Getting Started on PowerOmics

Introduction

PowerOmics (PO) is Rice's IBM POWER8 compute cluster. The system contains 6 IBM S822L compute nodes. Each node has two twelve-core IBM POWER8 processors running at 3.02 GHz. Each core is capable of running eight threads resulting in 192 threads per node with a cluster-wide total of 1152 threads and 144 cores. Two nodes have 1TB of RAM with the remaining having 256GB of RAM. One node has two NVIDIA Tesla K80 cards.

All nodes are available to all users (subject to change due to special projects, maintenance tasks, etc.)

All nodes use both 10 Gigabit Ethernet and Infiniband interconnects.

Prerequisite for Using This System

All of the clusters that make up our shared computing resources run the Linux operating system (not Windows or MacOS operating systems). In order to effectively utilize the clusters you must have knowledge of Linux, how to navigate the filesystem, how to create, edit, rename, and delete files, and how to run basic commands and write small scripts. If you need assistance in this area please review the tutorials that are available on our web site.

Logging Into the Cluster

The PowerOmics login nodes can be accessed through Secure Shell from any machine on the Rice campus network and certain Texas Medical Center campus networks.

If you need off-campus access, please visit our Off-Campus Access Guide

To login to the system from a Linux or Unix machine, use the *ssh* command:

```
$ ssh -Y (your_login_name)@po.rice.edu
```

To transfer files into the cluster from a Linux or Unix machine, use the *scp* command:

```
$ scp some_file.dat *.incl *.txt (your_login_name)@po.rice.edu:
```

For more information about using Secure Shell, please see our Using SSH to Login and Copy Files Guide.

First Time Login
If this is the first time you are logging in and you need login instructions, please see our FAQ .

Login Nodes

Once you are logged in to the system, you are logged into one of several login nodes as shown in the diagram below. These nodes are intended for users to compile software, prepare data files, and submit jobs to the job queue. They are not intended for running compute jobs. Please run all compute jobs in one of the job queues described later in this document.

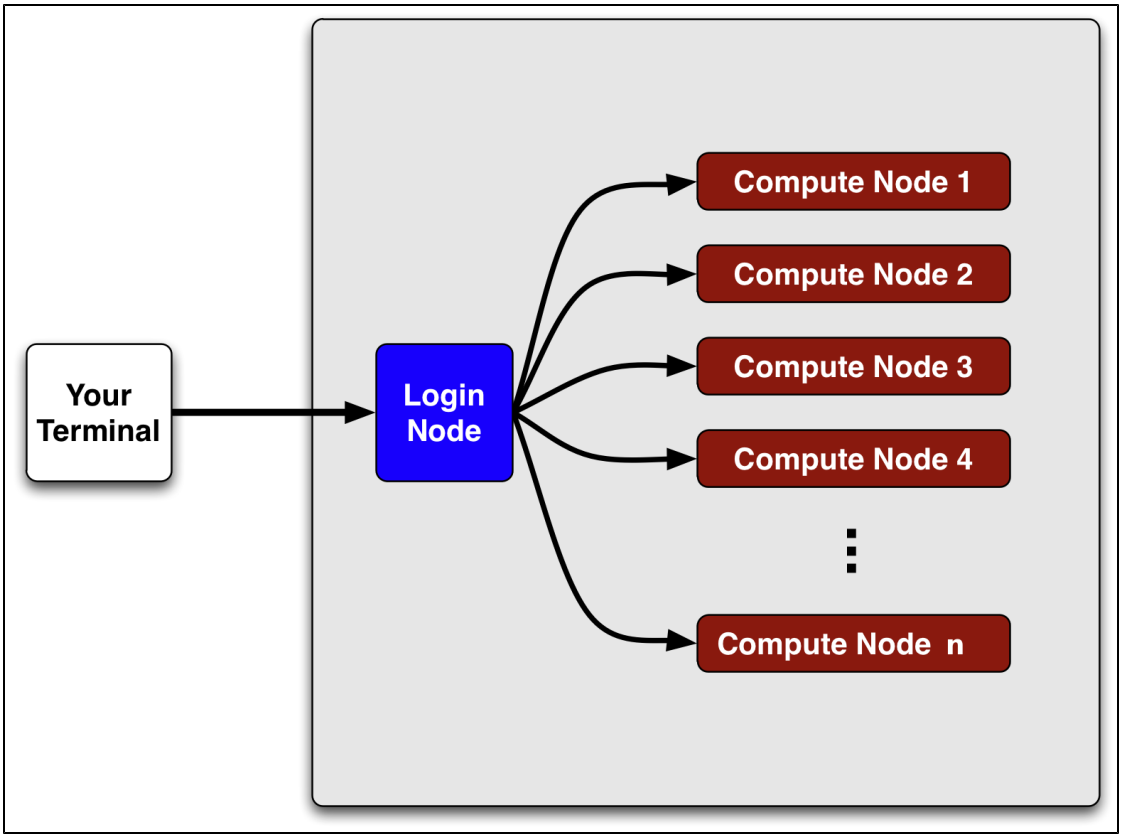

Diagram courtesy of Chris Hunter, Rice University

> ⚠ **Do not run compute jobs on login nodes**
> Cluster login nodes are multi-user access points intended for users to compile software, copy and prepare data files, and submit jobs to the job queue. Any user running intensive computational tasks directly on the login node risks **disciplinary action up to and including the loss of their access privileges**.

Data and Quotas

A summary of all filesystems available to all users is presented in the following table:

Filesystem	Accessed via environment variable	Physical Path	Size	Quota	Type	Purge Policy
Home directories	$HOME	/home	1.8 TB	5 GB	NFS	None
Group Project directories	$PROJECTS	/projects	18 TB	50 GB per group	NFS	None
Shared Scratch high performance I/O	$SHARED_SCRATCH	/poscratch	361 TB	None	GPFS	Every 14 days, but can be purged on short notice.
Local Scratch on each node	$LOCAL_SCRATCH	/tmp	375 GB	None	Local	When the compute job exits.

> ⚠

$SHARED_SCRATCH is not permanent storage
$SHARED_SCRATCH is to be used only for job I/O. Delete everything you do not need for another run at the end of the job or move to $WORK for analysis. Staff may periodically delete files from the $SHARED_SCRATCH file system even if files are less than 14 days old. A full file system inhibits use of the system for everyone. Using programs or scripts to actively circumvent the file purge policy will not be tolerated.

Data Backups Are the Responsibility of Each User
Backing up and archiving data remains the sole responsibility of the end user. At this point in time the shared computing enterprise does not offer these services in any automated way. We strongly encourage all users to take full advantage of Central IT's storage services, departmental servers, and/or individual group resources to prevent accidental loss or deletion of critical data. Please feel free to contact the CRC for advice on best practices in data management (e.g. IT's Subversion (SVN) services provide the safest environment for maintaining and developing source code). We welcome any suggestions for offering a higher level of data security as we move forward with shared computing at Rice.

Research Data Compliance

Due to recent changes in NSF, NIH, DOD, and other government granting agencies, Research Data Management has become an important area of growth for Rice and is a critical factor in both conducting and funding research. The onus of maintaining and preserving research data generated by funded research is placed squarely upon the research faculty, post docs, and graduate students conducting the research. It is imperative that you are aware of your compliance responsibilities so as not to jeopardize the ability of Rice University to receive federal funding. We will help in any way possible to provide you the information and assistance you need, but the best place to start is the campus research data management website.

To see your current quota and your disk usage for your home directory, run this command:

```
quota -s
```

To see the quota and usage for the $PROJECTS directories for all groups that you belong to, run this command:

```
quota -sg
```

To see the quota and usage for the $WORK directories on DAVinCI, and PO for the primary group to which you belong, run this command:

```
mmlsquota --block-size 1G -g $(id -gn)
```

The clustered file system $SHARED_SCRATCH provides fast, high-bandwidth I/O for running jobs. Though not limited by quotas, $SHARED_SCRATCH is intended for in-flight data being used as input and output for running jobs, and may be periodically cleaned through voluntary and involuntary means as use and abuse dictate.

Volatility of $SHARED_SCRATCH
The $SHARED_SCRATCH filesystem is designed for speed rather than data integrity and therefore **may be subject to catastrophic data loss!** It is designed for input and output files of running jobs, not persistent storage of data and software.

When dealing with $SHARED_SCRATCH always copy your data in. A "cp" will update the access time on files whereas a move "mv" will preserve the access time. This is important as our periodic cleaning mechanism may purge files where the access time is maintained via the "mv" command.

Avoid I/O over NFS
$HOME and $PROJECTS should **not** be used for job I/O. Jobs found to be using $HOME and $PROJECTS for job I/O are subject to **termination** without notice.

Use Variables Everywhere!
NOTE: The physical paths for the above file systems are subject to change. You should always access the filesystems using environment variables, especially in job scripts.

For information on how to use $PROJECTS, please see our FAQ.

Environment and Shells

The default shell on all the CRC clusters is bash. Other popular shells are available. To have your account's default shell changed from bash to one of these, please file a help request and specify the cluster, username, and desired shell in the ticket. Once your shell is changed this is reflective on all clusters with which you have access. Any active login sessions when your shell is changed will need to be terminated to effect change.

Due to the nature of high performance applications and the batch scheduling system used on CRC clusters, managing your shell environment variables properly is vital.

Customizing Your Environment With the Module Command

It is best practice to only call the *module* command on login nodes before launching jobs. Environments variables are set via the *module* command and can be inherited by your job via a batch script so the environment variables will be visible on the compute nodes. For assistance with module, type ***man module***.

To allow SLURM to inherit environment variables you will need to use the following respective statements within your batch scripts. Please see SLURM documentation for more clarification.

#SBATCH --export=ALL

Each user can customize their environment using the *module* command. This command lets you select software and will source the appropriate paths and libraries. All the requested user applications are located under the /opt/apps directory.

The examples here are taken from the NOTS cluster, but we use the module package on all our clusters in the same manner.

To list what applications are available, use the *avail* sub command:

```
$ module avail
--------------------------- /opt/apps/modulefiles ----------------------------
R/2.13.1          lammps/14Aug11    openmpi/1.4.3-gcc xlc/11.1
amber/11          libhugetlbfs      openmpi/1.4.3-ibm xlf/13.1
at/3.0            meme/4.6.1        papi/4.1.2.1
fftw/3.2.2        namd/2.8b2        python3/3.2.1
ibm               opencl/0.3        vmd/1.9
```

To see a description of each package, use the *whatis* sub command:

```
$ module whatis
at/3.0                : Advance Toolchain is a set of GNU utilites optimized for POWER7
fftw/3.2.2            : Fastest Fourier Transformation in the West
ibm                   : IBM Optimizing Compilers
lammps/14Aug11        : LAMMPS Molecular Dynamics Simulator
libhugetlbfs          : A library that provides easy access to huge (16MB) pages
meme/4.6.1            : MEME Tool for discovering motifs in a group of DNA or protein
sequences
namd/2.8b2            : NAMD molecular dynamics package
opencl/0.3            : IBM OpenCL SDK
openmpi/1.4.3-gcc     : Message Passing Interface
openmpi/1.4.3-ibm     : Message Passing Interface compiled natively with IBM Compilers
vmd/1.9               : Visual Molecular Dynamics package
xlc/11.1              : IBM C and C++ Optimizing Compilers
xlf/13.1              : IBM Fortran77 and Fortran90 Optimizing Compilers
```

To load the module for the IBM compilers, for example, use the *load* sub command:

```
$ module load ibm
```

To see a list of modules that you have loaded, use this command:

```
$ module list
```

To unload all of your modules, use this command:

```
$ module purge
```

The Job Scheduler

The batch job scheduling system implemented on this system uses SLURM. SLURM is responsible for resource management, job scheduling, and monitoring.

Fairshare Scheduling Policy

We implement the SLURM Fairshare feature to provide a fair utilization of the available resources. This is accomplished by allowing historical resource utilization information to be incorporated into job feasibility and priority decisions. This is normally the most significant component of a job's priority, which ultimately defines the position of the job on a queue. We do not use a FIFO (First-In-First-Out) scheduler. Your jobs' priority will be determined by your utilization over the past seven days (sliding window), with high utilization resulting in lower priority for new jobs.

Backfill Scheduling Policy

This is a scheduling optimization which allows SLURM to make better use of available resources by running jobs out of order. Using job data such as walltime and resources requested, the scheduler can start other, lower-priority jobs so long as they do not delay the highest priority jobs. Because of the way it works, essentially filling in holes in node space, backfill tends to favor smaller and shorter running jobs more than larger and longer running ones.

Accurate Walltime Improves Scheduling of Jobs
It is important to specify an accurate walltime for your job in your SLURM submission script. Selecting the default walltime for jobs that are known to run for less time may result in the job being delayed by the scheduler due to an overestimation of the time the job needs to

run.

Automatic Queue Routing

Each of our compute resources has a pre-defined default queue. If you submit your job without specifying a queue, your job will be automatically routed to the default queue. Therefore, be aware of which queue you intend for the job to run in and specify this queue in your SLURM batch script.

Available Partitions and System Load

Partition Name	Maximum jobs queuable	Job Limitations	Maximum total cores per job	Maximum walltime
commons	300	>= 1 node	120	24:00:00

The definition of each partition is as follows:

commons - A standard priority partition that can allocate the maximum number of threads per job and currently has a maximum job walltime of 24 hours. This partition is intended for jobs that need to run parallel (MPI) tasks. The total number of threads in this partition is subject to change at any time due to special projects and system maintenance tasks. You must request a minimum of 2 nodes to use this partition. The maximum number of threads that you can request is 2048 per job when the system is under low utilization. When the system is under high utilization, the maximum is 1204 threads.

Determining Partition Status

A good way to obtain the status of all partitions and their current usage is to run the following SLURM command:

```
user@host:~> sinfo

PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
commons*       up 1-00:00:00      4  alloc cn-[0125-0128]
commons*       up 1-00:00:00     85   idle cn-[0130,0161-0244]
interactive    up      30:00      4   idle cn-[0121-0124]
plasmonics     up 28-00:00:0     48   idle
cn-[0131-0141,0143-0145,0147-0160,0251-0270]
guscus         up 1-00:00:00     39   idle cn-[0041-0079]
earthsci       up 28-00:00:0     15   idle cn-[0246-0250,0271-0280]
cism           up 28-00:00:0      4   idle cn-[0081-0084]
oi1            up 7-00:00:00     16   idle cn-[0085-0100]
scavenge       up 7-00:00:00    122   idle
cn-[0041-0079,0081-0100,0131-0141,0143-0145,0147-0160,0246-0280]
```

Here is a brief description of the relevant fields:

PARTITION: Name of a partition. Node that the suffix "*" identifies the default partition.
AVAIL: Partition state: up or down.
TIMELIMIT: Maximum time limit for an user job in days-hours:minutes:seconds.
NODES: Count of nodes with this particular configuration by node state in the form "[A]vailable/[I]dle/[O]ther/[T]otal
STATE: State of the nodes.
NODELIST: Names of nodes associated with this configuration/partition.

See the manpage for sinfo for more information

```
man sinfo
```

Submitting Jobs with SLURM

Once you have an executable program and are ready to run it on the compute nodes, you **must** create a job script that performs the following functions:

- Use job batch options to request the resources that will be needed (i.e. number of processors, run time, etc.), and
- Use commands to prepare for execution of the executable (i.e. cd to working directory, source shell environment files, copy input data to a scratch location, copy needed output off of scratch location, clean up scratch files, etc).

After the job script has been constructed you must submit it to the job scheduler for execution. The remainder of this section will describe the anatomy of a job script and how to submit and monitor jobs.

⚠ Please note script options are being provided using the long options and not the short options for readability and consistency e.g. *--nodes* versus *-N*.

Per Cluster restrictions
Warning per cluster restrictions may require you to customize the following generic instructions. e.g. NOTS has a maximum of 1 node per job, but DAVinCI does not that restriction. Please refer to the introduction of each cluster for requirements.

SLURM Batch Script Options

All jobs must be submitted via a SLURM batch script or invoking *sbatch* at the command line . See the table below for SLURM submission options.

Option	Description
#SBATCH --account=AccountName **#SBATCH --partition=PartitionName**	Required: You need to specify both the name of the account and partition to schedule jobs especially to use a condo on the cluster. Use the command ***sacctmgr show assoc user=netID*** to show which accounts and partitions with which you have access.
#SBATCH --job-name=YourJobName	Required: Assigns a job name. The default is the name of SLURM job script.
#SBATCH --ntasks=2	Recommended: The number of tasks per job. Usually used for MPI jobs. You can get further explanation here .
#SBATCH --nodes=2	Recommended: The number of nodes requested. You can get further explanation here .
#SBATCH --ntasks-per-node=2	Recommended: The number of tasks per node. Usually used in combination with --nodes for MPI jobs. You can get further explanation here .
#SBATCH --cpus-per-task=4	Recommended: The number processes per task. Usually used for OpenMP or multi-threaded jobs.

#SBATCH --partition=PartitionName	Recommended: Specify the name of the Partition (queue) to use. Use this to specify the default partition or a special partition i.e. non-condo partiton with which you have access.
#SBATCH --time=08:00:00	Recommended: The maximum run time needed for this job to run, in days-hh:mm:ss. If not specified, the default run time will be chosen automatically.
#SBATCH --mem-per-cpu=1024M	Optional: The maximum amount of physical memory used by any single process of the job ([M]ega|[G]iga|[T]era)Bytes. ⓘ The value of "mem-per-cpu" multiplied by "tasks" (mem X tasks) should not exceed the amount of memory on a node. See our FAQ for more details.
#SBATCH --export=ALL	Required: Exports all environment variables to the job. See our FAQ for details.
#SBATCH --mail-user=YourEmailAddress	Recommended: Email address for job status messages.
#SBATCH --mail-type=ALL	Recommended: SLURM will notify the user via email when the job reaches the following states BEGIN, END, FAIL or REQUEUE.
#SBATCH --nodes=1 --exclusive	Optional: Using **both** of these options will give your job exclusive access to a node such that no other jobs can share the node. This combination of arguments will assign eight tasks to your job and will give it exclusive access to all of the resources (i.e. memory) of the entire node without interference from other jobs. Please see our FAQ for more details on exclusive access.
#SBATCH --output=mypath	Optional: The full path for the standard output (stdout) and standard error (stderr) "slurm-%j.out" file, where the "%j" is replaced by the job ID. Current working directory is the default.
#SBATCH --error=mypath	Optional: The full path for the standard error (stderr) "slurm-%j.out" files. Use this only when you want to separate (stderr) from (stdout). Current working directory is the default.

Serial Job Script

A job script may consist of SLURM directives, comments and executable statements. A SLURM directive provides a way of specifying job attributes in addition to the command line options. For example, we could create a *myjob.slurm* script this way:

myjob.slurm

```
#!/bin/bash
#SBATCH --job-name=YourJobNameHere
#SBATCH --account=commons
#SBATCH --partition=commons
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=1000m
#SBATCH --time=00:30:00
#SBATCH --mail-user=YourEmailAddressHere
#SBATCH --mail-type=ALL

echo "My job ran on:"
echo $SLURM_NODELIST
if [[ -d $SHARED_SCRATCH/$USER && -w $SHARED_SCRATCH/$USER ]]
then
 cd $SHARED_SCRATCH/$USER
 srun /path/to/myprogram
fi
```

This example script will submit a job to the default partition using 1 processor and 1GB of memory per processor, with a maximum run time of 30 minutes.

Definition of --ntasks-per-node
For the clusters the **--ntasks-per-node** option means *tasks per node.*

Accurate run time value is strongly recommended
It is important to specify an accurate run time for your job in your SLURM submission script. Selecting eight hours for jobs that are known to run for much less time may result in the job being delayed by the scheduler due to an overestimation of the time the job needs to run.

How to specify mem
The *--mem* value represents memory per processor core. If your *--mem* value multiplied by the number of tasks (*--ntasks-per-node*) exceeds the amount of memory per node, your job will not run. If your job is going to use the entire node, then you should use the *--exclusive* option instead of the *--mem* or *--ntasks-per-node* options (See Here). It is good practice to specify the *--mem* option if you are going to be using less than an entire node and thus sharing the node with other jobs.

If you need to debug your program and want to run in interactive mode, the same request above could be constructed like this (via the *srun* command):

```
srun --pty --partition=interactive --ntasks=1 --mem=1G --time=00:30:00 $SHELL
```

For more details on interactive jobs, please see our FAQ on this topic.

SLURM Environment Variables in Job Scripts

When you submit a job, it will inherit several environment variables that are automatically set by SLURM. These environment variables can be useful in your job submission scripts as seen in the examples above. A summary of the most important variables are presented in the table below.

Variable Name	Description

$SHARED_SCRATCH	Location of shared scratch space. See our FAQ for more details.
$LOCAL_SCRATCH	Location of local scratch space on each node.
$SLURM_JOB_NODELIST	Environment variable containing a list of all nodes assigned to the job.
$SLURM_SUBMIT_DIR	Path from where the job was submitted.

Job Launcher (srun)

For jobs that need two or more processors and **are compiled with MPI libraries**, you must use *srun* to launch your job. The job launcher's purpose is to spawn copies of your executable across the resources allocated to your job. We currently support *srun* for this task and do not support the *mpirun* or *mpiexec* launchers. By default *srun* only needs your executable, the rest of the information will be extracted from SLURM.

The following is an example of how to use *srun* inside your SLURM batch script. This example will run *myMPIprogram* as a parallel MPI code on all of the processors allocated to your job by SLURM:

myMPIjob.slurm

```
#!/bin/bash
#SBATCH --job-name=YourJobNameHere
#SBATCH --account=commons
#SBATCH --partition=commons
#SBATCH --ntasks=24
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:30:00
#SBATCH --mail-user=YourEmailAddressHere
#SBATCH --mail-type=ALL

echo "My job ran on:"
cat $SLURM_NODELIST
if [[ -d $SHARED_SCRATCH/$USER && -w $SHARED_SCRATCH/$USER ]]
then
 cd $SHARED_SCRATCH/$USER
 srun /path/to/myMPIprogram
fi
```

This example script will submit a job to the default partition using 24 processor cores and 1GB of memory per processor core, with a maximum run time of 30 minutes.

⚠ **Your Program must use MPI**
The above example assumes that *myMPIprogram* is a program designed to be parallel (using MPI). If your program has not been parallelized then running on more than one processor will not improve performance and will result in wasted processor time and could result in multiple copies of your program being executed.

The following example will run *myMPIprogram* on only four processors even if your batch script requested more than four.

```
srun -n 4 /path/to/myMPIprogram
```

To ensure that your job will be able to access an mpi runtime, you must load an mpi module before submitting your job as follows:

```
module load GCC OpenMPI
```

Submitting and Monitoring Jobs

Once your job script is ready, use *sbatch* to submit it as follows:

```
sbatch /path/to/myjob.slurm
```

This will return a jobID number while the output and error stream of the job will be saved to one file inside the directory where the job was submitted, unless you specified otherwise.

The status of the job can be obtained using SLURM commands. See the table below for a list of commands:

Command	Description
squeue	Show a detailed list of all submitted jobs.
squeue *-j jobID*	Show a detailed description of the job given by *jobID*.
squeue *-- start -j jobID*	Gives an estimate of the expected start time of the job given by *jobID*.

There are variations to these commands that can also be useful. They are described below:

Command	Description
squeue -l	Show a list of all running jobs.
squeue *-u username*	Show a list of all jobs in queue owned by the user specified by *username*.
scontrol *show job jobID*	To get a verbose description of the job given by *jobID*. The output can be used as a template when you are attempting to modify a job.

There are many different states that a job can be after submission: *BOOT_FAIL (BF), CANCELLED (CA), COMPLETED (CD), CONFIGURING (CF), COMPLETING (CG), FAILED (F), NODE_FAIL (NF), PENDING (PD), PREEMPTED (PR), RUNNING (R), SUSPENDED (S), TIMEOUT (TO),* or *SPECIAL_EXIT (SE)*. The *squeue* command with no arguments will list all jobs in their current state. The most common states are described below.

Running (R): These are jobs that are running.

Pending (PD): These jobs are eligible to run but there is simply not enough resources to allocate to them at this time.

Deleting Jobs

A job can be deleted by using the *scancel* command as follows:

```
scancel jobID
```

Compiling and Optimizing

Several programming models are supported on PowerOmics. Programs that are of sequential and parallel can be submitted on this system. Sequential programs require one processor thread to run. Parallel programs utilize multiple processor threads concurrently. Message passing and

threaded applications generally fit under the scope of parallel computing.

The supported compilers on PowerOmics are the IBM compilers (*XLC* and *XLF*) and the GNU compiler (GCC) enabled with Advance Toolchain c apabilities..OpenMPI implementations built with the IBM and GCC compilers are available and can be loaded upon demand using the *module* co mmand.

Compiling Serial Code

To compile serial code you will have to load the appropriate compiler environment. The IBM C/C++ and Fortran environments, respectively, are loaded as follows:

```
module load XLC
module load XLF
```

Once the environment is set, you can view the appropriate online documentation for the compiler:

```
man xlc

man xlf
```

Compiling Parallel Code

To compile a parallel version of your code that has OpenMPI library calls, use the appropriate OpemMPI library. Again, use the module command to load the appropriate compiler environment as follows (IBM versions highly recommended):

module command	Description
module load openmpi/1.4.3-gcc	For gcc compiled version.
module load openmpi/1.4.3-ibm	For IBM compiled version.

To compile your code you will have use the OpenMPI scripts that are currently in your default path. The OpenMPI scripts are responsible for invoking the compiler, linking your program with the OpenMPI library and setting the OpenMPI include files.

Once the environment is set, you can compile your program with one of the following (assuming the IBM compiler as above):

```
mpicc -o executablename mpi_sourcecode.c

mpicxx -o executablename mpi_sourcecode.cc

mpif77 -o executablename mpi_sourcecode.f77

mpif90 -o executablename mpi_sourcecode.f90
```

When invoked as described above, the compiler will perform the preprocessing, compilation, assembly and linking stages in a single step. The output file (or executable) is specified by *executablename* and the source code file is specificed by *mpi_sourcecode.f77*, for example. Omitting the *-o executablename* option will result in the executable being named *a.out* by default. For additional instructions and advanced options please view the online manual pages for each compiler (i.e. execute the command *man mpif77*).

GNU Compiler and Advance Toolchain

The GNU compiler is installed as part of the Ubuntu Linux distribution. Use *man gcc* to view the online manual for the C and C++ compiler, and *m an gfortran* to view the online manual for the Fortran compiler.

In addition to the Ubuntu basic version of GCC, there is an Advance Toolchain stack installed on PO that includes special optimization for POWER8 hardware. Advance Toolchain versions of GCC can be accessed by using the *module load at* command.

Getting Help

Request Help with the Center for Research Computing Resources