following table.

Frequently Used Batch System Commands	
<code>llstatus</code>	Show the LoadLeveler status
<code>llclass</code>	List the available classes (queues)
<code>llq</code>	List running and queued jobs
<code>llsubmit</code>	Submit a job script to LoadLeveler
<code>llcancel</code>	Cancel a running or queued job

## Sample LoadLeveler submit script

```
#@ job_type = bluegene
#@ bg_size = 512
# Specify which cluster to run on BG/L or BG/P
# (most loadleveler commands have a -X argument for cluster usage)
#@ cluster_list = BG/L
#@ input = /dev/null
#@ output = octave.$(jobid).out
#@ error = octave.$(jobid).err
#@ wall_clock_limit=14:00:00
#@ class = 512_24hrs
# Change the value for account_no to your project code
#@ account_no = sys_test
#@ queue
# The mpirun location is different for the /L and the /P
# load the correct environment module and use "which mpirun"
# to find the appropriate path
# See BlueGene documentation for mpirun arguments or use mpirun -h
/bgl/BlueLight/ppcfloor/bglsys/bin/mpirun -mode VN -np 1024
-env "MPITB_HOME=/ichec/home/staff/htapamo/octave/mpitb"
-cwd /ichec/home/staff/htapamo/projects/simulations/2_6_6
-exe /ichec/home/staff/htapamo/octave/octave-3.0.1b/bin/octave
-args my_octave_script
```

## Binary data handling: endianness

Special attention should be paid when manipulating binary data files. While PowerPC processors can generally support both big and little endian, in practice a system will adopt one approach. Blue Gene systems use big-endian.

## Signal 7 Issue

Some users may find that some BG/P jobs exit with a segmentation fault and signal 7 (SIGBUS). This is due to an undocumented feature on the BG/P that deals with memory alignment during I/O operations. This feature allows the user to set a memory alignment error threshold for I/O operations in their job but results in confusion due to its default value of 1000. This threshold can be controlled using an environment variable. Die as soon as a memory alignment error is encountered during I/O:

```
-env BG_MAXALIGNEXP=1
```

Silence all memory alignment errors during I/O:

```
-env BG_MAXALIGNEXP=-1
```

Configure a threshold to workaround occasional memory alignment errors during I/O (eg. 2000 in this case):

```
-env BG_MAXALIGNEXP=2000
```

Note that memory alignment errors during I/O do have the potential to reduce I/O performance and as such it is better to eliminate them rather than ignore them if possible. The Blue Gene/L system does not have this feature so jobs with memory alignment errors will continue to run but possibly with reduced I/O performance.