



Porting Applications from standard platforms to the IBM Blue Gene/P 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/P platform. Many of porting issues are not particular to a particular platform. However in the present document we will try focus only on Blue Gene/P specific problems. In the first section we give a brief overview of the machine hardware and software environment. Then we look at some important points to take into consideration when porting to Blue Gene/P.

Hardware overview

A good understanding of the underlying hardware architecture is important when porting and tuning applications. The Blue Gene/P (BG/P) 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 450 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. A BG/P cabinet contains 1024 compute nodes for 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.

Schrödinger, the Irish capability service BG/P, comprises 1024 nodes all contained in a single cabinet.

porting to the blue gene

Hardware overview	1
Networks	2
Software overview	2
Cross compilation	2
OpenMP	4
Memory management	4
System calls	4
LoadLeveler	4
Signal 7	5
Process Placement	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/P system is its internal networks. On the BG/P 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. When using smaller partitions of the machine, the network's topology is degraded to a 2 or 3D mesh i.e no closure of the rings in each dimension is ensured.

#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 highly specialised one that ensures the clock synchronisation for all nodes in the machine. Furthermore it transports global interrupts and supports MPI barriers.

#4 10 Gigabit Ethernet: This network uses optical cabling to interconnect all the I/O nodes and the storage infrastructure. All data accessed by the compute nodes is transferred via this network.

#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 select the most appropriate MPI communications types. For example knowing that global communications will use a dedicated network gives an extra motivation to the developer to favour global communications over point-to-point communications where possible. Furthermore, knowing that point-to-point communications are routed through a 3D torus topology might impact the parallelisation policy in ones code, by encouraging usage of MPI Cartesian topologies and nearby communications versus distant communications.

Software overview

For development the IBM XL family of C/C++ and Fortran compilers are recommended. If performance is not crucial or if the IBM compiler doesn't permit to install the tool you need, GCC is also available. It can be an easy fallback solution for most of the GNU packages, as this is their primary target.

The IBM Engineering and Scientific Subroutine Library (ESSL) is also available for the BG/P. The implementation

of MPI on the machine is based on MPICH2 developed by Argonne National Labs. One of the aspects of the MPI-2 standard that is not supported here is Dynamic Process Management (creating new MPI processes). However, the various threading modes are supported.

The IBM compiler supports OpenMP by using the `-qsmp=omp` option. Furthermore, if your code supports it, it is recommended to use the mixed MPI / OpenMP parallelisation as it is well supported on the machine and potentially allows more efficient code.

The Double Floating Point Unit

By default the IBM compiler will try to use both double floating point units available on backend node cores. The compiler does this by setting implicitly the `-qarch=450d` flag (450 stands for PowerPC450, which is the BG/P processor architecture, and d for double unit). However, one has to keep in mind that this option can cause problems for certain applications. So if an obscure floating point related error occurs at runtime, one could try to use only one floating point unit by explicitly setting `-qarch=450`. For example:
`$ bgxlf90_r -qarch=450 myprog.f90 -o myprog`
will instruct the compiler to use one floating point unit.

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 front-end node is not the same architecture as the back-end 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.

Cross compilation

One of the most important points you have to take into account while porting a code to BG/P is that you have to cross-compile it. Indeed, the login node you will use to compile the code and the compute nodes you will run it on have different and incompatible architectures. Therefore, you have to carefully select the right compiler version depending on the platform your code is meant to be run on, often depending on:

#1 If it is a proper HPC code to run in parallel on the compute nodes, use the BG/P cross compilers.

#2 If it is a pre or post processing tool, to be run on the login node, use the native compilers

	BG/P Compute Nodes	Front-End Node
IBM Compilers	mpixlc mpixlcxx mpixlf[77,90,95,2003]	XLC, XLC++ XLF[90,95,2003]
GNU Compiler Collection	mpicc mpicxx mpif[77,90]	gcc g++ gfortran
Available Compilers		

A few remarks about the above table:

- All the IBM compilers (for both BG/P and front-end nodes) generate by default non-thread-safe code. To ensure thread safety if needed, use the alternate version of each compiler by appending `_r` to the compiler name. For example, the thread-safe version of the BG/P C++ compiler is `mpixlcxx_r`.
- The Fortran 77 version of the front-end IBM compiler is `xlfc`, and not `xlfc77`.
- There are no MPI versions of the front-end compilers as the generated code is not meant to be run in parallel.
- Even in the unlikely case of compiling a non-MPI code for BG/P nodes, you can use the `mpixl__` compilers.

Some applications default build process will fail or give incorrect results because the compilation is done on the front-end whereas the code is meant to run on the back-end. Similarly some build systems, using configuration tools for example, perform tests to check whether the compilation has been successful. These tests will be carried out on the front-end 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 most compilers.

Loading modules

In order to use these compilers on the BG/P 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 BG/P development environment. All subsequent invocations of an XL compiler will refer to the compilation for the BG/P

```
$ module load bgp
```

Word length 32/64 bit

It is important to bear in mind when porting/developing applications to the BG/P 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 `mpixlf90 -q64` is not valid. Users should also note that the PowerPC 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 IBM compilers support the following features:

- Full support of the OpenMP 2.5 standard.
- Interoperability with MPI
 - MPI at outer level, across the compute nodes.
 - OpenMP at the inner level, within a compute node.

It is however important to remember to use the thread-safe compiler versions with any threaded, OpenMP, or SMP application. This is achieved by invoking the compilers with a trailing `_r`, such as in the following example:

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

The above command shows the compilation of a program with the option `-qsmpt` followed by a few sub-options. Note that in general shared memory programming on BG/P implies hybrid parallelization since the multi-threading is done within a node which is usually running one MPI task. In other words, the BG/P system 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 (the ancestor of the BG/P) only static linking was supported. However, unlike what is commonly found, static linking is the default on BG/P as it gives marginally better performance and dynamic linking is a relatively new feature. The drawback of this is that when new libraries or compilers are installed on the machine (such as new version of the MPI library), one has to recompile / relink codes for taking advantage of the newly installed version.

Memory management

Because of the relatively small amount of physical memory available on a given BG/P compute node, particular attention should be paid to application's memory requirement. On each BG/P node, the 2GB of memory will be distributed evenly between the different running processes. That is:

- Virtual Node mode: 4 processes per node, each with 512MB of memory available.
- Dual mode: 2 processes per node, each with 1GB of memory available.
- SMP mode: 1 single process per node able to access the whole 2GB of memory available.

Depending on the actual memory requirements of the code, one might have to run on DUAL or SMP mode rather than in VN mode. It is therefore very important to be able to access and ultimately to reduce the memory footprint. Furthermore it also reinforces the potential interest of the mixed parallelisation mode MPI/OpenMP, as it allows to take advantage of the extra available core on the nodes, depending on the mode required for running the code.

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/P 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/redbooks/pdfs/sg247287.pdf> 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/P does not support `gethostname()` or `getlogin()` as Unix normally provides.
- Calls to `usleep()` are not supported.

Running Applications, Job Launching

The BG/P system uses the IBM LoadLeveler batch processing system. Parallel applications are started by invoking `mpirun` into a LoadLeveler batch script. LoadLeveler provides a set of functions for job scheduling and cluster resource management. Its most important 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
#@ cluster_list = BG/P
#@ input = /dev/null
#@ output = octave.$(jobid).out
#@ error = octave.$(jobid).err
#@ wall_clock_limit=48:00:00
#@ class = 512_24hrs
# Change the value for account_no to your project code
#@ account_no = my_project_code
#@ queue
# See BlueGene documentation for mpirun arguments or use mpirun -h
# See BlueGene documentation for mpirun arguments or use mpirun -h
/bgsys/drivers/ppcfloor/bin/mpirun -np 1024 -mode DUAL -env "OMP_NUM_THREADS=2" -mapfile
TXYZ -cwd $PWD -exe $PWD/a.out -args my_args
```

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. Unlike x86 type systems, 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 (e.g. 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.

Process placement

By default, process placement on the allocated partition is done by mpirun with the XYZT mapping. This means that, on the 3D mesh or torus allocated for the job, process of rank 0 will run on node of coordinates (0,0,0), then process of rank 1 will run on node of coordinates (1,0,0): the X axis is primarily used for placing the processes. Then the Y axis will be used, and then the Z axis. Once each node of the partition will have a running process allocated on, then the T coordinate will be used (T probably standing for thread). The drawback of this approach is that processes of nearby ranks will be spread across nodes instead of being packed into the same nodes. To prevent this from happening, it is usually a good idea to use a mapping of type TXYZ. This can be obtained with the mpirun option:

```
-mapfile TXYZ
```