ACRC: BlueCrystal User Guide

Table of Contents

Introduction

BlueCrystal is the University of Bristol's high performance computing (HPC) facility. The purpose of this user guide is two-fold: The first aim is to help you start using BlueCrystal as quickly as possible. The second is to help you get the most out of the available resources. Following this second strand of advice will help you run your jobs faster and so boost your productivity.

If you have any questions about the use of BlueCrystal which are not covered in this guide, or would like more advice regarding the use of high performance computing in your work, please get in contact with the ACRC team. For general enquiries, you can email hpc-help@bristol.ac.uk. If you would like to speak to Caroline Gardener, the ACRC's academic research facilitator, her email address and phone number are Caroline.Gardiner@bristol.ac.uk, 0117 331 4375 (internal 14375).

How to apply for an account

BlueCrystal can be used by all members of the University and is free at the point of use. To apply for an account, visit: https://www.acrc.bris.ac.uk/login-area/apply.cgi.

BlueCrystal has existed since 2005 and three separate compute clusters have been procured to date. These are known as phases 1, 2 and 3 respectively. Phase 3 is our newest cluster, installed in 2013. Phase 1 will be retired in April 2015. As a short hand in the text, I will refer to, e.g. BlueCrystal phase 3 as BCp3.

How to login

Once you have a user-name and password for one of the phases, the first thing you will want to do is to connect to one of the login nodes of the cluster. You can do this using the SSH (Secure Shell) protocol. SSH tools exist for nearly all operating systems, including Microsoft Windows, Apple's OSX and Linux.

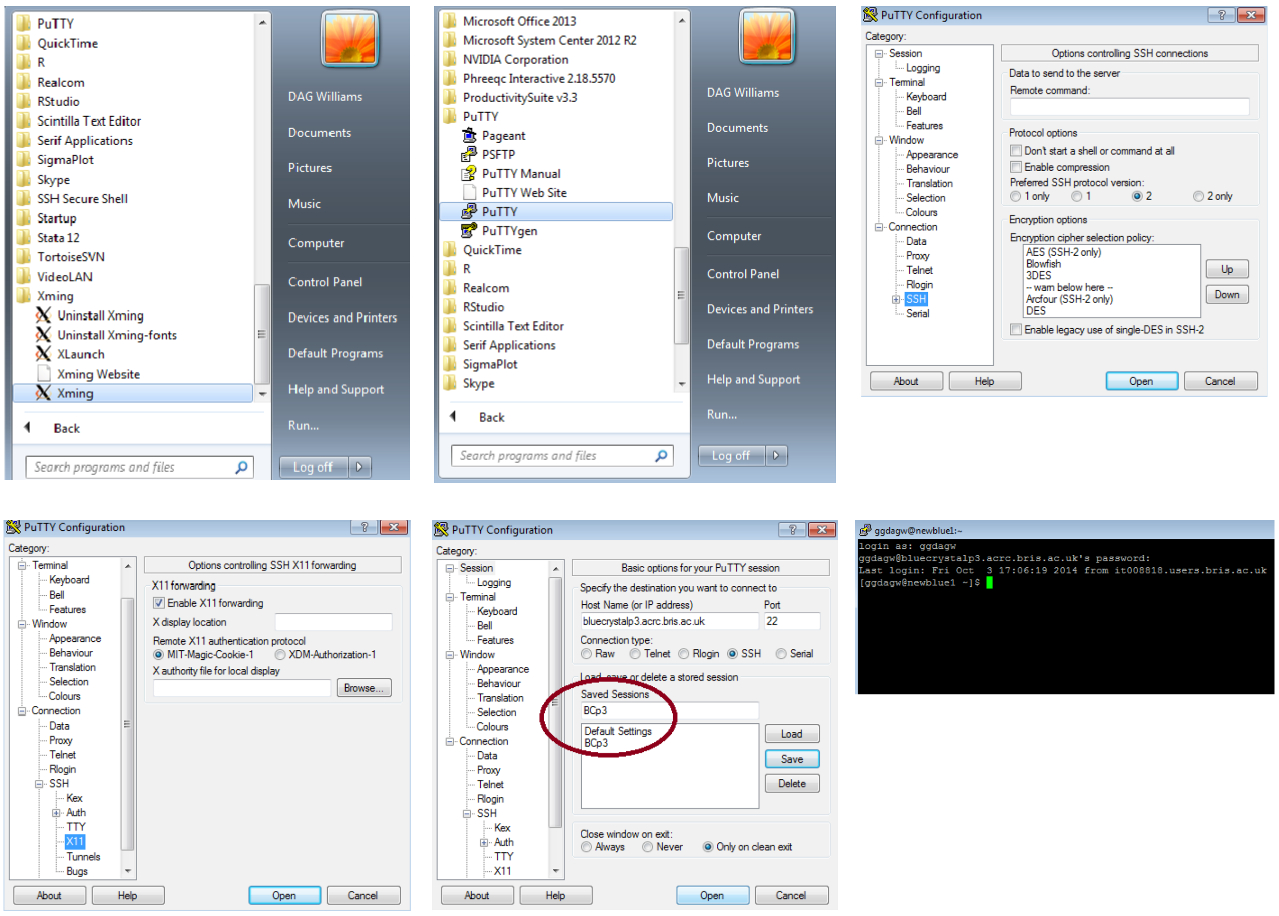
From Windows

Figure 1 is a gallery of screen-shots collected when connecting to BlueCrystal from a Windows PC. First start the xming application, located under the All Programs menu. This is an X server, which will enable you to view windows on your PC which were spawned from your session on the cluster. An example of such a window might be a text editor with a graphical user interface (GUI) such as gedit.

Next start the putty application, also located the All Programs menu. Under the putty configurations, you must enable X11 forwarding to make use of xming.

Click on Session in the left-hand pane and you can enter the details of the machine that you want to connect to. In this example, I am connecting to BCp3 (bluecrystalp3.acrc.bris.ac.uk).

You can save your configurations for re-use when you next login (highlighted using a red circle). Once you have entered your user-name and password, you are rewarded with a Linux command line prompt on one of the cluster's login nodes. If you are new to the Linux operating system, you will want pay close attention to the section on the topic below.



Options controlling SSH X11 forwarding
X11 forwarding
[✓ Enable X11 forwarding]

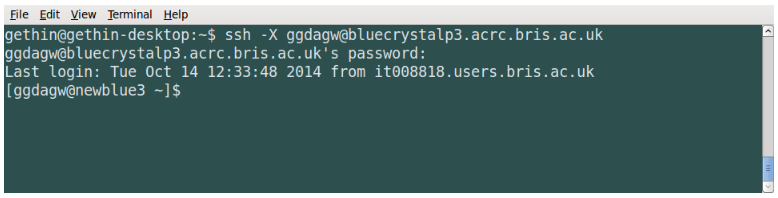
Basic options for your PuTTY session
Host Name (or IP address): bluecrystalp3.acrc.bris.ac.uk
Port: 22
Connection type: [✓ SSH]
Saved Sessions: BCp3

```
login as: ggdagw
ggdagw@bluecrystalp3.acrc.bris.ac.uk's password:
Last login: Fri Oct  3 17:06:19 2014 from it008818.users.bris.ac.uk
[ggdagw@newblue1 ~]$
```

Fig1: Connecting to BlueCrystal from Windows

From an OSX or Linux terminal

For those connecting from an Apple or Linux computer, you will login using the command line from a terminal. An example of using SSH from the command line is shown in figure 2. The -X option ensures X11 forwarding.



```
gethin@gethin-desktop:~$ ssh -X ggdagw@bluecrystalp3.acrc.bris.ac.uk
ggdagw@bluecrystalp3.acrc.bris.ac.uk's password:
Last login: Tue Oct 14 12:33:48 2014 from it008818.users.bris.ac.uk
[ggdagw@newblue3 ~]$
```

Fig2: Connecting to BlueCrystal from a terminal

How to copy files to and from the cluster

After you have logged in, it is likely that you will want to transfer some files to the cluster. These could be source code, scripts or data. After you have run a job on BlueCrystal it is likely that you will want to move the results off the cluster to another computer for further analysis or longer term storage. At this point it should be noted that all of the user storage on BlueCrystal is scratch-space. As such it is never backed-up and any data stored on the cluster must be considered as 'at risk'.

If you have a need to store research data securely, you may want to consider using the University's Research Data Storage Facility (RDSF). For more information see:

- https://www.acrc.bris.ac.uk/acrc/storage.htm.

The remainder of this section gives examples of file transfer using the winSCP tool on Windows and the terminal for OSX and Linux.

On Windows

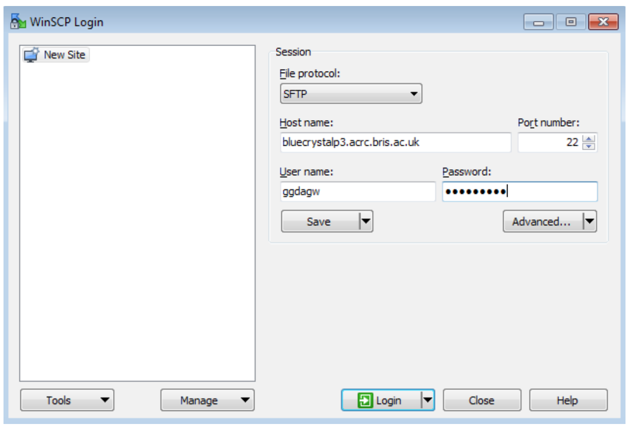
Figure 3 is a gallery of screen-shots taken when using winSCP. The login procedure is straightforward. Again, my example uses bluecrystalp3.acrc.bris.ac.uk as the host name. As before, you can save the details that you entered for convenient re-use another time. Once connected, you will be presented with two panes: One for the local machine (left-hand pane) and one for the remote machine (right-hand pane). In this example, BlueCrystal is the remote machine. You can drag and drop files and directories from one pane to the other. Note that you can access MyFiles (mapped to the O: drive letter) from the pull down menu, highlighted in the third image.

If you would like to quickly edit a file, you can double click on it (on either the local or remote machine) and you will be presented with a text editor—see the last image in the series. You should be aware of line ending differences between, e.g. Windows and Linux. These differences can sometimes cause trouble. See, e.g. http://en.wikipedia.org/wiki/Newline, to learn more on this topic.

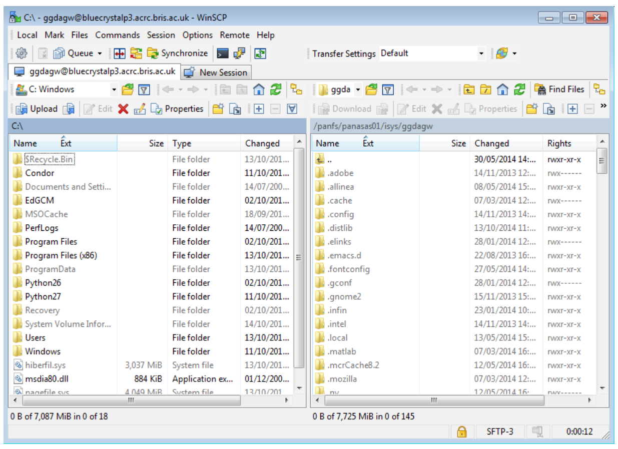
For more information about winSCP, a useful destination is:

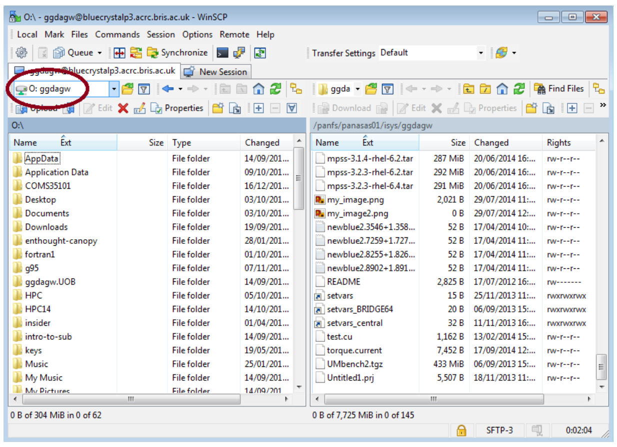
- http://en.wikipedia.org/wiki/WinSCP.

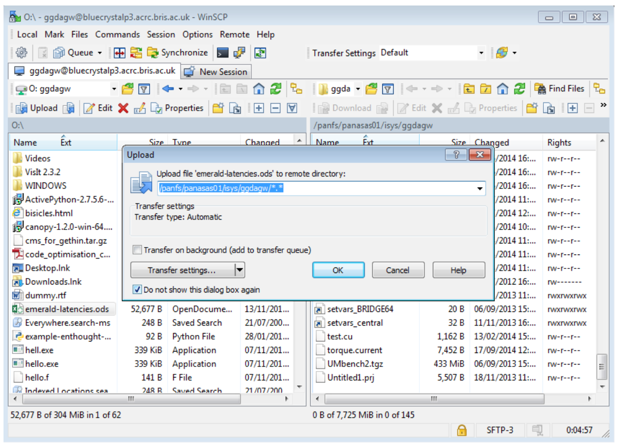
(The page also contains a link to a comparison of other file management tools.)


WinSCP Login

Session
File protocol: SFTP
Host name: bluecrystalp3.acrc.bris.ac.uk
Port number: 22
User name: ggdagw
Password: ●●●●●●●●●


C:\ - ggdagw@bluecrystalp3.acrc.bris.ac.uk - WinSCP


O:\ - ggdagw@bluecrystalp3.acrc.bris.ac.uk - WinSCP

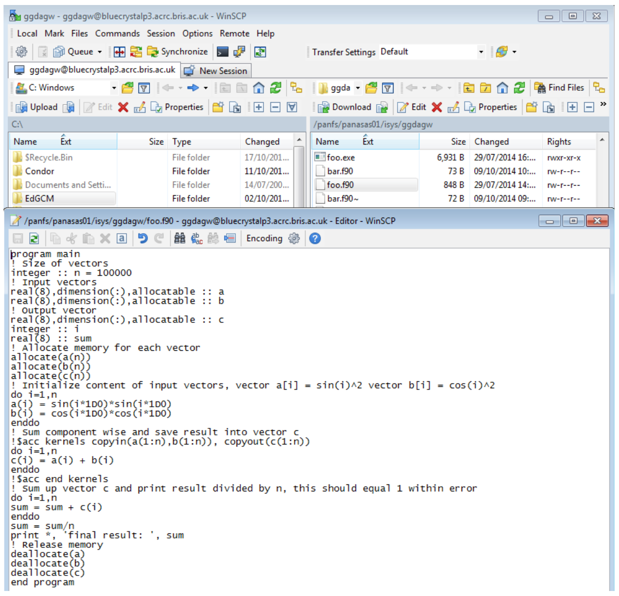

Upload

Upload file 'emerald-latencies.ods' to remote directory:
/panfs/panasas01/isys/ggdagw/*.*

Transfer settings
Transfer type: Automatic

☐ Transfer on background (add to transfer queue)

☑ Do not show this dialog box again


/panfs/panasas01/isys/ggdagw/foo.f90 - ggdagw@bluecrystalp3.acrc.bris.ac.uk - Editor - WinSCP

```
program main
! Size of vectors
integer :: n = 100000
! Input vectors
real(8),dimension(:),allocatable :: a
real(8),dimension(:),allocatable :: b
! Output vector
real(8),dimension(:),allocatable :: c
integer :: i
real(8) :: sum
! Allocate memory for each vector
allocate(a(n))
allocate(b(n))
allocate(c(n))
! Initialize content of input vectors, vector a[i] = sin(i)^2 vector b[i] = cos(i)^2
do i=1,n
a(i) = sin(i*1D0)*sin(i*1D0)
b(i) = cos(i*1D0)*cos(i*1D0)
enddo
! Sum component wise and save result into vector c
!$acc kernels copyin(a(1:n),b(1:n)), copyout(c(1:n))
do i=1,n
c(i) = a(i) + b(i)
enddo
!$acc end kernels
! Sum up vector c and print result divided by n, this should equal 1 within error
do i=1,n
sum = sum + c(i)
enddo
sum = sum/n
print *, 'final result: ', sum
! Release memory
deallocate(a)
deallocate(b)
deallocate(c)
end program
```

Fig3: Copying files to/from BlueCrystal on Windows

Using an OSX or Linux terminal

You can use the scp command to transfer files when using an OSX or Linux terminal. The basic syntax is:

- `scp <local-file> <remote-file>`

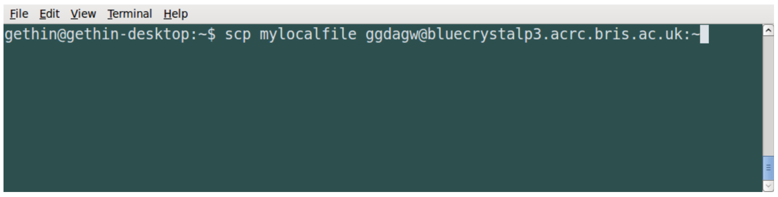
See figure 4 for an example. The remote file name shown has three sections:

1. The remote user name.
2. The remote machine name, and
3. The remote file name, prefixed with a ':'.

I've used ~ to indicate my home directory on BlueCrystal. In this case, the local file will be copied to my home directory on the cluster. You can given pathnames, instead of ~ and also the name of the file to be created on the remote file system. For example, I might type:

- `scp mylocalfile ggdagw@bluecrystalp3.acrc.bris.ac.uk:~/tmp/foo.txt`

to copy a local file called 'mylocalfile' to a file called 'foo.txt' in the tmp sub-directory of my home directory on the cluster.



```
gethin@gethin-desktop:~$ scp mylocalfile ggdagw@bluecrystalp3.acrc.bris.ac.uk:~
```

Fig4: Copying files to/from BlueCrystal using a terminal

All users have a file-store quota, and so there is a limit on how much data you can store on the clusters. The file-store is a resource shared among all the users. The purpose of the quota is to prevent the file-store from over filling, which would render the cluster unusable. To view your file-store usage with reference to your quota type:

- `showquota`: on BlueCrystal phases 1 & 2.
- `pan_quota`: on BlueCrystal phase 3.
  (Note that usage and quota in this case are given in bytes, which is not the most convenient of units, as you will typically be presented with some very large numbers.)

If you have space on the RDSF, you can arrange for it to be accessed via the login nodes of one of the clusters. The RDSF project must have NFS support enabled. Once access is requested the files will be available under the /projects/<rdsf-project-name> directory.

File-system tips

Each phase of BlueCrystal employs a high-performance, parallel file-system to cope with the stringent demands made by a high-performance compute cluster. Each running process can read from- and write to files located on this shared resource. If the file-system is unable to serve these requests (potentially a great many) in a timely manner, processes will be forced to wait—slowing their progress.

BlueCrystal uses the best parallel file-systems available, but even these can be taxed by certain workloads. **The key message of this section is that your job will run faster if it uses a few large files, rather than many small ones**. The reason for this is that it takes time for a file to be located on the file-system. If your program accesses many small files, the time spent seeking for these files will grow and may perhaps become the major fraction of the run-time. It is very likely that your program will be stalled when attempting to access a file and it will be unable to perform any useful computation until that file is found and successfully opened.

The fastest running programs will make best use of the available volatile memory (RAM) to store intermediate results and will visit the file-system as infrequently as possible. However, you must also be vigilant not to exceed the limit of your available RAM, as this will lead to a significant slow down of your running program too.

How to use the Linux operating system

In common with many high-performance compute clusters, BlueCrystal uses the Linux operating system. Rather than include a full Linux tutorial in this section, I will refer you to other material. In the context of using BlueCrystal, Linux is a command line driven operating system and as such may seem rather alien to those used to Windowing environments. Learning the full capability of the Linux command line is a many-year project. However, rest assured that you can quickly assimilate the basics, sufficient to begin using the BlueCrystal.

There are many good Linux courses on the web. For example here is the URL for one hosted by the University of Surrey:

- http://www.ee.surrey.ac.uk/Teaching/Unix/

The ACRC resources page also contains a handy quick reference card:

- https://www.acrc.bris.ac.uk/acrc/resources.htm

How to customise your environment

The Environment Modules package provides BlueCrystal users with a powerful and convenient way to customise your environment. Access to the software available on the cluster is provided through modules. To use your choice of compiler or application you must first load the appropriate module. Most of time, you will need only four commands:

- `module avail`: lists all the modules available on the cluster.
- `module list`: lists all the modules that you currently have loaded.
- `module add <modulename>`: will load the specified module, and
- `module del <modulename>`: will delete it.

You can include module commands in shell start-up files, such as your .bashrc, to ensure that your desired environment is automatically created each time you login. Loading modules in this way is particularly recommended since the environment for jobs running on the compute nodes of the cluster will be set by your shell start-up files.

As an example, if you wanted to compile and run MPI code using the Intel compiler on BCp2, you could include the following in your shell start-up file:

```
module add ofed/qlogic/intel/64/1.5.1
```

Modules can load other modules, and in this case, the above command also load the languages/Intel-Composer-12 module.

To compile and run MPI programs using Intel on BCp3, you could use:

```
module add openmpi/intel/64/1.6.5
module add intel-cluster-studio/compiler/64/13.1/117
```

Be sure not to load multiple MPI modules, at the same time, as this can lead to problems.

As further examples, to use R, Matlab, Python and Subversion on BCp3, your shell start-up file could include:

```
module add languages/R-3.0.2
module add apps/matlab-r2013b
module add languages/python-3.3.2
module add tools/subversion-1.8.4
```

Note that in all these cases, the version of the tool to be loaded is included in the module name.

To learn more about what any particular module does you can use the following commands:

- `module whatis <modulename>`: gives a concise summary for the module.
- `module help <modulename>`: can give more detail on the purpose of the module.
- `module show <modulename>`: lists how the module file will alter your environment.

More information on customising your environment is provided by a 'How-To' document on the ACRC resources page:

- https://www.acrc.bris.ac.uk/acrc/resources.htm.

If you would like to change your password on any of the clusters, type, `passwd`.

An overview of the clusters components

When reading the subsequent sections, it is useful to have an overview of the physical make-up of a cluster. From your local machine, you connect to one of several login nodes. Connections are distributed among this set on a round-robin basis. In this way, connection traffic is equally shared out.

When you submit a job—more on the mechanics of that in the next section—it is assigned to one or more of the compute nodes. There are several categories of compute nodes. The vast majority of nodes in the cluster are standard compute nodes, consisting of some x86 processors, a quantity of RAM, a local disk and little else. The large memory nodes are similar to the standard nodes, except that they are equipped with more RAM. Some other nodes in the cluster may be equipped with GPU cards or other acceleration hardware.

All the nodes of the cluster are connected together using a high-performance network. This network allows the nodes to exchange data with low latency and high bandwidth. All phases of BlueCrystal use an InfiniBand for this purpose. For distributed memory workloads, this network allows the nodes of the cluster to co-operate on a task to good effect.

The login nodes are connected to the University campus network and are fire-walled from the internet beyond.

Technical specifications – BCp3

- 223 standard compute nodes each with 2 x 2.6GHz 8-core Intel E5-2670 (SandyBridge) chips (a total of 16 cores), 4GB of RAM per core (64GB total) and 880GB of disk space on /local.
- There are an additional 100 identical nodes that can host dual GPGPUs. At date of writing, 38 of these nodes are equipped with 2 NIVIDIA K20 cards each. Two nodes have a pair of Intel Xeon Phi co-processors each and one node contains a pair of AMD S1950 GPUs.
- 18 large memory nodes, each containing 256GB of memory and 16 processors. Some operate as stand-alone systems and some are configured to emulate a virtual SMP with up to 4TB RAM usable using ScaleMP software.
- Intel True Scale QDR Infiniband fabric.
- Panasas file-store. 344TB capacity, accessible form all compute nodes.

Technical specifications – BCp2

- When commissioned, 416 standard compute nodes each with 2 x 2.8GHz 4-core Intel Harpertown E5462 chips (a total of 8 cores) , 1GB of RAM per core (8GB total) and 193GB of scratch disk space on /local. (Since then the total number of compute nodes has been diminished by hardware failures. At the date of writing BCp2 has a little over 300 active compute nodes.)
- 4 nodes each with 2 x 2.4 GHz 4-core processors, 24GB RAM, 2 Nvidia Fermi M2050 GPU cards and 164GB of scratch space on /local.
- 2 large memory nodes each with 4 x 2.0 GHz 6-core processors, 256GB RAM (10GB per core) and 300GB of scratch space on /local.
- QLogic Infinipath high-speed network.
- IBM's General Parallel File System (GPFS). 88TB capacity, accessible from all compute nodes.

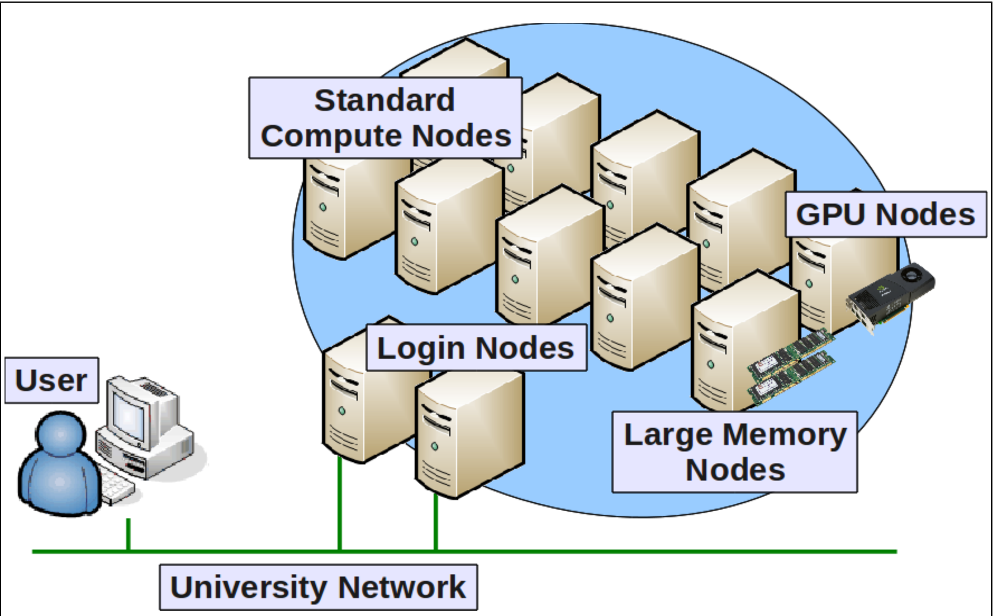


fig5: A schematic of the components of a cluster.

The file-servers are omitted from figure 5, but it is worth remembering that the high-performance, parallel file-system is available to all the nodes, and is a resource shared by all users of the cluster.

How to run a job

Now that we have covered logging-in, transferring files, customising your environment, the operating system and an overview of the physical make-up of the cluster, let's look at how to run jobs.

Queues and resources

The workload of the cluster is mediated through a queuing system. This piece of software performs several highly desirable functions:

- It matches available computing resources to those required to run a particular job.
- It enforces a strict fair-share policy, where all users are given equal access to the compute resources. No monopolisation is possible.
- It balances the work load across all the nodes of the cluster.
- It maintains a queue of jobs to be run on the cluster. Jobs are typically submitted in batch mode, meaning that no human intervention is required when it is time to run the job. Depending on the available resources, your job may run immediately or perhaps when you are away from your computer screen.

The essential features of a queue are shown in figure 6. Each job has; (i) a unique ID; (ii) a record of the user who submitted it; (iii) an associated resource request; and (iv) a priority. As old jobs are completed and new ones submitted, the queue scheduler updates the queue, assigns any new jobs a priority, sorts the queue and finally decides which jobs will be run on which nodes.

The two main resources which you can request for your job are the number of nodes and processors per node and the amount of time needed to run the job. You can also specify whether you require any GPUs and you can specify how much memory your job will need. Examples of how you make these resource requests are given the following sections.

jobID	user	requested resources	state	requested time	priority
234	Jane	16 cores	running	2 hours	1
235	Clare	128 cores	running	48 hours	2
236	John	1 core & 1 GPU	running	6 hours	3
237	Sarah	32 cores, large mem	queued	24 hours	4
238	Clare	128 cores	queued	48 hours	5
...	...	...	...	...	...

fig6: A schematic of a job queue.

A cluster will typically maintain several queues aimed at different kinds of jobs and BlueCrystal is no exception. You can see all the queues available by typing `qstat -q`. On BlueCrystal, these include queues for different length jobs and for access to specialist resources, such as large memory nodes, or those equipped with GPUs.

You can query the state of the queues using the commands qstat or showq. These both take various options to customise their behaviour. A few examples of running these commands which you may find useful are:

- `qstat -u <username>`: to see all the jobs owned by a particular user, typically yourself!
- `qstat <queuename>`: to see all jobs that are currently running which were submitted to a given queue. (Note that jobs waiting to run will not be shown.)
- `showq | less`: lists all the jobs on the cluster, running or waiting, sorted in priority order.
- `qstat -f <jobid>`: gives a lot of detail about the specified job—useful for checking precisely what is happening to a job.

Submission scripts

In order to submit a job to the queue, you must first write a submission script. This script is then given as an argument to the qsub command:

```
qsub your_submission_script
```

The submission script is perhaps most conveniently written as a shell script, but it could also be written in perl or python, for example.

You are not restricted in what you can put into a submission script. However, there are a few key elements that are typically included. For example, you can include special comment lines that are understood by the queuing system. These comment lines are often used to request the resources required for your job. qsub will accept various options to achieve the same effect. However, it is often more convenient to include the equivalents of these options in the script itself. (Command line options will override any equivalent requests contained in the submission script.)

I have grouped submission script examples for different types of job into this section. It is worth reading through all the examples in turn, as the explanation of what submission scripts do is spread throughout the section.

Below is a small example of a submission script which we could use for running a program that uses only one processor. We would typically refer to such a program as a serial, or single-threaded.

```
#!/bin/bash
# request resources:
#PBS -l nodes=1:ppn=1
#PBS -l walltime=02:00:00
# on compute node, change directory to 'submission directory':
cd $PBS_O_WORKDIR
# run your program, timing it for good measure:
time ./my-serial-program
```

Let's look at each line of this script in turn, so that we can get a good appreciation for what it is doing:

- '#!/bin/bash' tells us that this is a shell script and that all subsequent lines will be parsed by the BASH interpreter.
- '#PBS -l nodes=1:ppn=1' is the first of the #PBS directives which are parsed by the queuing system. This line is requesting the number of processors needed by the job—a single processor (resident in a single compute node).
- '#PBS -l walltime=02:00:00' The second of the #PBS directives completes our resource request by specifying how long we will need the processor for. In this case, two hours.
- 'cd $PBS_O_WORKDIR' When your job has been successfully queued and scheduled to run, it will be executed on an allocated compute node. An SSH connection will be made from the relevant login node to the allocated compute node. Although this is a batch operation, requiring no direct user actions, the connection shared characteristics of an interactive login. Notably the connection will be initiated in your home directory. In order to run the rest of the script in the directory from which the job was submitted—let's call this the 'submission directory'—a cd command must be issued. The environment variable PBS_O_WORKDIR is set the relevant directory, courtesy of the queuing system, and so we can use the line above to always move to the correct directory.
- 'time ./my-serial-program' Finally we run the program that we are interested in. The time command will tell you how long the program took to run, which can be useful for tuning the walltime aspect of your resource request.

The output of 'qstat -q' tells you a number of useful details about the queuing system configuration. First, it tells you the names of all the queues available on the cluster. It also tells you how many jobs are waiting and running, and the maximum walltime allowed for a job under those queues. Some of the queues listed will be dedicated to private nodes. Others will be 'standard' queues available to all. The standard queues are: testq, veryshort, short, medium and long.

If you have specified a walltime in your submission script you can invoke the qsub command passing the name of your submission script as the only argument. In this case the queueing system will select the appropriate 'standard' queue for your job. Alternatively, you can specify your desired queue using an option to qsub (and optionally omit the walltime resource resource):

```
qsub -q short your_submission_script
```

Here, we've chosen to submit the job to the short queue. Use the `qdel` command with the relevant jobid to remove an unwanted job from the queue:

```
qdel <jobid>
```

The shorter the walltime of your job (whether explicitly specified or implicit set by queue membership), the higher the priority of the job. It is therefore prudent to specify the minimum amount of time required for your job. However, do not specify less than the required walltime as any running job found to exceed this parameter will be terminated by the queuing system.

When complete, any output collected from your job will be placed in files in the submission directory. The standard error stream will be collected into a file with the naming convention: <submission-script-name>**.e**<jobid>. Standard out will go to: <submission-script-name>**.o**<jobid>.

This short submission script can be extended in a number of ways. Some possible queuing system directives include:

- #PBS -N <jobname>: to set the name of the job which will appear in the queues.
- #PBS -O <outname>: to modify the first part of the name of the output files.

Email forwarding from the cluster is, in general, not supported.

In terms of your resource request, you can also specify how much physical memory your job will need. For example, adding:

- #PBS -l mem=4gb

will mean that your job will only be allocated to a node that has at least 4GB of RAM free. Note that each standard compute node of BCp3 has 64GB of RAM and 16 cores, so each core nominally has 4GB of RAM associated with it. Standard compute nodes on BCp2 have 8GB of RAM and 8 cores, so in this case each core is nominally associated with 1GB of RAM. If your job requires more than these nominal limits, please see the How-To guide for large memory jobs on the ACRC resources page, https://www.acrc.bris.ac.uk/acrc/resources.htm. Be aware that if you do specify a memory usage in a resource request and your job exceeds the stated amount, the queuing system will kill your job.

Please do not submit a large number of relatively short jobs to the queue. The queuing system is designed to handle traditional HPC workloads, typically comprised of relatively few jobs running for a long period of time, using many processors concurrently. We can contrast this with high throughput workloads where many thousands of jobs may be queued, each running for only a few minutes or seconds. The queuing system will struggle to schedule jobs in a high throughput scenario and can lock-up for all users in extreme cases. Nobody wants this. See the tips section below for more advice on successfully submitting many, short jobs.

The queuing system is configured with a number of limits. These include a maximum number of jobs a user can have in the queue and the maximum number of processors that a user may exploit at any one time. This limits are required to protect the normal operation of the queuing system. If these limits cause you any difficulty, please contact the ACRC team: hpc-help@bristol.ac.uk.

The next example is a template suitable for running multi-threaded jobs:

```
#!/bin/bash
# request resources:
#PBS -l nodes=1:ppn=16
#PBS -l walltime=02:00:00
# on compute node, change directory to 'submission directory':
cd $PBS_O_WORKDIR
# run your program, timing it for good measure:
time ./my-multi-threaded-program
```

The only change here is that we have requested the use of 16 processors, resident in a single compute node of the cluster (nodes=1:ppn=16). The constitutes a complete node of BCp3. Note that this submission script could not be run on a standard compute node of BCp2. This is because BCp2 has only 8 processors is a standard compute node. In this case, the scheduler can never satisfy the resource request and the job will be left in the queue indefinitely. You will not receive a warning regarding the resource mismatch.

If you are running an OpenMP program, there are a number of environment variables which you can also profitably use in your submission script. In the example below, we have explicitly set the number of OpenMP threads to match the resource requested (8 cores, note ppn=8):

```
#!/bin/bash
# request resources:
#PBS -l nodes=1:ppn=8
#PBS -l walltime=02:00:00
# on compute node, change directory to 'submission directory':
cd $PBS_O_WORKDIR
# set the number of OpenMP threads to match the resource request:
export OMP_NUM_THREADS=8
# run your program, timing it for good measure:
time ./my-multi-threaded-program
```

By default most OpenMP runtime environments will spawn as many threads as there are cores in the node. If you have requested only part of a node, however, this will mean that your threads will interfere with someone else's job. So we must control the number of threads spawned so as to match the requested resources.

If you are concerned about NUMA, or memory bandwidth effects, there are other environment variables that you can use, such as:

- 'OMP_PROC_BIND=true' will ensure that threads are not switched between sockets.
- GOMP_GPU_AFFINITY and KMP_AFFINITY can be set to control thread affinity for GNU and Intel compiled programs, respectively.

The next example is for running an MPI distributed memory job:

```
#!/bin/bash
# request resources:
#PBS -l nodes=2:ppn=16
#PBS -l walltime=02:00:00
# on compute node, change directory to 'submission directory':
cd $PBS_O_WORKDIR

# record some potentially useful details about the job:
echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo PBS job ID is $PBS_JOBID
echo This jobs runs on the following machines:
echo `cat $PBS_NODEFILE | uniq`

# count the number of processors available:
numprocs=`wc $PBS_NODEFILE | awk '{print $1}'`
# launch the program using mpiexec:
mpiexec -np $numprocs ./my-mpi-program
```

At the top of the file, you can see that we are now requesting to full nodes (of BCp3). The environment variable PBS_NODEFILE records the number of nodes and cores allocated to the job. We can use it to record which nodes were used to run the job and to also tell mpiexec how many MPI processes to launch. mpiexec is the preferred command for launching an MPI job as it often does a better job of cleaning up processes after the job has completed than mpirun.

However, if you must use mpirun, then here is an example which also creates a machine file, which mpirun needs:

```
#!/bin/bash
# request resources:
#PBS -l nodes=2:ppn=16
#PBS -l walltime=02:00:00
# on compute node, change directory to 'submission directory':
cd $PBS_O_WORKDIR

# record some potentially useful details about the job:
echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo PBS job ID is $PBS_JOBID
echo This jobs runs on the following machines:
echo `cat $PBS_NODEFILE | uniq`

# create a machine file for subsequent use with mpirun:
cat $PBS_NODEFILE > machine.file.$PBS_JOBID
# count the available processors:
numprocs=`wc $PBS_NODEFILE | awk '{print $1}'`
# launch the program using mpirun:
mpirun -np $numprocs -machinefile machine.file.$PBS_JOBID ./my-mpi-program
```

Once you have sucessfully queued a job, other useful commands which use the jobid include:

- `showstart <jobid>`: to get an estimate of when your job will be run.
- `qstat -f <jobid>`: for lots of information about a job.
- `checkjob <jobid>`: (optionally with -v) to give information about a running job.
- `tracejob <jobid>`: to learn more about a completed job.

Submission script tips

As mentioned previously, if you have a large number of short jobs—i.e. you have a high throughput workload—please do not submit them all individually to the queue, as the scheduler will struggle to process them all. Here are some alternatives that you can try instead:

- If you can, group some of your short jobs together and run them in sequence in your submission script. You may find a for loop useful for this task.
- You can also perform several tasks concurrently (assuming you have requested sufficient resource) by running them in the background and then using the wait command to ensure that they have all completed.
- Job arrays can be useful for concisely submitting and managing a number of related jobs. See the relevant How-To document on the ACRC resources page: https://www.acrc.bris.ac.uk/acrc/resources.htm.

Using packages, applications and libraries

The best way to see what software is installed on the cluster is to type module avail.

The ACRC training page, https://www.acrc.bris.ac.uk/acrc/training.htm, contains links a training materials on a number of topics, including:

- An introduction to HPC.
- R.
- Python.
- C and Fortran programming.
- Parallel programming with OpenMP and MPI.
- Using tools such as Make and Subversion.

The ACRC resources page, https://www.acrc.bris.ac.uk/acrc/resources.htm, contains links to a number of How-To guides for topics such as:

- Using MATLAB on BCp3.
- Getting started with compiler flags.
- Linking to libraries, such as LAPACK and BLAS used for linear algebra.
- Using the DDT parallel debugger.
- Using Intel's VTune profiler.

Appendix – Web Links

Below are some web links which you may find useful.

Programming Guides

Here is a collection of programming tutorials available on the web. C/C++ and Fortran are the principle languages used to write scientific software. I've included links to tutorials for Python, R and Matlab in a later section. This section also contains links to tutorials on parallel programming prepared by a couple of the US national laboratories. OpenMP is a popular tool for writing threaded code which runs within a node. MPI remains the most commonly used technology for distributed memory programming with message passing between nodes. The link from Oak Ridge National Laboratory contains a lot of material on programming accelerators, such as GPUs, using technologies such as OpenCL, CUDA and OpenACC.

- C programming: http://www.cprogramming.com/tutorial.html
- C++:
  - http://www.learncpp.com/
  - http://www.cprogramming.com/tutorial/c++-tutorial.html
  - http://www.penguinprogrammer.co.uk/c-beginners-tutorial/
- Fotran:
  - https://www.dur.ac.uk/resources/its/info/guides/138fortran90.pdf
  - http://www.mrao.cam.ac.uk/~rachael/compphys/SelfStudyF95.pdf
- POSIX threads, OpenMP & MPI: https://computing.llnl.gov/?set=training&page=index
- https://computing.llnl.gov/tutorials/parallel_comp/
- OpenACC, CUDA (inc. CUDA aware MPI) & OpenCL:
  https://www.olcf.ornl.gov/support/tutorials/

Tools: Compilers, debuggers, profilers & version control

This section provides some links for the primary tools used for software development.

This first group of links covers the most commonly used compilers.

- GCC: https://gcc.gnu.org/
- Intel: https://software.intel.com/en-us/intel-compilers
- PGI: http://www.pgroup.com/
- Open64: http://www.open64.net/

GDB is a widely used debugger, however it's support for parallel code is limited to threads only. Valgrind is a very useful suite of tools bundled into the same front-end. It contains a memory checker (useful for finding illegal memory accesses and leaks), function and cache profilers as well as a thread error checker. For parallel code, be aware that you can use valgrind with threaded code but that it will execute only one thread at a time. Allinea's DDT can be used to debug both threaded and MPI parallel code. It is available on BCp3.

- GDB:
  - http://www.gnu.org/software/gdb/
  - http://www.cs.cmu.edu/~gilpin/tutorial/
- Valgrind: http://valgrind.org/
- Allinea's DDT: http://www.allinea.com/products/ddt

The third group of links covers profiling tools. gprof is convenient for serial code, but cannot be used with parallel code. TAU and PAPI are very useful, open-source profiling tools. TAU is a profiler that can be used with both threaded and MPI code. PAPI is a library of routines that you can include in your code for timing as well as monitoring aspects of code execution such as cache misses. Intel's VTune is a powerful profiler which can be used with threaded and MPI code. It too can provide a profile of cache use, as well as monitoring locks and waits etc.

- gprof: https://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html
- TAU: https://www.cs.uoregon.edu/research/tau/home.php
- PAPI: http://icl.cs.utk.edu/papi/
- Intel's VTune: https://software.intel.com/en-us/intel-vtune-amplifier-xe

This last group provides links to more information on version control tools. All are well-used and open-source. Also included are links to two popular hosting sites, github and bitbucket. Be sure to check the small print regarding location of hosting and intellectual property if you choose to host your code offsite.

- Subversion: https://subversion.apache.org/
- Git: http://git-scm.com/
- Mercurial: http://mercurial.selenic.com/
- Github: https://github.com/
- Bitbucket: https://bitbucket.org/

Packages, applications and libraries

Third party libraries can offer a great deal. Firstly they contain, potentially very intricate routines, which you can used use without investing the time to write them first. They are often implemented very efficiently and also tested and debugged. Three very good reasons to adopt libraries, where they exist. Venerable examples include the BLAS and LAPACK libraries for linear alegra, alonfg with their associated parallel implementations, PLASMA and MAGMA. MUMPS is a library for the direct solution of systems of linear equations involving sparse matrices. PETSc offers access to many Krylov sub-space methods as well as a multi-grid pre-conditioner. Trilinos is collection of algorithms for the solution of large-scale, complex multi-physics engineering and scientific problems, presented as a group of packages.

Sometimes the boundaries between a package and a language are blurred. R, Python and Matlab potentially fall into this category and links to tutorial information is included in this section.

- BLAS: http://www.netlib.org/blas/
- BOOST: http://www.boost.org/
- LAPACK: http://www.netlib.org/lapack/
- MAGMA: http://icl.cs.utk.edu/magma/
- Matlab: http://uk.mathworks.com/academia/student_center/tutorials/launchpad.html
- MUMPS: http://mumps.enseeiht.fr/
- PETSc:
  - https://www.olcf.ornl.gov/tutorials/petsc/
  - http://www.mcs.anl.gov/petsc/documentation/tutorials/
- PLASMA: http://icl.cs.utk.edu/plasma/
- Python:
  - http://www.learnpython.org/
  - http://software-carpentry.org/v4/python/
  - https://docs.python.org/2/tutorial/
  - http://lorenabarba.com/blog/cfd-python-12-steps-to-navier-stokes/
- R:
  - http://cran.r-project.org/doc/manuals/r-release/R-intro.html
  - http://msenux.redwoods.edu/mbutler/math15/R/
  - http://www.cyclismo.org/tutorial/R/
- Trilinos: http://trilinos.sandia.gov/documentation.html