<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6 Example parallel MPI job submission scripts</td>
<td>37</td>
</tr>
<tr>
<td>6.7 Serial Jobs</td>
<td>40</td>
</tr>
<tr>
<td>6.8 Job arrays</td>
<td>41</td>
</tr>
<tr>
<td>6.9 Interactive Jobs</td>
<td>42</td>
</tr>
<tr>
<td>6.10 Reservations</td>
<td>42</td>
</tr>
<tr>
<td>7 Singularity Containers</td>
<td>45</td>
</tr>
<tr>
<td>7.1 Useful Links</td>
<td>45</td>
</tr>
<tr>
<td>7.2 About Singularity Containers (Images)</td>
<td>45</td>
</tr>
<tr>
<td>7.3 Using Singularity Images on Cirrus</td>
<td>46</td>
</tr>
<tr>
<td>7.4 Creating Your Own Singularity Images</td>
<td>49</td>
</tr>
<tr>
<td>8 Using Python</td>
<td>51</td>
</tr>
<tr>
<td>8.1 Accessing central Anaconda Python</td>
<td>51</td>
</tr>
<tr>
<td>8.2 Custom Environments</td>
<td>52</td>
</tr>
<tr>
<td>9 Using the Cirrus GPU Nodes</td>
<td>57</td>
</tr>
<tr>
<td>9.1 Hardware details</td>
<td>57</td>
</tr>
<tr>
<td>9.2 Compiling software for the GPU nodes</td>
<td>58</td>
</tr>
<tr>
<td>9.3 Submitting jobs to the GPU nodes</td>
<td>59</td>
</tr>
<tr>
<td>10 The Object Store</td>
<td>63</td>
</tr>
<tr>
<td>10.1 Access Keys</td>
<td>63</td>
</tr>
<tr>
<td>10.2 Buckets</td>
<td>64</td>
</tr>
<tr>
<td>10.3 Permissions and ACLs</td>
<td>64</td>
</tr>
<tr>
<td>10.4 Managing the Object Store from SAFE</td>
<td>65</td>
</tr>
<tr>
<td>10.5 Browsing the Object store from your desktop</td>
<td>65</td>
</tr>
<tr>
<td>10.6 Using s3cmd to work with the object store on Cirrus</td>
<td>66</td>
</tr>
<tr>
<td>11 References and further reading</td>
<td>69</td>
</tr>
<tr>
<td>11.1 Online Documentation and Resources</td>
<td>69</td>
</tr>
<tr>
<td>11.2 MPI programming</td>
<td>69</td>
</tr>
<tr>
<td>11.3 OpenMP programming</td>
<td>69</td>
</tr>
<tr>
<td>11.4 Parallel programming</td>
<td>70</td>
</tr>
<tr>
<td>11.5 Programming languages</td>
<td>70</td>
</tr>
<tr>
<td>11.6 Programming skills</td>
<td>70</td>
</tr>
<tr>
<td>12 Altair Hyperworks</td>
<td>71</td>
</tr>
<tr>
<td>12.1 Useful Links</td>
<td>71</td>
</tr>
<tr>
<td>12.2 Using Hyperworks on Cirrus</td>
<td>71</td>
</tr>
<tr>
<td>12.3 Running serial Hyperworks jobs</td>
<td>71</td>
</tr>
<tr>
<td>12.4 Running parallel Hyperworks jobs</td>
<td>72</td>
</tr>
<tr>
<td>13 ANSYS Fluent</td>
<td>75</td>
</tr>
<tr>
<td>13.1 Useful Links</td>
<td>75</td>
</tr>
<tr>
<td>13.2 Using ANSYS Fluent on Cirrus</td>
<td>75</td>
</tr>
<tr>
<td>13.3 Running parallel ANSYS Fluent jobs</td>
<td>75</td>
</tr>
<tr>
<td>13.4 Actual Fluent script (“inputfile.fl”):</td>
<td>76</td>
</tr>
<tr>
<td>14 CASTEP</td>
<td>79</td>
</tr>
<tr>
<td>14.1 Useful Links</td>
<td>79</td>
</tr>
<tr>
<td>14.2 Using CASTEP on Cirrus</td>
<td>79</td>
</tr>
<tr>
<td>14.3 Running parallel CASTEP jobs</td>
<td>79</td>
</tr>
<tr>
<td>15 CP2K</td>
<td>81</td>
</tr>
</tbody>
</table>
Cirrus is a HPC and data science service hosted and run by EPCC at The University of Edinburgh. It is one of the EPSRC Tier-2 National HPC Services.

Cirrus is available to industry and academic researchers. For information on how to get access to the system please see the Cirrus website.

The Cirrus facility is based around an SGI ICE XA system. There are 280 standard compute nodes and 2 GPU compute nodes. Each standard compute node has 256 GiB of memory and contains two 2.1 GHz, 18-core Intel Xeon (Broadwell) processors. Each GPU compute node has 384 GiB of memory, contains two 2.4 GHz, 20-core Intel Xeon (Skylake) processors and four NVIDIA Tesla V100-SXM2-16GB (Volta) GPU accelerators connected to the host processors and each other via PCIe. All nodes are connected using a single Infiniband fabric and access the shared, 406 TiB Lustre file system.

This documentation covers:

- Cirrus User Guide: general information on how to use Cirrus
- Software Applications: notes on using specific software applications on Cirrus
- Software Libraries: notes on compiling against specific libraries on Cirrus. Most libraries work as expected so no additional notes are required however a small number require specific documentation
- Software Tools: Information on using tools such as debuggers and profilers on Cirrus

Information on using the SAFE web interface for managing and reporting on your usage on Cirrus can be found on the Tier-2 SAFE Documentation

This documentation draws on the Sheffield Iceberg Documentation and the documentation for the ARCHER National Supercomputing Service.
CHAPTER 1

Introduction

This guide is designed to be a reference for users of the high-performance computing (HPC) facility: Cirrus. It provides all the information needed to access the system, transfer data, manage your resources (disk and compute time), submit jobs, compile programs and manage your environment.

1.1 Acknowledging Cirrus

You should use the following phrase to acknowledge Cirrus in all research outputs that have used the facility:

This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1)

You should also tag outputs with the keyword Cirrus whenever possible.

1.2 Hardware

Details of the Cirrus hardware are available on the Cirrus website:

- Cirrus Hardware

1.3 Useful terminology

This is a list of terminology used throughout this guide and its meaning.

CPUh  Cirrus CPU time is measured in CPUh. Each job you run on the service consumes CPUhs from your budget. You can find out more about CPUhs and how to track your usage in the File and Resource Management
CHAPTER 2

Connecting to Cirrus

On the Cirrus system interactive access can be achieved via SSH, either directly from a command line terminal or using an SSH client. In addition data can be transferred to and from the Cirrus system using `scp` from the command line or by using a file transfer client.

This section covers the basic connection methods. The connection procedure is then expanded on and the use of SSH agent is described for ease of access.

2.1 Interactive access

To log into Cirrus you should use the “login.cirrus.ac.uk” address:

```
ssh [userID@login.cirrus.ac.uk]
```

2.1.1 Initial passwords

The SAFE web interface is used to provide your initial password for logging onto Cirrus (see the Tier-2 SAFE Documentation for more details on requesting accounts and picking up passwords).

Note: you may now change your password on the Cirrus machine itself using the `passwd` command. This change will not be reflected in the SAFE. If you forget your password, you should use the SAFE to request a new one-shot password.

2.2 SSH Clients

Interaction with Cirrus is done remotely, over an encrypted communication channel, Secure Shell version 2 (SSH-2). This allows command-line access to one of the login nodes of a Cirrus, from which you can run commands or use a command-line text editor to edit files. SSH can also be used to run graphical programs such as GUI text editors and debuggers when used in conjunction with an X client.
2.2.1 Logging in from Linux and Macs

Linux distributions and OS X each come installed with a terminal application that can be use for SSH access to the login nodes. Linux users will have different terminals depending on their distribution and window manager (e.g. GNOME Terminal in GNOME, Konsole in KDE). Consult your Linux distribution’s documentation for details on how to load a terminal.

OS X users can use the Terminal application, located in the Utilities folder within the Applications folder.

You can use the following command from the terminal window to login into Cirrus:

```
ssh username@login.cirrus.ac.uk
```

To allow remote programs, especially graphical applications to control your local display, such as being able to open up a new GUI window (such as for a debugger), use:

```
ssh -X username@login.cirrus.ac.uk
```

Some sites recommend using the `-Y` flag. While this can fix some compatibility issues, the `-X` flag is more secure.

Current OS X systems do not have an X window system. Users should install the XQuartz package to allow for SSH with X11 forwarding on OS X systems:

- XQuartz website

2.2.2 Logging in from Windows using MobaXterm

A typical Windows installation will not include a terminal client, though there are various clients available. We recommend all our Windows users to download and install MobaXterm to access Cirrus. It is very easy to use and includes an integrated X server with SSH client to run any graphical applications on Cirrus.

You can download MobaXterm Home Edition (Installer Edition) from the following link:

- Install MobaXterm

Double-click the downloaded Microsoft Installer file (.msi), and the Windows wizard will automatically guides you through the installation process. Note, you might need to have administrator rights to install on some Windows OS. Also make sure to check whether Windows Firewall hasn’t blocked any features of this program after installation.

Start MobaXterm using, for example, the icon added to the Start menu during the installation process.

If you would like to run any small remote GUI applications, then make sure to use -X option along with the ssh command (see above) to enable X11 forwarding, which allows you to run graphical clients on your local X server.

2.3 Making access more convenient using a SSH Agent

Using a SSH Agent makes accessing the resources more convenient as you only have to enter your passphrase once per day to access any remote resource - this can include accessing resources via a chain of SSH sessions.

This approach combines the security of having a passphrase to access remote resources with the convenience of having password-less access. Having this sort of access set up makes it extremely convenient to use client applications to access remote resources, for example:

- the Tramp Emacs plugin that allows you to access and edit files on a remote host as if they are local files;
- the Parallel Tools Platform for the Eclipse IDE that allows you to edit your source code on a local Eclipse installation and compile and test on a remote host;
Note: this description applies if your local machine is Linux or macOS. The procedure can also be used on Windows using the PuTTY SSH terminal with the PuTTYgen key pair generation tool and the Pageant SSH Agent. See the PuTTY documentation for more information on how to use these tools.

Note: not all remote hosts allow connections using a SSH key pair. If you find this method does not work it is worth checking with the remote site that such connections are allowed.

### 2.3.1 Setup a SSH key pair protected by a passphrase

Using a terminal (the command line), set up a key pair that contains your e-mail address and enter a passphrase you will use to unlock the key:

```bash
ssh-keygen -t rsa -C "your@email.com"
```

- `bash-4.1$ ssh-keygen -t rsa -C "your@email.com"
  
  Generating public/private rsa key pair.
  
Enter file in which to save the key (/Home/user/.ssh/id_rsa): [Enter]
  
Enter passphrase (empty for no passphrase): [Passphrase]
  
Enter same passphrase again: [Passphrase]
  
  Your identification has been saved in /Home/user/.ssh/id_rsa.
  
  Your public key has been saved in /Home/user/.ssh/id_rsa.pub.
  
  The key fingerprint is:
  
  03:d4:c4:6d:58:0a:e2:4a:f8:73:9a:e8:e3:07:16:c8 your@email.com
  
  The key's randomart image is:
  
  +--[ RSA 2048]----+
  | . ...+o++++. |
  | . . .=o.. |
  |+ . . .......o o |
  |oE . . |
  |o = . S |
  |. +.+ |
  |. oo |
  |. . |
  | .. |
  +-----------------+
```

(remember to replace “your@email.com” with your e-mail address).

### 2.3.2 Copy the public part of the key to the remote host

Using your normal login password, add the public part of your key pair to the “authorized_keys” file on the remote host you wish to connect to using the SSH Agent. This can be achieved by appending the contents of the public part of the key to the remote file:

```bash
bash-4.1$ cat ~/.ssh/id_rsa.pub | ssh user@login.cirrus.ac.uk 'cat - >> ~/.ssh/authorized_keys'
```

Password: [Password]

(remember to replace “user” with your username).

Now you can test that your key pair is working correctly by attempting to connect to the remote host and run a command. You should be asked for your key pair passphrase (which you entered when you created the key pair) rather than your remote machine password.

### 2.3. Making access more convenient using a SSH Agent
2.3.3 Enabling the SSH Agent

So far we have just replaced the need to enter a password to access a remote host with the need to enter a key pair passphrase. The next step is to enable an SSH Agent on your local system so that you only have to enter the passphrase once per day and after that you will be able to access the remote system without entering the passphrase.

Most modern Linux distributions (and macOS) should have ssh-agent running by default. If your system does not then you should find the instructions for enabling it in your distribution using Google.

To add the private part of your key pair to the SSH Agent, use the `ssh-add` command (on your local machine), you will need to enter your passphrase one more time:

```
-bash-4.1$ ssh-add ~/.ssh/id_rsa
Enter passphrase for Home/user.ssh/id_rsa: [Passphrase]
Identity added: Home/user.ssh/id_rsa (Home/user.ssh/id_rsa)
```

Now you can test that you can access the remote host without needing to enter your passphrase:

```
-bash-4.1$ ssh user@login.cirrus.ac.uk 'date'
Warning: Permanently added the RSA host key for IP address '192.62.216.27' to the list of known hosts.
Wed May  8 10:42:55 BST 2013
```

(remember to replace “user” with your username).

2.3.4 Adding access to other remote machines

If you have more than one remote host that you access regularly, you can simply add the public part of your key pair to the ‘authorized_keys’ file on any hosts you wish to access by repeating step 2 above.

2.3.5 SSH Agent forwarding

Now that you have enabled an SSH Agent to access remote resources you can perform an additional configuration step that will allow you to access all hosts that have your public key part uploaded from any host you connect to with the SSH Agent without the need to install the private part of the key pair anywhere except your local machine.

This increases the security of the key pair as the private part is only stored in one place (your local machine) and makes access more convenient (as you only need to enter your passphrase once on your local machine to enable access between all machines that have the public part of the key pair).

Forwarding is controlled by a configuration file located on your local machine at “.ssh/config”. Each remote site (or group of sites) can have an entry in this file which may look something like:

```
Host cirrus
  HostName login.cirrus.ac.uk
  User user
  ForwardAgent yes
```
The “Host cirrus” line defines a short name for the entry. In this case, instead of typing “ssh login.cirrus.ac.uk” to access the Cirrus login nodes, you could use “ssh cirrus” instead. The remaining lines define the options for the “cirrus” host.

- Hostname login.cirrus.ac.uk - defines the full address of the host
- User username - defines the username to use by default for this host (replace “username” with your own username on the remote host)
- ForwardAgent yes - tells SSH to forward the local SSH Agent to the remote host, this is the option that allows you to store the private part of your key on your local machine only and export the access to remote sites

Now you can use SSH to access Cirrus without needing to enter my username or the full hostname every time:

```
-bash-4.1$ ssh cirrus 'date'
Tue Dec 20 16:48:32 GMT 2016
```

You can set up as many of these entries as you need in your local configuration file. Other options are available. See the `ssh_config man page` (or `man ssh_config` on any machine with SSH installed) for a description of the SSH configuration file.
CHAPTER 3

Data Transfer Guide

This section covers the different ways that you can transfer data on to and off Cirrus. In particular, we cover:

- Using the Cirrus Data Services Node (DSN)
- SSH-based methods (scp, sftp, rsync)
- Using the UK Research Data Facility

In all cases of data transfer, users should use the Cirrus DSN in preference to the Cirrus login nodes as the DSN has higher bandwidth external network connections (10 Gb/s rather than 1 Gb/s on the login nodes).

3.1 Before you start

Read Harry Mangalam’s guide on How to transfer large amounts of data via network. This tells you all you want to know about transferring data.

3.2 Accessing the Cirrus Data Services Node (DSN)

You can access the Cirrus DSN using your Cirrus username at the address dsn.cirrus.ac.uk:

```
ssh username@dsn.cirrus.ac.uk
```

(remember to replace username with your Cirrus username).

The Cirrus DSN has high bandwidth connections (10 Gb/s) to the external network and so has the potential to give higher performance for data transfers.
3.3 Data Transfer via SSH

The easiest way of transferring data to/from Cirrus is to use one of the standard programs based on the SSH protocol such as `scp`, `sftp` or `rsync`. These all use the same underlying mechanism (ssh) as you normally use to log-in to Cirrus. So, once the command has been executed via the command line, you will be prompted for your password for the specified account on the remote machine.

To avoid having to type in your password multiple times you can set up an `ssh-key` as documented in the User Guide at

*Connecting to Cirrus*

### 3.3.1 SSH Transfer Performance Considerations

The ssh protocol encrypts all traffic it sends. This means that file-transfer using ssh consumes a relatively large amount of CPU time at both ends of the transfer. The Cirrus DSN has fairly fast processors that can sustain about 100 MB/s transfer. The encryption algorithm used is negotiated between the ssh-client and the ssh-server. There are command line flags that allow you to specify a preference for which encryption algorithm should be used. You may be able to improve transfer speeds by requesting a different algorithm than the default. The `arcfour` algorithm is usually quite fast if both hosts support it.

A single ssh based transfer will usually not be able to saturate the available network bandwidth or the available disk bandwidth so you may see an overall improvement by running several data transfer operations in parallel. To reduce metadata interactions it is a good idea to overlap transfers of files from different directories.

In addition, you should consider the following when transferring data:

1. Only transfer those files that are required. Consider which data you really need to keep.
2. Combine lots of small files into a single `tar` archive, to reduce the overheads associated in initiating many separate data transfers (over SSH each file counts as an individual transfer).
3. Compress data before sending it, e.g. using `gzip`.

### 3.3.2 scp command

The `scp` command creates a copy of a file, or if given the `-r` flag, a directory, on a remote machine.

For example, to transfer files to Cirrus:

```
scp [options] source user@dsn.cirrus.ac.uk:[destination]
```

(Remember to replace `user` with your Cirrus username in the example above.)

In the above example, the `[destination]` is optional, as when left out `scp` will simply copy the source into the user's home directory. Also the `source` should be the absolute path of the file/directory being copied or the command should be executed in the directory containing the source file/directory.

If you want to request a different encryption algorithm add the `-c [algorithm-name]` flag to the `scp` options. For example, to use the (usually faster) `arcfour` encryption algorithm you would use:

```
scp [options] -c arcfour source user@dsn.cirrus.ac.uk:[destination]
```

(Remember to replace `user` with your Cirrus username in the example above.)
### 3.3.3 rsync command

The `rsync` command can also transfer data between hosts using a `ssh` connection. It creates a copy of a file or, if given the `-r` flag, a directory at the given destination, similar to `scp` above.

Given the `-a` option `rsync` can also make exact copies (including permissions), this is referred to as *mirroring*. In this case the `rsync` command is executed with `ssh` to create the copy on a remote machine.

To transfer files to Cirrus using `rsync` the command should have the form:

```
rsync [options] -e ssh source user@dsn.cirrus.ac.uk:[destination]
```

(Remember to replace `user` with your Cirrus username in the example above.)

In the above example, the `[destination]` is optional, as when left out `rsync` will simply copy the source into the users home directory. Also the `source` should be the absolute path of the file/directory being copied or the command should be executed in the directory containing the source file/directory.

Additional flags can be specified for the underlying `ssh` command by using a quoted string as the argument of the `-e` flag, e.g.

```
rsync [options] -e "ssh -c arcfour" source user@login.cirrus.ac.uk:[destination]
```

(Remember to replace `user` with your Cirrus username in the example above.)

### 3.4 Using the RDF from Cirrus

The Cirrus DSN provides access to the RDF file system via direct mounts on a virtual machine (VM). To access the VM with the RDF file system mounts you should log into the DSN as normal and then use SSH to connect to the `dsn-rdf` VM:

```
ssh user@dsn-rdf
```

(Remember to replace `user` with your Cirrus username in the example above.)

Once you are on the RDF access VM, you will be able to find the RDF file systems mounted as:

- `/epsrc`
- `/nerc`
- `/general`

The specific file system for your project’s data will depend on which was allocated when the project was setup.

**Note:** Not all projects on Cirrus have space allocated on the RDF. If you are unsure if you have space or not, please contact the [Cirrus Helpdesk](#).

### 3.4.1 Moving data between the RDF and the Cirrus file system

The simplest (and most efficient) way to do this is to use the `cp` command on the RDF access VM. For example, once you are logged onto the RDF access VM you could copy data from the RDF to the Cirrus file system with:

```
cp /general/t01/t01/user/some_data.tar.gz /lustre/home/t01/user/
```
Warning: You should never use `mv` to move data between RDF file systems and Cirrus file systems (or between any two different file systems) as there is the potential to lose data. You should always copy the data, verify that the copy is not corrupted and then delete the original version.

### 3.4.2 Transferring data to/from the RDF

If you wish to transfer data to/from the RDF then you should use the RDF Data Transfer Nodes (DTNs) rather than the Cirrus DSN node. Documentation on how to transfer data to/from the RDF can be found on the RDF website:

- RDF Data Transfer Guide
This section covers some of the tools and technical knowledge that will be key to maximising the usage of the Cirrus system, such as the online administration tool SAFE and calculating the CPU-time available.

The default file permissions are then outlined, along with a description of changing these permissions to the desired setting. This leads on to the sharing of data between users and systems often a vital tool for project groups and collaboration.

Finally we cover some guidelines for I/O and data archiving on Cirrus.

### 4.1 The Cirrus Administration Web Site (SAFE)

All users have a login and password on the Cirrus Administration Web Site (also know as the ‘SAFE’): SAFE. Once logged into this web site, users can find out much about their usage of the Cirrus system, including:

- Account details - password reset, change contact details
- Project details - project code, start and end dates
- CPUh balance - how much time is left in each project you are a member of
- Filesystem details - current usage and quotas
- Reports - generate reports on your usage over a specified period, including individual job records
- Helpdesk - raise queries and track progress of open queries

### 4.2 Checking your CPU-time allocation

You can view these details by logging into the SAFE (https://www.archer.ac.uk/tier2/).

Use the Login accounts menu to select the user account that you wish to query. The page for the login account will summarise the resources available to account.
You can also generate reports on your usage over a particular period and examine the details of how many CPUh individual jobs on the system cost. To do this use the Service information menu and select Report generator.

### 4.3 Disk quotas

Disk quotas on Cirrus are managed via SAFE

For live disk usage figures use

```
lfs quota -hu <username> /lustre
lfs quota -hg <groupname> /lustre
```

### 4.4 File permissions and security

By default, each user is a member of the group with the same name as [group_code] in the /lustre/home directory path, e.g. x01. This allows the user to share files with only members of that group by setting the appropriate group file access permissions. As on other UNIX or Linux systems, a user may also be a member of other groups. The list of groups that a user is part of can be determined by running the `groups` command.

Default Unix file permissions can be specified by the `umask` command. The default umask value on Cirrus is 22, which provides “group” and “other” read permissions for all files created, and “group” and “other” read and execute permissions for all directories created. This is highly undesirable, as it allows everyone else on the system to access (but at least not modify or delete) every file you create. Thus it is strongly recommended that users change this default umask behaviour, by adding the command `umask 077` to their `$HOME/.profile` file. This umask setting only allows the user access to any file or directory created. The user can then selectively enable “group” and/or “other” access to particular files or directories if required.

#### 4.4.1 ASCII (or formatted) files

These are the most portable, but can be extremely inefficient to read and write. There is also the problem that if the formatting is not done correctly, the data may not be output to full precision (or to the subsequently required precision), resulting in inaccurate results when the data is used. Another common problem with formatted files is FORMAT statements that fail to provide an adequate range to accommodate future requirements, e.g. if we wish to output the total number of processors, NPROC, used by the application, the statement:

```
WRITE (*,'I3') NPROC
```

will not work correctly if NPROC is greater than 999.

#### 4.4.2 Binary (or unformatted) files

These are much faster to read and write, especially if an entire array is read or written with a single READ or WRITE statement. However the files produced may not be readable on other systems.

**GNU compiler -fconvert=swap compiler option.** This compiler option often needs to be used together with a second option -frecord-marker, which specifies the length of record marker (extra bytes inserted before or after the actual data in the binary file) for unformatted files generated on a particular system. To read a binary file generated by a big-endian system on Cirrus, use `-fconvert=swap -frecord-marker=4`. Please note that due to the same ‘length of record marker’ reason, the unformatted files generated by GNU and other compilers on Cirrus are not compatible. In fact, the same WRITE statements would result in slightly larger files
with GNU compiler. Therefore it is recommended to use the same compiler for your simulations and related pre- and post-processing jobs.

Other options for file formats include:

**Direct access files** Fortran unformatted files with specified record lengths. These may be more portable between different systems than ordinary (i.e. sequential IO) unformatted files, with significantly better performance than formatted (or ASCII) files. The “ endian” issue will, however, still be a potential problem.

**Portable data formats** These machine-independent formats for representing scientific data are specifically designed to enable the same data files to be used on a wide variety of different hardware and operating systems. The most common formats are:

- netCDF: [http://www.unidata.ucar.edu/software/netcdf/](http://www.unidata.ucar.edu/software/netcdf/)
- HDF: [http://www.hdfgroup.org/HDF5/](http://www.hdfgroup.org/HDF5/)

It is important to note that these portable data formats are evolving standards, so make sure you are aware of which version of the standard/software you are using, and keep up-to-date with any backward-compatibility implications of each new release.

### 4.5 File IO Performance Guidelines

Here are some general guidelines

- Whichever data formats you choose, it is vital that you test that you can access your data correctly on all the different systems where it is required. This testing should be done as early as possible in the software development or porting process (i.e. before you generate lots of data from expensive production runs), and should be repeated with every major software upgrade.

- Document the file formats and metadata of your important data files very carefully. The best documentation will include a copy of the relevant I/O subroutines from your code. Of course, this documentation must be kept up-to-date with any code modifications.

- Use binary (or unformatted) format for files that will only be used on the Intel system, e.g. for checkpointing files. This will give the best performance. Binary files may also be suitable for larger output data files, if they can be read correctly on other systems.

- Most codes will produce some human-readable (i.e. ASCII) files to provide some information on the progress and correctness of the calculation. Plan ahead when choosing format statements to allow for future code usage, e.g. larger problem sizes and processor counts.

- If the data you generate is widely shared within a large community, or if it must be archived for future reference, invest the time and effort to standardise on a suitable portable data format, such as netCDF or HDF.

### 4.6 Backup policies

There are currently no backups of data on Cirrus as backing up the whole Lustre file system would adversely affect the performance of write access for simulations. The nature of the Lustre parallel file system means that there is data resilience in the case of failures of individual hardware components. However, we strongly advise that you keep copies of any critical data on different systems.

We are currently investigating options for providing backups of critical data.
The application development environment on Cirrus is primarily controlled through the *modules* environment. By loading and switching modules you control the compilers, libraries and software available.

This means that for compiling on Cirrus you typically set the compiler you wish to use using the appropriate modules, then load all the required library modules (e.g. numerical libraries, IO format libraries).

Additionally, if you are compiling parallel applications using MPI (or SHMEM, etc.) then you will need to load one of the MPI environments and use the appropriate compiler wrapper scripts.

By default, all users on Cirrus start with no modules loaded.

Basic usage of the `module` command on Cirrus is covered below. For full documentation please see:

- Linux manual page on modules

### 5.1 Using the modules environment

#### 5.1.1 Information on the available modules

Finding out which modules (and hence which compilers, libraries and software) are available on the system is performed using the `module avail` command:

```
[user@cirrus-login0 ~]$ module avail
...
```

This will list all the names and versions of the modules available on the service. Not all of them may work in your account though due to, for example, licencing restrictions. You will notice that for many modules we have more than one version, each of which is identified by a version number. One of these versions is the default. As the service develops the default version will change.

You can list all the modules of a particular type by providing an argument to the `module avail` command. For example, to list all available versions of the Intel Compiler type:
If you want more info on any of the modules, you can use the `module help` command:

```bash
[user@cirrus-login0 ~]$ module help mpt
```

---

**Module Specific Help for 'mpt/2.14'**

The HPE Message Passing Toolkit (MPT) is an optimized MPI implementation for HPE systems and clusters. See the MPI(1) man page and the MPT User's Guide for more information.

The simple `module list` command will give the names of the modules and their versions you have presently loaded in your environment:

```bash
[user@cirrus-login0 ~]$ module list
```

---

**Currently Loaded Modulefiles:**

1) mpt/2.14 3) intel-fc-16/16.0.3.210
2) intel-cc-16/16.0.3.210 4) intel-compilers-16/16.0.3.210

### 5.1.2 Loading, unloading and swapping modules

To load a module to use `module add` or `module load`. For example, to load the intel-compilers-17 into the development environment:

```bash
module load intel-compilers-17
```

This will load the default version of the intel compilers. If you need a specific version of the module, you can add more information:

```bash
module load intel-compilers-17/17.0.2.174
```

will load version 17.0.2.174 for you, regardless of the default. If you want to clean up, `module remove` will remove a loaded module:

```bash
module remove intel-compilers-17
```

(or `module rm intel-compilers-17` or `module unload intel-compilers-17`) will unload whatever version of intel-compilers-17 (even if it is not the default) you might have loaded. There are many situations in which you might want to change the presently loaded version to a different one, such as trying the latest version which is not yet the default or using a legacy version to keep compatibility with old data. This can be achieved most easily by using “module swap oldmodule newmodule”.

Suppose you have loaded version 16.0.2.181, say, of intel-compilers-16, the following command will change to version 16.0.3.210:

```bash
module swap intel-compilers-16 intel-compilers-16/16.0.2.181
```
5.1.3 Modules provided by Spack

Note: The majority of users will not need to use the modules provided by Spack. The standard set of modules available to users should cover most common use cases on Cirrus.

The Spack package manager provides many more modules (particularly for libraries and dependencies) than are visible by default to users. If you wish to see or use the modules provided by Spack, then you must first load the spack module:

```
module load spack
```

Once this module is loaded, the `module avail` command will list the additional modules that have been installed using Spack.

Care must be taken when using modules provided by Spack as they behave differently from standard Linux modules. The Spack package management tool is used to manage much of the software and libraries installed on Cirrus. Spack allows us to automatically resolve dependencies and have multiple versions of tested software installed simultaneously without them interfering with each other.

To achieve this, Spack makes use of RPATH to hardcode the paths of dependencies into libraries. This means that when you load a module for a particular library you do not need to load any further modules for dependencies of that library.

For example, the boost toolkit depends on the MPI, zlib and bzip2 libraries:

```
boost@1.64.0
    ^bzip2@1.0.6
    ^mpich@2.14
    ^zlib@1.2.10
```

Spack arranges things so that if you load the boost module:

```
module load boost-1.64.0-gcc-6.2.0-pftxg46
```

then you do not also need to load the bzip2, mpich and zlib modules.

This, however, can lead to behaviour that is unexpected for modules. For example, on Cirrus there are two versions of zlib available: 1.2.8 and 1.2.10. You may imagine that you can use boost with zlib 1.2.8 with the following commands:

```
module load zlib-1.2.8-gcc-6.2.0-epathtp
module load boost-1.64.0-gcc-6.2.0-pftxg46
```

but this will not work. boost will still use zlib 1.2.10 as the path to this is hardcoded into boost itself via RPATH. If you wish to use the older version of zlib then you must load it and then compile boost yourself.

If you wish to see what versions of libraries are hardcoded into a particular Spack module then you must use Spack commands, e.g.

```
[auser@cirrus-login0 ~]$ module load spack
[auser@cirrus-login0 ~]$ module avail boost
------------ /lustre/sw/spack/share/spack/modules/linux-centos7-x86_64 ------------
boost-1.63.0-intel-17.0.2-f125xqm boost-1.64.0-gcc-6.2.0-pftxg46
```

```
[auser@cirrus-login0 ~]$ spack find -dl boost
```

(continues on next page)
This shows there are two boost modules installed (one for the Intel compilers and one for the GCC compilers), they both depend on zlib 1.0.6 and bzip2 1.2.10 and the GCC version also depends on MPI 2.14 (HPE MPT 2.14). The paths for these dependencies are hardcoded into the boost RPATH.

### 5.2 Available Compiler Suites

**Note:** As Cirrus uses dynamic linking by default you will generally also need to load any modules you used to compile your code in your job submission script when you run your code.

#### 5.2.1 Intel Compiler Suite

The Intel compiler suite is accessed by loading the `intel-compilers-*` module. For example:

```
module load intel-compilers-17
```

Once you have loaded the module, the compilers are available as:

- `ifort` - Fortran
- `icc` - C
- `icpc` - C++

**C++ with Intel Compilers**

Intel compilers rely on GCC C++ headers and libraries to support most recent C++ features. If you are using Intel compilers to compile C++ on Cirrus you should also load the `gcc/6.2.0` module to have access to the correct C++ files:

```
:: module load gcc/6.2.0
```

**Note:** You will also need to load this module in your job submission scripts when running code compiled in this way.

#### 5.2.2 GCC Compiler Suite

The GCC compiler suite is accessed by loading the `gcc` module. For example:
Once you have loaded the module, the compilers are available as:

- gfortran - Fortran
- gcc - C
- g++ - C++

### 5.3 Compiling MPI codes

There are two MPI libraries currently available on Cirrus:

- HPE Message Passing Toolkit (MPT)
- Intel MPI

The compilation and run commands are different depending on which of these libraries you choose. Most of the applications we have compiled on Cirrus have made use of the HPE MPT library and we only use Intel MPI if HPE MPT cannot be used for some reason. If you can use either library it is worthwhile running a few tests to discover if either provides a performance advantage for your application.

The following sections discuss each of the MPI library options in turn.

You should also consult the chapter on running jobs through the batch system for examples of how to run jobs compiled against the different MPI libraries.

**Note:** By default, all compilers produce dynamic executables on Cirrus. This means that you must load the same modules at runtime (usually in your job submission script) as you have loaded at compile time.

#### 5.3.1 Using HPE MPT

To compile MPI code with HPE MPT, using any compiler, you must first load the “mpt” module.

```
module load mpt
```

This makes the compiler wrapper scripts mpicc, mpicxx and mpif90 available to you.

What you do next depends on which compiler (Intel or GCC) you wish to use to compile your code.

**Note:** We recommend that you use the Intel compiler wherever possible to compile MPI applications as this is the method officially supported and tested by HPE.

**Note:** You can always check which compiler the MPI compiler wrapper scripts are using with, for example, `mpicc -v` or `mpif90 -v`.

### Using Intel Compilers and HPE MPT

Once you have loaded the MPT module you should next load the appropriate `intel-compilers` module (e.g. `intel-compilers-17`):
module load intel-compilers-17

Remember, if you are compiling C++ code, then you will also need to load the gcc/6.2.0 module for the C++ 11 headers to be available.

Compilers are then available as

- mpif90 - Fortran with MPI
- mpicc - C with MPI
- mpicxx - C++ with MPI

**Note:** mpicc uses gcc by default:

When compiling C applications you must also specify that mpicc should use the icc compiler with, for example, mpicc -cc=icc. (This is not required for Fortran as the mpif90 compiler automatically uses ifort.) If in doubt use mpicc -cc=icc -v to see which compiler is actually being called.

Alternatively, you can set the environment variable MPICC_CC=icc to ensure the correct base compiler is used:

```
export MPICC_CC=icc
```

**Note:** mpicxx uses g++ by default:

When compiling C++ applications you must also specify that

mpicxx should use the icpc compiler with, for example, mpicxx -cxx=icpc. (This is not required for Fortran as the mpif90 compiler automatically uses ifort.) If in doubt use mpicxx -cxx=icpc -v to see which compiler is actually being called.

Alternatively, you can set the environment variable MPICXX_CXX=icpc to ensure the correct base compiler is used:

```
export MPICXX_CXX=icpc
```

### Using GCC Compilers and HPE MPT

Once you have loaded the MPT module you should next load the gcc module:

```
module load gcc
```

Compilers are then available as

- mpif90 - Fortran with MPI
- mpicc - C with MPI
- mpicxx - C++ with MPI

**Note:** HPE MPT does not support the syntax `use mpi` in Fortran applications with the GCC compiler gfortran. You should use the older `include "mpif.h"` syntax when using GCC compilers with mpif90. If you cannot change this, then use the Intel MPI library.
5.3.2 Using Intel MPI

To compile MPI code with Intel MPI, using any compiler, you must first load the “intel-mpi-17” module:

```
module load intel-mpi-17
```

This makes the compiler wrapper scripts available to you. The name of the wrapper script depends on the compiler suite you are using. In summary:

<table>
<thead>
<tr>
<th>Language</th>
<th>Intel</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>mpiifort</td>
<td>mpif90</td>
</tr>
<tr>
<td>C++</td>
<td>mpiicpc</td>
<td>mpicxx</td>
</tr>
<tr>
<td>C</td>
<td>mpiicc</td>
<td>mpicc</td>
</tr>
</tbody>
</table>

Further details on using the different compiler suites with Intel MPI are available in the following sections.

Using Intel Compilers and Intel MPI

Once you have loaded the `intel-mpi-17` module you should next load the appropriate `intel-compilers` module (e.g. `intel-compilers-17`):

```
module load intel-compilers-17
```

Remember, if you are compiling C++ code, then you will also need to load the `gcc/6.2.0` module for the C++ 11 headers to be available.

MPI compilers are then available as

- `mpiifort` - Fortran with MPI
- `mpiicc` - C with MPI
- `mpiicpc` - C++ with MPI

**Note:** Intel compilers with Intel MPI use non-standard compiler wrapper script names. If you use the standard names you will end up using the GCC compilers.

Using GCC Compilers and Intel MPI

Once you have loaded the `intel-mpi-17` module you should next load the `gcc` module.

```
module load gcc
```

MPI compilers are then available as

- `mpif90` - Fortran with MPI
- `mpicc` - C with MPI
- `mpicxx` - C++ with MPI
5.4 Compiler Information and Options

The manual pages for the different compiler suites are available:

GCC Fortran man gfortran, C/C++ man gcc
Intel Fortran man ifort, C/C++ ma ICC

5.4.1 Useful compiler options

Whilst difference codes will benefit from compiler optimisations in different ways, for reasonable performance on Cirrus, at least initially, we suggest the following compiler options:

Intel -O2
GNU -O2 -ftree-vectorize -funroll-loops -ffast-math

When you have a application that you are happy is working correctly and has reasonable performance you may wish to investigate some more aggressive compiler optimisations. Below is a list of some further optimisations that you can try on your application (Note: these optimisations may result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions):

Intel -fast
GNU -Ofast -funroll-loops

Vectorisation, which is one of the important compiler optimisations for Cirrus, is enabled by default as follows:

Intel At -O2 and above
GNU At -O3 and above or when using -ftree-vectorize

To promote integer and real variables from four to eight byte precision for FORTRAN codes the following compiler flags can be used:

Intel -real-size 64 -integer-size 64 -xAVX (Sometimes the Intel compiler incorrectly generates AVX2 instructions if the -real-size 64 or -r8 options are set. Using the -xAVX option prevents this.)
GNU -freal-4-real-8 -finteger-4-integer-8

5.5 Using static linking/libraries

By default, executables on Cirrus are built using shared/dynamic libraries (that is, libraries which are loaded at run-time as and when needed by the application) when using the wrapper scripts.

An application compiled this way to use shared/dynamic libraries will use the default version of the library installed on the system (just like any other Linux executable), even if the system modules were set differently at compile time. This means that the application may potentially be using slightly different object code each time the application runs as the defaults may change. This is usually the desired behaviour for many applications as any fixes or improvements to the default linked libraries are used without having to recompile the application, however some users may feel this is not the desired behaviour for their applications.

Alternatively, applications can be compiled to use static libraries (i.e. all of the object code of referenced libraries are contained in the executable file). This has the advantage that once an executable is created, whenever it is run in the future, it will always use the same object code (within the limit of changing runtime environment). However, executables compiled with static libraries have the potential disadvantage that when multiple instances are running simultaneously multiple copies of the libraries used are held in memory. This can lead to large amounts of memory being used to hold the executable and not application data.
To create an application that uses static libraries you must pass an extra flag during compilation, `-Bstatic`.

Use the UNIX command `ldd exe_file` to check whether you are using an executable that depends on shared libraries. This utility will also report the shared libraries this executable will use if it has been dynamically linked.
The Cirrus facility uses PBSPro to schedule jobs.

Writing a submission script is typically the most convenient way to submit your job to the job submission system. Example submission scripts (with explanations) for the most common job types are provided below.

Interactive jobs are also available and can be particularly useful for developing and debugging applications. More details are available below.

**Note:** There are a number of different queues on Cirrus. In general, you should not specify a queue and the submission system will select the correct one for your job.

If you have any questions on how to run jobs on Cirrus do not hesitate to contact the Cirrus Helpdesk.

### 6.1 Using PBS Pro

You typically interact with PBS by (1) specifying PBS directives in job submission scripts (see examples below) and (2) issuing PBS commands from the login nodes.

There are three key commands used to interact with the PBS on the command line:

- `qsub`
- `qstat`
- `qdel`

Check the PBS man page for more advanced commands:
6.1.1 The qsub command

The qsub command submits a job to PBS:

```
qsub job_script.pbs
```

This will submit your job script “job_script.pbs” to the job-queues. See the sections below for details on how to write job scripts.

Note: There are a number of different queues on Cirrus. In general, you should not specify a queue and the submission system will select the correct one for your job.

6.1.2 The qstat command

Use the command qstat to view the job queue. For example:

```
qstat
```

will list all jobs on Cirrus.

You can view just your jobs by using:

```
qstat -u $USER
```

The -a option to qstat provides the output in a more useful format.

To see more information about a queued job, use:

```
qstat -f $JOBID
```

This option may be useful when your job fails to enter a running state. The output contains a PBS comment field which may explain why the job failed to run.

6.1.3 The qdel command

Use this command to delete a job from Cirrus’s job queue. For example:

```
qdel $JOBID
```

will remove the job with ID $JOBID from the queue.

6.2 Queue Limits

Queues on Cirrus are designed to enable users to use the system flexibly while retaining fair access for all.

6.2.1 Standard compute node queues

There are a number of queues available to general users on Cirrus that route jobs to the standard compute nodes. Standard jobs are automatically routed into either workq, indy or large depending on the project that submitted the job and the size of the job. To use any of these queues, you should not specify a queue name in your job script.
Cirrus Documentation, Release 1.1

• **workq**: Jobs in this queue can have a maximum walltime of 96 hours (4 days) and a maximum job size of 2520 cores (70 nodes). Each project can use a maximum of 2520 cores (70 nodes) summed across all their running jobs at any one time and each user can have a maximum of 20 jobs running (note that these limits can be dynamically altered by the service to improve throughput on the system). Jobs running in this queue are node shared by default (i.e. multiple jobs can share a single compute node). If you want to use node exclusive then you must specify this using the PBS options described below.

• **indy**: Jobs in this queue have a variable maximum job size and walltime (use `qstat -Qf indy` on Cirrus to check the current limits). Jobs running in this queue are node shared by default (i.e. multiple jobs can share a single compute node). If you want to use node exclusive then you must specify this using the PBS options described below. This queue is only available to projects from industrial clients.

• **large**: There is no upper limit on job size in this queue but there is a minimum job size of 2521 cores (71 nodes), a maximum walltime of 48 hours (2 days), each user can have a maximum of 1 job running at any one time, and a maximum of 4 jobs in the queue (including a running job). Jobs running in this queue are node shared by default (i.e. multiple jobs can share a single compute node). If you want to use node exclusive then you must specify this using the PBS options described below.

### 6.2.2 GPU compute node queues

If you wish to use the GPU compute nodes then you need to submit to the `gpu` queue by adding `-q gpu` to your submission. You will also need to specify how many GPU accelerators you wish to use. Full details are available in the GPU chapter of this User Guide.

Jobs in the `gpu` queue can have a maximum walltime of 6 hours. There is a maximum job size of 2 nodes (as there are 2 nodes available). Each user can only have one job running at any one time.

### 6.2.3 Queue error messages

If you try to submit a job that asks for more than the maximum allowed wall time or cores you will see an error similar to:

```
[user@cirrus-login0 ~]$ qsub submit.pbs
qsub: Job violates queue and/or server resource limits
```

### 6.3 Output from PBS jobs

PBS produces standard output and standard error for each batch job can be found in files `<jobname>.o<Job ID>` and `<jobname>.e<Job ID>` respectively. These files appear in the job’s working directory once your job has completed or its maximum allocated time to run (i.e. wall time, see later sections) has run out.

### 6.4 Running Parallel Jobs

This section describes how to write job submission scripts specifically for different kinds of parallel jobs on Cirrus.

All parallel job submission scripts require (as a minimum) you to specify four things:

- The number of nodes and cores per node you require via the `-l select=[Nodes]:ncpus=36` option. Each node has 36 physical cores (2x 18-core sockets). For example, to select 4 nodes (144 physical cores in total) you would use `-l select=4:ncpus=36`. **We strongly recommend that all parallel jobs use node exclusive mode as described below to get best performance.**
• The placement option `-l place=scatter` to ensure that parallel processes/threads are scheduled to the full set of compute nodes assigned to the job.

• The maximum length of time (i.e. walltime) you want the job to run for via the `-l walltime=[hh:mm:ss]` option. To ensure the minimum wait time for your job, you should specify a walltime as short as possible for your job (i.e. if your job is going to run for 3 hours, do not specify 12 hours). On average, the longer the walltime you specify, the longer you will queue for.

• The project code that you want to charge the job to via the `-A [project code]` option

In addition to these mandatory specifications, there are many other options you can provide to PBS. The following options may be useful:

• The name for your job is set using `-N My_job`. In the examples below the name will be “My_job”, but you can replace “My_job” with any name you want. The name will be used in various places. In particular it will be used in the queue listing and to generate the name of your output and/or error file(s). Note there is a limit on the size of the name.

6.4.1 Exclusive Node Access

Exclusive node access means each node is dedicated to one user only.

To make sure your jobs have exclusive node access you should add the `excl` sharing directive to the `place` option in your jobs:

```bash
#PBS -l place=scatter:excl
```

All of our example parallel job submission scripts below specify this option as this mode of use is strongly recommended for all parallel jobs on Cirrus.

6.5 Running MPI parallel jobs

When you are running parallel jobs requiring MPI you will use an MPI launch command to start your executable in parallel. The name and options for this MPI launch command depend on which MPI library you are using: HPE MPT (Message Passing Toolkit), Intel MPI or OpenMPI. We give details below of the commands used in each case and our example job submission scripts have examples for both libraries.

Note: If you are using a centrally-installed MPI software package you will need to know which MPI library was used to compile it so you can use the correct MPI launch command. You can find this information using the `module show` command. For example:

```bash
[auser@cirrus-login0 ~]$ module show vasp
-------------------------------------------------------------------
/lustre/sw/modulefiles/vasp/5.4.4-intel17-mpt214:               
conflict    vasp
module      load mpt
module      load intel-compilers-17
module      load intel-cmk1-17
module      load gcc/6.2.0
prepend-path PATH /lustre/home/y07/vasp5/5.4.4-intel17-mpt214/bin
setenv      VASP5 /lustre/home/y07/vasp5/5.4.4-intel17-mpt214
```

(continues on next page)
This shows that VASP was compiled with HPE MPT (from the `module load mpt` in the output from the command. If a package was compiled with Intel MPI there would be `module load intel-mpi-17` in the output instead.

6.5.1 HPE MPT (Message Passing Toolkit)

HPE MPT is accessed at both compile and runtime by loading the `mpt` module:

```
module load mpt
```

**HPE MPT: parallel launcher `mpiexec_mpt`**

The HPE MPT parallel launcher on Cirrus is `mpiexec_mpt`.

**Note:** This parallel job launcher is only available once you have loaded the `mpt` module.

A sample MPI launch line using `mpiexec_mpt` looks like:

```
mpiexec_mpt -ppn 36 -n 72 ./my_mpi_executable.x arg1 arg2
```

This will start the parallel executable “my_mpi_executable.x” with arguments “arg1” and “arg2”. The job will be started using 72 MPI processes, with 36 MPI processes are placed on each compute node (this would use all the physical cores on each node). This would require 2 nodes to be requested in the PBS options. Note that the ordering of flags is important.

The most important `mpiexec_mpt` flags are:

```
-n [total number of MPI processes] Specifies the total number of distributed memory parallel processes (not including shared-memory threads). For jobs that use all physical cores this will usually be a multiple of 36. The default on Cirrus is 1.

-ppn [parallel processes per node] Specifies the number of distributed memory parallel processes per node. There is a choice of 1-36 for physical cores on Cirrus compute nodes (1-72 if you are using Hyper-Threading) If you are running with exclusive node usage, the most economic choice is always to run with “fully-packed” nodes on all physical cores if possible, i.e. `-ppn 36`. Running “unpacked” or “underpopulated” (i.e. not using all the physical cores on a node) is useful if you need large amounts of memory per parallel process or you are using more than one shared-memory thread per parallel process.
```

**Note:** `mpiexec_mpt` only works from within a PBS job submission script.

**Warning:** You must use the `-ppn` option to the `mpiexec_mpt` command otherwise you will see an error similar to: `mpiexec_mpt error: Need 36 processes but have only 1 left in PBS_NODEFILE`.

6.5. Running MPI parallel jobs
**Warning:** When using the `mpiexec_mpt` command, the `-ppn` option must come before the `-n` option otherwise you will see an error similar to: *MPT ERROR: Not enough slots from job scheduler for requested ranks.* (This applies to the default version of MPT and versions from 2.18 upwards.)

**Note:** If you are using an older version of MPT (2.17 or earlier), the `-n` option must come before the `-ppn` option when using the `mpiexec_mpt` command. If you get the options the wrong way around you will see an error similar to: *MPT ERROR: Not enough slots from job scheduler for requested ranks*

Please use `man mpiexec_mpt` query further options. (This is only available once you have loaded the `mpt` module.)

**HPE MPT: interactive MPI using `mpirun`**

If you want to run short interactive parallel applications (e.g. for debugging) then you can run HPE MPT compiled MPI applications on the login nodes using the `mpirun` command.

For instance, to run a simple, short 4-way MPI job on the login node, issue the following command (once you have loaded the appropriate modules):

```
mpirun -n 4 ./hello_mpi.x
```

**Note:** you should not run long, compute- or memory-intensive jobs on the login nodes. Any such processes are liable to termination by the system with no warning.

**HPE MPT: running hybrid MPI/OpenMP applications**

If you are running hybrid MPI/OpenMP code using HPE MPT you will also often make use of the `omplace` tool in your job launcher line. This tool takes the number of threads as the option `-nt`:

```
-nt [threads per parallel process] Specifies the number of cores for each parallel process to use for shared-memory threading. (This is in addition to the `OMP_NUM_THREADS` environment variable if you are using OpenMP for your shared memory programming.) The default on Cirrus is 1.
```

Please use `man mpiexec_mpt` and `man omplace` to query further options. (Again, these are only available once you have loaded the `mpt` module.)

**6.5.2 Intel MPI**

Intel MPI is accessed at runtime by loading the `intel-mpi-17`.

```
mplmod load intel-mpi-17
```

**Intel MPI: parallel job launcher `mpirun`**

The Intel MPI parallel job launcher on Cirrus is `mpirun`.

>This parallel job launcher is only available once you have loaded the `intel-mpi-17` module.

A sample MPI launch line using `mpirun` looks like:
mpirun -n 72 -ppn 36 ./my_mpi_executable.x arg1 arg2

This will start the parallel executable “my_mpi_executable.x” with arguments “arg1” and “arg2”. The job will be started using 72 MPI processes, with 36 MPI processes are placed on each compute node (this would use all the physical cores on each node). This would require 2 nodes to be requested in the PBS options.

The most important mpirun flags are:

- **-n [total number of MPI processes]** Specifies the total number of distributed memory parallel processes (not including shared-memory threads). For jobs that use all physical cores this will usually be a multiple of 36. The default on Cirrus is 1.

- **-ppn [parallel processes per node]** Specifies the number of distributed memory parallel processes per node. There is a choice of 1-36 for physical cores on Cirrus compute nodes (1-72 if you are using Hyper-Threading) If you are running with exclusive node usage, the most economic choice is always to run with “fully-packed” nodes on all physical cores if possible, i.e. \(-ppn 36\). Running “unpacked” or “underpopulated” (i.e. not using all the physical cores on a node) is useful if you need large amounts of memory per parallel process or you are using more than one shared-memory thread per parallel process.

Documentation on using Intel MPI (including mpirun) can be found online at:

- Intel MPI Documentation

**Intel MPI: running hybrid MPI/OpenMP applications**

If you are running hybrid MPI/OpenMP code using Intel MPI you need to set the `I_MPI_PIN_DOMAIN` environment variable to `omp` so that MPI tasks are pinned with enough space for OpenMP threads.

For example, in your job submission script you would use:

```bash
export I_MPI_PIN_DOMAIN=omp
```

You can then also use the `KMP_AFFINITY` environment variable to control placement of OpenMP threads. For more information, see:

- Intel OpenMP Thread Affinity Control

**Intel MPI: MPI-IO setup**

If you wish to use MPI-IO with Intel MPI you must set a couple of additional environment variables in your job submission script to tell the MPI library to use the Lustre file system interface. Specifically, you should add the lines:

```bash
export I_MPI_EXTRA_FILESYSTEM=on
export I_MPI_EXTRA_FILESYSTEM_LIST=lustre
```

after you have loaded the `intel-mpi-17` module.

If you fail to set these environment variables you may see errors such as:

```
This requires fcntl(2) to be implemented. As of 8/25/2011 it is not. Generic MPICH Message: File locking failed in
ADIOI_Set_lock(fd 0,cmd F_SETLKW/7,type F_WRLCK/1,whence 0) with return value
FFFFFFFF and errno 26.
- If the file system is NFS, you need to use NFS version 3, ensure that the lockd
demon is running on all the machines, and mount the directory with the 'noac'
option (no attribute caching).
```

(continues on next page)
- If the file system is LUSTRE, ensure that the directory is mounted with the 'flock' option.

ADIOI_Set_lock:: Function not implemented
ADIOI_Set_lock:offset 0, length 10
application called MPI_Abort(MPI_COMM_WORLD, 1) - process 3

6.5.3 OpenMPI

OpenMPI is accessed at runtime by loading the module openmpi. There are three OpenMPI modules currently installed:

module load openmpi/2.1.0
module load openmpi/3.1.4
module load openmpi/4.0.1

openmpi/2.1.0 is installed to be primarily used with Singularity. For user applications not using Singularity the newer versions of OpenMPI should be selected, with openmpi/4.0.1 being preferable.

OpenMPI: parallel job launcher mpirun

The OpenMPI parallel job launcher on Cirrus is mpirun.

.note :: This parallel job launcher is only available once you have loaded one of the OpenMPI modules.

A sample MPI launch line using mpirun looks like:

mpirun -n 72 -N 36 ./my_mpi_executable.x arg1 arg2

This will start the parallel executable “my_mpi_executable.x” with arguments “arg1” and “arg2”. The job will be started using 72 MPI processes, with 36 MPI processes are placed on each compute node (this would use all the physical cores on each node). This would require 2 nodes to be requested in the PBS options.

The most important mpirun flags are:

- **n [total number of MPI processes]** Specifies the total number of distributed memory parallel processes (not including shared-memory threads). For jobs that use all physical cores this will usually be a multiple of 36.

- **N [parallel processes per node]** Specifies the number of distributed memory parallel processes per node. There is a choice of 1-36 for physical cores on Cirrus compute nodes (1-72 if you are using Hyper-Threading) If you are running with exclusive node usage, the most economic choice is always to run with “fully-packed” nodes on all physical cores if possible, i.e. -N 36. Running “unpacked” or “underpopulated” (i.e. not using all the physical cores on a node) is useful if you need large amounts of memory per parallel process or you are using more than one shared-memory thread per parallel process.

Note, to use OpenMPI the PBS batch script used for running parallel jobs must include the `mpiprocs` keyword when specifying the number of nodes and processes to run, i.e. to run on 2 nodes using 36 process on each node (72 in total), the PBS select line would be:

```bash
#PBS -l select=2:ncpus=36:mpiprocs=36
```

Documentation on using OpenMPI (including mpirun) can be found online at:

- OpenMPI Documentation
6.6 Example parallel MPI job submission scripts

A subset of example job submission scripts are included in full below. The full set are available via the following links:

- **HPE MPT MPI Job**: example_mpi_hpempt.bash
- **Intel MPI Job**: example_mpi_impi.bash
- **HPE MPT Hybrid MPI/OpenMP Job**: example_hybrid_hpempt.bash
- **Intel MPI Hybrid MPI/OpenMP Job**: example_hybrid_impi.bash

### 6.6.1 Example: HPE MPT job submission script for MPI parallel job

A simple MPI job submission script to submit a job using 4 compute nodes (maximum of 144 physical cores) for 20 minutes would look like:

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N Example_MPI_Job
# Select 4 full nodes
#PBS -l select=4:ncpus=36
# Parallel jobs should always specify exclusive node access
#PBS -l place=scatter:excl
#PBS -l walltime=00:20:00
# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]
# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR
# Load any required modules
module load mpt
module load intel-compilers-17
# Set the number of threads to 1
# This prevents any threaded system libraries from automatically
# using threading.
export OMP_NUM_THREADS=1
# Launch the parallel job
# Using 144 MPI processes and 36 MPI processes per node
# '-ppn' option is required for all HPE MPT jobs otherwise you will get an error similar to:
# 'mpiexec_mpt error: Need 36 processes but have only 1 left in PBS_NODEFILE.'
# mpiexec_mpt -ppn 36 -n 144 ./my_mpi_executable.x arg1 arg2 > my_stdout.txt 2> my_stderr.txt
```

This will run your executable “my_mpi_executable.x” in parallel on 144 MPI processes using 2 nodes (36 cores per node, i.e. not using hyper-threading). PBS will allocate 4 nodes to your job and mpiexec_mpt will place 36 MPI processes on each node (one per physical core).

See above for a more detailed discussion of the different PBS options.
Warning: You must use the \texttt{-ppn} option when using HPE MPT otherwise you will see an error similar to: \texttt{mpiexec_mpt error: Need 36 processes but have only 1 left in PBS_NODEFILE.}

6.6.2 Example: HPE MPT job submission script for MPI+OpenMP (mixed mode) parallel job

Mixed mode codes that use both MPI (or another distributed memory parallel model) and OpenMP should take care to ensure that the shared memory portion of the process/thread placement does not span more than one node. This means that the number of shared memory threads should be a factor of 18.

In the example below, we are using 4 nodes for 6 hours. There are 4 MPI processes in total and 18 OpenMP threads per MPI process. Note the use of the \texttt{omplace} command to specify the number of threads.

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N Example_MixedMode_Job
# Select 4 full nodes
#PBS -l select=4:ncpus=36
# Parallel jobs should always specify exclusive node access
#PBS -l place=scatter:excl
#PBS -l walltime=6:0:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load mpt
module load intel-compilers-17

# Set the number of threads to 18
# There are 18 OpenMP threads per MPI process
export OMP_NUM_THREADS=18

# Launch the parallel job
# Using 8 MPI processes
# 2 MPI processes per node
# 18 OpenMP threads per MPI process
# '-ppn' option is required for all HPE MPT jobs otherwise you will get an error similar to: mpiexec_mpt error: Need 36 processes but have only 1 left in PBS_NODEFILE.'
mpiexec_mpt -ppn 2 -n 8 omplace -nt 18 ./my_mixed_executable.x arg1 arg2 > my_stdout.txt 2> my_stderr.txt
```

Warning: You must use the \texttt{-ppn} option when using HPE MPT otherwise you will see an error similar to: \texttt{mpiexec_mpt error: Need 36 processes but have only 1 left in PBS_NODEFILE.}
6.6.3 Example: OpenMPI job submission script for MPI parallel job

A simple MPI job submission script to submit a job using 4 compute nodes (maximum of 144 physical cores) for 20 minutes would look like:

```bash
#!/bin/bash --login

#PBS job options (name, compute nodes, job time)
#PBS -N Example_MPI_Job
# Select 4 full nodes
#PBS -l select=4:ncpus=36:mpiprocs=36
# Parallel jobs should always specify exclusive node access
#PBS -l place=scatter:excl
#PBS -l walltime=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load openmpi/4.0.1
module load intel-compilers-17

# Set the number of threads to 1
# This prevents any threaded system libraries from automatically
# using threading.
export OMP_NUM_THREADS=1

# Launch the parallel job
# Using 144 MPI processes and 36 MPI processes per node
mpirun --mca pml ucx --mca btl ^openib -N 36 -n 144 ./my_mpi_executable.x arg1 arg2 > my_stdout.txt 2> my_stderr.txt
```

This will run your executable “my_mpi_executable.x” in parallel on 144 MPI processes using 2 nodes (36 cores per node, i.e. not using hyper-threading). PBS will allocate 4 nodes to your job and mpirun will place 36 MPI processes on each node (one per physical core).

Note the `--mca pml ucx --mca btl ^openib` part of the command above is only required for OpenMPI version 4.0.1. It is not required for the older versions of OpenMPI installed on ARCHER.

6.6.4 Example: job submission script for parallel non-MPI based jobs

If you want to run on multiple nodes, where each node is running a self-contained job, not using MPI (e.g.) for processing data or a parameter sweep, you can use the HPE MPT mpiexec_mpt launcher to control job placement.

In the example script below, `work.bash` is a bash script which runs a threaded executable with a command-line input and `perf.bash` is a bash script which copies data from the CPU performance counters to an output file. As both handle the threading themselves, it is sufficient to allocate 1 MPI rank. Using the ampersand `&` allows both to execute simultaneously. Both `work.bash` and `perf.bash` run on 4 nodes.

```bash
#!/bin/bash --login

#PBS job options (name, compute nodes, job time)
#PBS -N Example_MixedMode_Job

# Load any required modules
module load openmpi/4.0.1
module load intel-compilers-17

# Set the number of threads to 1
# This prevents any threaded system libraries from automatically
# using threading.
export OMP_NUM_THREADS=1

# Launch the parallel job
# Using 144 MPI processes and 36 MPI processes per node
mpirun --mca pml ucx --mca btl ^openib -N 36 -n 144 ./my_mpi_executable.x arg1 arg2 > my_stdout.txt 2> my_stderr.txt
```

(continues on next page)
# Select 4 full nodes
#PBS -l select=4:ncpus=36
# Parallel jobs should always specify exclusive node access
#PBS -l place=scatter:excl
#PBS -l walltime=6:0:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load mpt

# Set this variable to inform mpiexec_mpt these are not MPI jobs
export MPI_SHEPHERD=true

# Execute work and perf scripts on nodes simultaneously.
mpiexec_mpt -ppn 1 -n 4 work.bash &
mpiexec_mpt -ppn 1 -n 4 perf.bash &
wait

.note :: The `wait` command is required to stop the PBS job finishing before the scripts finish. If you find odd behaviour, especially with respect to the values of bash variables, double check you have set `MPI_SHEPHERD=true`

## 6.7 Serial Jobs

Serial jobs are setup in a similar way to parallel jobs on Cirrus. The only changes are:

1. You should request a single core with `select=1:ncpus=1`
2. You will not need to use a parallel job launcher to run your executable

A simple serial script to compress a file would be:

```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N Example_Serial_Job
#PBS -l select=1:ncpus=1
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load intel-compilers-16

# Set the number of threads to 1 to ensure serial
export OMP_NUM_THREADS=1
```
# Run the serial executable
gzip my_big_file.dat

## 6.8 Job arrays

The PBSPro job scheduling system offers the *job array* concept, for running collections of almost-identical jobs, for example running the same program several times with different arguments or input data.

Each job in a job array is called a *subjob*. The subjobs of a job array can be submitted and queried as a unit, making it easier and cleaner to handle the full set, compared to individual jobs.

All subjobs in a job array are started by running the same job script. The job script also contains information on the number of jobs to be started, and PBSPro provides a subjob index which can be passed to the individual subjobs or used to select the input data per subjob.

### 6.8.1 Job script for a job array

As an example, to start 56 subjobs, with the subjob index as the only argument, and 4 hours maximum runtime per subjob, save the following content into the file `job_script.pbs`:

```
#!/bin/bash --login
#PBS -l select=1:ncpus=1
#PBS -l walltime=04:00:00
#PBS -J 1-56
#PBS -q workq
#PBS -V

cd ${PBS_O_WORKDIR}
/path/to/exe $PBS_ARRAY_INDEX
```

Another example of a job script for submitting a job array is given [here](#).

### 6.8.2 Starting a job array

When starting a job array, most options can be included in the job file, but the project code for the resource billing has to be specified on the command line:

```
qsub -A [project code] job_script.pbs
```

### 6.8.3 Querying a job array

In the normal PBSPro job status, a job array will be shown as a single line:

```
> qstat
<table>
<thead>
<tr>
<th>Job id</th>
<th>Name</th>
<th>User</th>
<th>Time</th>
<th>Use</th>
<th>S</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>112452</td>
<td>.indy2-lo</td>
<td>user1</td>
<td>0 B</td>
<td>B</td>
<td></td>
<td>workq</td>
</tr>
</tbody>
</table>
```

To monitor the subjobs of the job 112452, use
6.9 Interactive Jobs

When you are developing or debugging code you often want to run many short jobs with a small amount of editing the code between runs. This can be achieved by using the login nodes to run MPI but you may want to test on the compute nodes (e.g. you may want to test running on multiple nodes across the high performance interconnect). One of the best ways to achieve this on Cirrus is to use interactive jobs.

An interactive job allows you to issue `mpirun_mpt` commands directly from the command line without using a job submission script, and to see the output from your program directly in the terminal.

To submit a request for an interactive job reserving 8 nodes (288 physical cores) for 1 hour you would issue the following `qsub` command from the command line:

```
qsub -IVl select=8:ncpus=36,walltime=1:0:0,place=scatter:excl -A [project code]
```

When you submit this job your terminal will display something like:

```
qsub: waiting for job 19366.indy2-login0 to start
```

It may take some time for your interactive job to start. Once it runs you will enter a standard interactive terminal session. Whilst the interactive session lasts you will be able to run parallel jobs on the compute nodes by issuing the `mpirun_mpt` command directly at your command prompt (remember you will need to load the `mpt` module and any compiler modules before running) using the same syntax as you would inside a job script. The maximum number of cores you can use is limited by the value of `select` you specify when you submit a request for the interactive job.

If you know you will be doing a lot of intensive debugging you may find it useful to request an interactive session lasting the expected length of your working session, say a full day.

Your session will end when you hit the requested walltime. If you wish to finish before this you should use the `exit` command.

6.10 Reservations

Resource reservations are available on Cirrus. These allow users to reserve a number of nodes for a specified length of time starting at a particular time on the system.

Examples of the reasons for using reservations could be:

- An exceptional job requires longer than 96 hours runtime.
- You require a job/jobs to run at a particular time e.g. for a demonstration or course.
Warning: For multi-node jobs we strongly recommend requesting a reservation two nodes larger than the size you want to stop the reservation failing if a node crashes. This is particularly important if the reservation involves long jobs or those of a time critical nature.

Note: Reservations will be charged at 1.5 times the usual rate and you will be charged the full rate for the entire reservation whether or not you use the resources reserved for the full time. In addition, you will not be refunded the resources if you fail to use them due to a job crash unless this crash is due to a system failure. To allow people to create multi-node reservations, we will charge at number of nodes - 2 for reservations (with a minimum of 2 nodes charged).

### 6.10.1 Requesting reservations

You request a reservation on Cirrus using PBS from the command line. Before requesting the reservation, you will need the following information:

- The start time for the reservation
- The duration of the reservation
- The number of cores (or nodes for multi-node, node-exclusive jobs)
- The project ID you wish to charge the reservation to

You use the `pbs_rsub` command to create a reservation. This command has a similar syntax to the `qsub` command for requesting resources but takes the additional parameters `-R` (to specify the reservation start time); `-D` (to specify the reservation duration); and `-G` (to specify the project ID to charge the reservation to). For example, to create a reservation for 3 hours at 10:30 (UK time) on Saturday 26 August 2017 for 4 full nodes (144 physical cores, 288 hyperthreads) and charge to project “t01” you would use the command:

```
pbs_rsub -R 1708261030 -D 3:0:0 -l select=6:ncpus=36,place=scatter:excl -G +t01
```

Generating response:

```
R122604.indy2-login0 UNCONFIRMED
```

The command will return a reservation ID (R122604 in the example above) and note that it is currently UNCONFIRMED. PBSPro will change the status to CONFIRMED once it has checked that it is possible to schedule the reservation. Note that we requested 6 nodes rather than the required 4 to reduce the risk of hardware failure affecting the reservation.

Note: Only the user that requested this reservation will be able to submit jobs to it. To create a reservation that is available to all users in a particular project, see the instructions below.

There are many other options to the `pbs_rsub` command. Please check the man page for a full description.

### 6.10.2 Checking the status of your reservation

You can check the status of your reservation request with the `pbs_rstat` command:

```
pbs_rstat
```

Which will generate a response:
and, as you can see, the status of the requested reservation is now CO (CONFIRMED).

### 6.10.3 Submitting jobs to a reservation

You submit jobs to reservations in the same way as you do for all other jobs using the `qsub` command. The only additional information required is to specify the reservation ID to the `-q` option. For example, to submit to the reservation created above you would use:

```
qsub -q R122604 ...usual qsub options/job script name...
```

**Note:** You can submit jobs to the reservation ahead of the start time and the job will start as soon as the reservation begins.

### 6.10.4 Reservations for all project users

By default, a reservation will only be available to the user who requested it. If you wish to create a reservation that is usable by all members of your project you need to modify the user permissions using the `-U` option.

For example, to create a reservation for 192 hours, starting at 16:15 (UK time) on Monday 18 September 2017 for 64 nodes accessible by all users in the t01 project you would use:

```
pbs_rsub -R 1709181615 -D 192:0:0 -l select=66:ncpus=36,place=scatter:excl -G +t01 -U+
```

Generating a response:

```
R122605.indy2-login0 UNCONFIRMED
```

Here, the `-G +t01` option charges the reservation to the t01 project and restricts access to users in the t01 project; the `-U +` option allows all users (in the t01 project) access to the reservation. Note that, as above, we created the reservation with 66 nodes instead of the required 64 to reduce the risk of hardware failures affecting the reservation.

**Note:** You can restrict access to specific users within a project, see the `pbs_rsub` man page for more information on how to do this.

### 6.10.5 Deleting a reservation

Use the `pbs_rdel` command to delete a reservation:

```
[auser@cirrus-login0 ~]$ pbs_rdel R122605
```
CHAPTER 7

Singularity Containers

This page was originally based on the documentation at the University of Sheffield HPC service.

Designed around the notion of mobility of compute and reproducible science, Singularity enables users to have full control of their operating system environment. This means that a non-privileged user can “swap out” the Linux operating system and environment on the host for a Linux OS and environment that they control. So if the host system is running CentOS Linux but your application runs in Ubuntu Linux with a particular software stack; you can create an Ubuntu image, install your software into that image, copy the image to another host (e.g. Cirrus), and run your application on that host in its native Ubuntu environment.

Singularity also allows you to leverage the resources of whatever host you are on. This includes high-speed interconnects (i.e. Infinband on Cirrus), file systems (i.e. /lustre on Cirrus) and potentially other resources (e.g. the licensed Intel compilers on Cirrus).

Note: Singularity only supports Linux containers. You cannot create images that use Windows or macOS (this is a restriction of the containerisation model rather than Singularity).

7.1 Useful Links

- Singularity website
- Singularity documentation

7.2 About Singularity Containers (Images)

Similar to Docker, a Singularity container (or, more commonly, image) is a self-contained software stack. As Singularity does not require a root-level daemon to run its images (as is required by Docker) it is suitable for use on a multi-user HPC system such as Cirrus. Within the container/image, you have exactly the same permissions as you do in a standard login session on the system.

In practice, this means that an image created on your local machine with all your research software installed for local development will also run on Cirrus.
Pre-built images (such as those on DockerHub or SingularityHub) can simply be downloaded and used on Cirrus (or anywhere else Singularity is installed); see Using Singularity Images on Cirrus.

Creating and modifying images requires root permission and so must be done on a system where you have such access (in practice, this is usually within a virtual machine on your laptop/workstation); see Creating Your Own Singularity Images.

7.3 Using Singularity Images on Cirrus

Singularity images can be used on Cirrus in a number of ways, including:

- Interactively on the login nodes
- Interactively on compute nodes
- As serial processes within a non-interactive batch script
- As parallel processes within a non-interactive batch script (not yet documented)

We provide information on each of these scenarios (apart from the parallel use where we are still preparing the documentation) below. First, we describe briefly how to get existing images onto Cirrus so you can use them.

7.3.1 Getting existing images onto Cirrus

Singularity images are simply files so, if you already have an image file, you can use scp to copy the file to Cirrus as you would with any other file.

If you wish to get a file from one of the container image repositories then Singularity allows you to do this from Cirrus itself.

This functionality requires tools that are not part of the standard OS on Cirrus so we have provided a Singularity image that allows you to build images from remote repositories (i.e. you use a Singularity image to build Singularity images!).

For example, to retrieve an image from DockerHub on Cirrus we first need to enter an interactive session in the image we provide for building Singularity images:

```
[user@cirrus-login0 ~]$ module load singularity
[user@cirrus-login0 ~]$ singularity exec $CIRRUS_SIMG/cirrus-sbuild.simg /bin/bash --login

Singularity>
```

This invokes a login bash shell within the $CIRRUS_SIMG/cirrus-sbuild.simg image as indicated by our prompt change. (We need a login shell to allow module commands to work within the image.)

Now we are in the image we can load the singularity module (to get access to the Singularity commands) and pull an image from DockerHub:

```
Singularity> module load singularity
Singularity> singularity build lolcow.simg docker://godlovedc/lolcow
Docker image path: index.docker.io/godlovedc/lolcow:latest
Cache folder set to /lustre/home/t01/user/.singularity/docker
Importing: base Singularity environment
Importing: /lustre/home/t01/user/.singularity/docker/ →sha256:9fb6c798fa41e509b58bccc5c29654c3ff4648b608f5daa67c1a8e7d02c118.tar.gz
Importing: /lustre/home/t01/user/.singularity/docker/ →sha256:3b61feb4aeefe982e0cb9c696d415137384d1a01052b50a85e46439e15e49a.tar.gz
```

(continues on next page)
The first argument to `singularity build` (lolcow.simg) specifies a path and name for your container. The second argument (docker://godlovecd/lolcow) gives the DockerHub URI from which to download the image.

Now we can exit the image and run our new image we have just built on the Cirrus login node:

```
[user@cirrus-login0 ~]$ singularity run lolcow.simg
```

This image contains a `runscript` that tells Singularity what to do if we run the image. We demonstrate different ways to use images below.

Similar syntax can be used for Singularity Hub. For more information see the Singularity documentation:

- **Build a Container**

### 7.3.2 Interactive use on the login nodes

Once you have an image file, using it on the login nodes in an interactive way is extremely simple: you use the `singularity shell` command. Using the image we built in the example above:

```
[user@cirrus-login0 ~]$ module load singularity
[user@cirrus-login0 ~]$ singularity shell lolcow.simg
Singularity lolcow.simg:~>
```

Within a Singularity image your home directory will be available. The directory with centrally-installed software (/lustre/sw) is also available in images by default. Note that the `module` command will not work in images unless you have installed he required software and configured the environment correctly; we describe how to do this below.

Once you have finished using your image, you return to the Cirrus login node command line with the `exit` command:

```
Singularity lolcow.simg:~> exit
exit
[user@cirrus-login0 ~]$
```

### 7.3.3 Interactive use on the compute nodes

The process for using an image interactively on the compute nodes is very similar to that for using them on the login nodes. The only difference is that you have to submit an interactive serial job to get interactive access to the compute
node first.

For example, to reserve a full node for you to work on interactively you would use:

```
[ user@cirrus-login0 ~ ] $ qsub -IVl select=1:ncpus=36,walltime=0:20:0,
        ->
        place=scatter:excl -A t01

qsub: waiting for job 234192.indy2-login0 to start

...wait until job starts...

qsub: job 234192.indy2-login0 ready

[ user@rli2n13 ~ ] $
```

Note the prompt has changed to show you are on a compute node. Now you can use the image in the same way as on the login node.

```
[ user@rli2n13 ~ ] $ module load singularity
[ user@rli2n13 ~ ] $ singularity shell lolcow.simg

Singularity: Invoking an interactive shell within container...

Singularity lolcow.simg:~> exit
exit

[ user@rli2n13 ~ ] $ exit

[ user@cirrus-login0 ~ ] $
```

Note we used `exit` to leave the interactive image shell and then `exit` again to leave the interactive job on the compute node.

### 7.3.4 Serial processes within a non-interactive batch script

You can also use Singularity images within a non-interactive batch script as you would any other command. If your image contains a `runscript` then you can use `singularity run` to execute the runscript in the job. You can also use `singularity exec` to execute arbitrary commands (or scripts) within the image.

An example job submission script to run a serial job that executes the runscript within the `lolcow.simg` we built above on Cirrus would be:

```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N simg_test
#PBS -l select=1:ncpus=1
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load singularity

# Run the serial executable
singularity run $HOME/lolcow.simg
```

You submit this in the usual way and the output would be in the STDOUT/STDERR files in the usual way.
7.4 Creating Your Own Singularity Images

As we saw above, you can create Singularity images by importing from DockerHub or Singularity Hub on Cirrus itself. If you wish to create your own custom image then you must install Singularity on a system where you have root (or administrator) privileges - often your own laptop or workstation.

We provide links below to instructions on how to install Singularity locally and then cover what options you need to include in a Singularity recipe file to create images that can run on Cirrus and access the software development modules. (This can be useful if you want to create a custom environment but still want to compile and link against libraries that you only have access to on Cirrus such as the Intel compilers, HPE MPI libraries, etc.)

7.4.1 Installing Singularity on Your Local Machine

You will need Singularity installed on your machine in order to locally run, create and modify images. How you install Singularity on your laptop/workstation depends on the operating system you are using.

If you are using Windows or macOS, the simplest solution is to use Vagrant to give you an easy to use virtual environment with Linux and Singularity installed. The Singularity website has instructions on how to use this method to install Singularity:

- Installing Singularity on macOS with Vagrant
- Installing Singularity on Windows with Vagrant

If you are using Linux then you can usually install Singularity directly, see:

- Installing Singularity on Linux

7.4.2 Singularity Recipes to Access modules on Cirrus

You may want your custom image to be able to access the modules environment on Cirrus so you can make use of custom software that you cannot access elsewhere. We demonstrate how to do this for a CentOS 7 image but the steps are easily translated for other flavours of Linux.

For the Cirrus modules to be available in your Singularity container you need to ensure that the environment-modules package is installed in your image.

In addition, when you use the container you must invoke access as a login shell to have access to the module commands.

Here is an example recipe file to build a CentOS 7 image with access to TCL modules already installed on Cirrus:

BootStrap: docker
From: centos:centos7

%post
    yum update -y
    yum install environment-modules -y

If we save this recipe to a file called cirrus-mods.def then we can use the following command to build this image (remember this command must be run on a system where you have root access, not Cirrus):

me@my-system:---> sudo singularity build cirrus-mods.simg cirrus-mods.def

The resulting image file (cirrus-mods.simg) can then be copied to Cirrus using scp.

When you use the image interactively on Cirrus you must start with a login shell, i.e.:
[user@cirrus-login0 ~]$ module load singularity
[user@cirrus-login0 ~]$ singularity exec cirrus-mods.simg /bin/bash --login
Singularity> module avail intel-compilers

------------------------- /lustre/sw/modulefiles ---------------------
intel-compilers-16/16.0.2.181
intel-compilers-16/16.0.3.210 (default)
intel-compilers-17/17.0.2.174 (default)
Python on Cirrus is provided by the Anaconda distribution. Both Python 2 and Python 3 versions of the distributions are supported.

The central installation provides many of the most common packages used for scientific computation and data analysis. If the packages you require are not included in the central Anaconda Python distribution, then the simplest way to make these available is often to install your own version of Miniconda and add the packages you need. We provide instructions on how to do this below. An alternative way to provide your own packages (and to make them available more generally to other people in your project and beyond) would be to use a Singularity container, see the Singularity Containers chapter of this User Guide for more information on this topic.

### 8.1 Accessing central Anaconda Python

Users have the standard system Python available by default. To setup your environment to use the Anaconda distributions you should use:

```
module load anaconda/python2
```

for Python 2, or:

```
module load anaconda/python3
```

for Python 3.

You can verify the current version of Python with:

```
[user@cirrus-login0 ~]$ module load anaconda/python2
[user@cirrus-login0 ~]$ python --version
Python 2.7.12 :: Anaconda 4.2.0 (64-bit)
```

Full details on the Anaconda distributions can be found on the Continuum website at:

- [http://docs.continuum.io/anaconda/index.html](http://docs.continuum.io/anaconda/index.html)
### 8.1.1 Packages included in Anaconda distributions

You can list the packages currently available in the distribution you have loaded with the command `conda list`:

```
[user@cirrus-login0 ~]$ module load anaconda
[user@cirrus-login0 ~]$ conda list
# packages in environment at /lustre/sw/anaconda/anaconda2:
#
  _license 1.1 py27_1
  alabaster 0.7.10 py27_0
  anaconda custom py27_0
  anaconda-client 1.6.3 py27_0
  anaconda-navigator 1.2.3 py27_0
  anaconda-project 0.6.0 py27_0
  argcomplete 1.0.0 py27_1
  asn1crypto 0.22.0 py27_0
  astroid 1.4.9 py27_0
  ...
```

### 8.1.2 Adding packages to the Anaconda distribution

Adding packages to the central Anaconda distribution cannot be done by users. If you wish to have additional packages, we recommend installing your own local version of Miniconda and adding the packages you need. This approach is described in Custom Environment below.

### 8.2 Custom Environments

To setup a custom Python environment including packages that are not in the central installation, the simplest approach is the install Miniconda locally in your own directories.

#### 8.2.1 Installing Miniconda

First, you should download Miniconda. You can use `wget` to do this, for example:

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

You can find links to the various miniconda versions on the Miniconda website:

- https://conda.io/miniconda.html

For Cirrus, you should use the Linux 64-bit (bash installer).

Once you have downloaded the installer, you can run it. For example:

```
[user@cirrus-login0 ~]$ bash Miniconda3-latest-Linux-x86_64.sh
Welcome to Miniconda3 4.3.31
In order to continue the installation process, please review the license agreement.
Please, press ENTER to continue
```
Do you accept the license terms? [yes/no]
[no] >>> yes

Miniconda3 will now be installed into this location:
/lustre/home/t01/user/miniconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/lustre/home/t01/user/miniconda3] >>>
PREFIX=/lustre/home/t01/user/miniconda3
installing: python-3.6.3-h6c0c0dc5 ...
installing: ca-certificates-2017.08.26-h1d4fec50 ...
installing: conda-env-2.6.0-h36134e31 ...
installing: libgcc-ng-7.2.0-h7cc24e2 ...
installing: libstdcxx-ng-7.2.0-h7a57d052 ...
installing: libffi-3.2.1-hd88cf554 ...
installing: ncurses-6.0-h9df7e312 ...
installing: openssl-1.0.2n-hb7f436b0 ...
installing: tk-8.6.7-hc7452773 ...
installing: xz-5.2.3-h55aa19d2 ...
installing: yaml-0.1.7-had098182 ...
installing: zlib-1.2.11-ha838bed2 ...
installing: libedit-3.1-heed36240 ...
installing: readline-7.0-ha6073c64 ...
installing: sqlite-3.20.1-hb8981582 ...
installing: asn1crypto-0.23.0-py36h46393420 ...
installing: certifi-2017.11.5-py36hf29ccca0 ...
installing: charset-3.0.4-py36h0f667ec1 ...
installing: idna-2.6-py36h82fb2a81 ...
installing: pycosat-0.6.3-py36h0a5515d0 ...
installing: pycparser-2.18-py36hf9f622e1 ...
installing: pysocks-1.6.7-py36hd97a5b11 ...
installing: ruamel_yaml-0.11.14-py36ha2fb22d2 ...
installing: six-1.11.0-py36h372c4331 ...
installing: cffi-1.11.2-py36h28250820 ...
installing: setuptools-36.5.0-py36he42e2e10 ...
installing: cryptography-2.1.4-py36hd09be540 ...
installing: wheel-0.30.0-py36hfd4bb980 ...
installing: pip-9.0.1-py36h6c6f99ce4 ...
installing: requests-2.18.4-py36he2e5f8d1 ...
installing: conda-4.3.31-py360 ...
installation finished.
WARNING:
You currently have a PYTHONPATH environment variable set. This may cause unexpected behavior when running the Python interpreter in Miniconda3. For best results, please verify that your PYTHONPATH only points to directories of packages that are compatible with the Python interpreter.
Miniconda is now installed in your local directories but we still need to setup a way to access it correctly. There are a number of ways to do this.

- If you are always going to be using this Python environment on ARCHER and do not wish to use any other Python environment, you can follow the advice of the Miniconda installer and add a line to your .bashrc file:

  ```bash
  export PATH=/lustre/home/t01/user/miniconda3/bin:$PATH
  ```

- You can export PATH every time you wish to use you local install by using the bash command 

  ```bash
  export PATH=/lustre/home/t01/user/miniconda3/bin:$PATH
  ```

  (using the correct PATH as specified by the installer). This will become tedious if you use the environment often!

- You can create an alias in your .bashrc file to set the path. For example, adding the line

  ```bash
  alias condasetup="export PATH=/lustre/home/t01/user/miniconda3/bin:$PATH"
  ```

  would allow you to use the command condasetup to initialise the Miniconda environment.

- You could also create a modulefile to provide a way to initialise the environment using module load ...

  as we do for our Anaconda environments. Please contact the helpdesk if you want help to do this.

### 8.2.2 Installing packages into Miniconda

Once you have installed Miniconda and setup your environment to access it, you can then add whatever packages you wish to the installation using the `conda install ...` command. For example:

```bash
[user@cirrus-login0 ~]$ conda install numpy
```

```
Fetching package metadata ...............  
Solving package specifications: .      
```

```
Package plan for installation in environment /lustre/home/t01/user/miniconda3: 
```

The following NEW packages will be INSTALLED:

- blas: 1.1-openblas conda-forge
- libgfortran: 3.0.0-1
- numpy: 1.14.0-py36_blas_openblas_200 conda-forge [blas_openblas]
- openblas: 0.2.20-7 conda-forge

The following packages will be UPDATED:

- conda: 4.3.31-py36_0 --> 4.3.33-py36_0 conda-forge

The following packages will be SUPERSEDED by a higher-priority channel:

- conda-env: 2.6.0-h36134e3_1 --> 2.6.0-0 conda-forge

(continues on next page)
Proceed ([y]/n)? y

conda-env-2.6. 100% |#################################################################
libgfortran-3. 100% |#################################################################
openblas-0.2.2 100% |#################################################################
blas-1.1-openbl 100% |#################################################################
numpy-1.14.0-p 100% |#################################################################
conda-4.3.33-p 100% |#################################################################
Here we see the numpy module has been installed in the local environment:

```
[user@cirrus-login0 ~]$ conda list
# packages in environment at /lustre/home/t01/user/miniconda3:
#
asn1crypto 0.23.0 py36h4639342_0
blas 1.1 openblas conda-forge
cacertificates 2017.08.26 h1d4fec5_0
certifi 2017.11.5 py36hf29ccca_0
cffi 1.11.2 py36h2825082_0
chardet 3.0.4 py36h0f667ec_1
conda 4.3.33 py36_0 conda-forge
conda-env 2.6.0 0 conda-forge
cryptography 2.1.4 py36hd09be54_0
idna 2.6 py36h82fb2a8_1
libedit 3.1 heed3624_0
libffi 3.2.1 hd88cf55_4
libgcc-ng 7.2.0 h7a57d05_2
libgfortran 3.0.0 1
libstdcxx-ng 7.2.0 h7a57d05_2
ncurses 6.0 h9df7e31_2
numpy 1.14.0 py36_blas_openblas_200 [blas_openblas]

--conda-forge
openblas 0.2.20 py36hf436b_0
openssl 1.0.2n hb7f436b_0
pyscipy 0.6.3 py36h0a5515d_0
pycparser 2.18 py36hf9f622e_1
pyopenssl 17.5.0 py36hd09be54_0
pysocks 1.6.7 py36hd97a5b1_1
python 3.6.3 h6c0c0dc_5
readline 7.0 ha6073c6_4
requests 2.18.4 py36he2e5f8d_1
ruamel_yaml 0.11.14 py36ha2f22d_2
setuptools 36.5.0 py36he2e5f8d_1
six 1.11.0 py36h0a5515d_0
sqlite 3.20.1 hbb898158_2
tk 8.6.7 hbc745277_3
urllib3 1.22 py36h82fb2a8_1
wheel 0.29.0 py36h9f622e_1
xz 5.2.3 h55aa19d_2
yaml 0.1.7 had09818_2
zlib 1.2.11 ha838bed_2
```

8.2. Custom Environments
Please note, for some package installations it may also be necessary to specify a channel such as conda-forge. For example, the following command installs the pygobject module.

[`
[user@cirrus-login0 ~]$ conda install -c conda-forge pygobject
`]

56 Chapter 8. Using Python
Using the Cirrus GPU Nodes

Cirrus has two compute nodes equipped with GPGPU accelerators. This section of the user guide explains how to compile code and submit jobs to the GPU nodes.

Note: The GPU accelerators on Cirrus are only available in TCC (Tesla Compute Cluster) mode and so do not support graphics rendering tasks, only computational tasks.

9.1 Hardware details

The Cirrus GPU compute nodes each contain two 2.4 GHz, 20-core Intel Xeon Gold 6148 (Skylake) series processors. Each of the cores in these processors support 2 hardware threads (Hyperthreads), which are enabled by default. The nodes also each contain four NVIDIA Tesla V100-SXM2-16GB (Volta) GPU accelerators connected to the host processors and each other via PCIe.

The GPU compute nodes on Cirrus have 384 GB of main memory shared between the two processors. The memory is arranged in a non-uniform access (NUMA) form: each 20-core processor is a single NUMA region with local memory of 192 GB. Access to the local memory by cores within a NUMA region has a lower latency and higher bandwidth than accessing memory on the other NUMA region.

There are three levels of cache, configured as follows:

- L1 Cache 32 KiB Instr., 32 KiB Data (per core)
- L2 Cache 1 GiB (per core)
- L3 Cache GiB MB (shared)

Each GPU accelerator has 16 GiB of fast GPU memory.

There are 2 GPU compute nodes on Cirrus giving a total of 80 CPU cores and 8 GPU accelerators.
9.2 Compiling software for the GPU nodes

Note: As the Cirrus login nodes use Intel Xeon Broadwell processors and the GPU compute nodes are equipped with Intel Xeon Sylke processors additional flags are needed to compile code for the correct processors. These flags are described in the different compiler suites below.

9.2.1 CUDA

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

To use the CUDA toolkit on Cirrus, you should load the cuda module:

```
module load cuda
```

Once you have loaded the cuda module, you can access the CUDA compiler with the `nvcc` command.

As well as the CUDA compiler, you will also need a compiler module to support compilation of the host (CPU) code. The CUDA toolkit supports both GCC and Intel compilers. You should load your chosen compiler module before you compile.

Note: The `nvcc` compiler currently supports versions of GCC up to 6.x and versions of the Intel compilers up to 17.x.

Using CUDA with GCC

By default, `nvcc` will use the system version of GCC. We recommend that you load a more recent version of GCC than the system default to support the CUDA compiler, e.g.

```
module load gcc/6.3.0
```

Note: GCC 6.x is the latest version of the GCC compiler supported by `nvcc`.

You can now use `nvcc` to compile your source code, e.g.:

```
nvcc -o cuda_test.x cuda_test.cu
```

Note: When compiling using GCC for the CPUs on the GPU compute nodes you should add the flag `-march=skylake-avx512` to get the correct instructions for the Skylake processors.

Using CUDA with Intel compilers

You should load either the Intel 16 or Intel 17 compilers to use with `nvcc`. We recommend the Intel 17 compilers, you also need the `gcc` module to provide C++ support:
module load intel-compilers-17
module load gcc/6.3.0

Note: Intel 17 is the latest version of the Intel compilers supported by nvcc.

You can now use nvcc -ccbin icpc to compile your source code, e.g.:

```
nvcc -ccbin icpc -o cuda_test.x cuda_test.cu
```

The -ccbin icpc tells nvcc to use the Intel C++ compiler to compile the host (CPU) code.

Note: When compiling using Intel compilers for the CPUs on the GPU compute nodes you should add the flag
-xCore-AVX512 -qopt-zmm-usage=high to get the correct instructions for the Skylake processors.

9.3 Submitting jobs to the GPU nodes

Two additional options are needed in GPU job submission scripts over those in standard jobs:

- `-q gpu` This option is required to submit the job to the `gpu` queue on Cirrus
- `ngpus=N` (where N is the number of GPU accelerators you wish to use). This resource request needs to be added to your select statement

Note: We generally recommend that you should request 10 CPU cores per GPU accelerator even if you do not need them.

9.3.1 Job submission script using single GPU on a single node

A job script that required 1 GPU accelerator and 10 CPU cores for 20 minutes could look like:

```
#!/bin/bash
#
#PBS -N cuda_test
#PBS -q gpu
#PBS -l select=1:ncpus=10:ngpus=1
#PBS -l walltime=0:20:0
# Budget: change 't01' to your budget code
#PBS -A t01

# Load the required modules (this assumes you compiled with GCC 6.3.0)
module load cuda
module load gcc/6.3.0

cd $PBS_O_WORKDIR
./cuda_test.x
```

The line #PBS -l select=1:ncpus=10:ngpus=1 requests 1 node, 10 cores on that node and 1 GPU accelerator on that node.
9.3.2 Job submission script using multiple GPUs on a single node

Note: Remember that there are a maximum of 4 GPU accelerators per node and a maximum of 40 CPU cores per node.

A job script that required 4 GPU accelerators and 40 CPU cores for 20 minutes could look like:

```bash
#!/bin/bash
#
#PBS -N cuda_test
#PBS -q gpu
#PBS -l select=1:ncpus=40:ngpus=4
#PBS -l walltime=0:20:0
# Budget: change 't01' to your budget code
#PBS -A t01

# Load the required modules (this assumes you compiled with GCC 6.3.0)
module load cuda
module load gcc/6.3.0

#PBS -l select=1:ncpus=40:ngpus=4
requests 1 node, 40 cores on that node and 4 GPU accelerators on that node (i.e. a full GPU compute node).

```

9.3.3 Job submission script using multiple GPUs on multiple nodes

Note: Remember that there are a maximum of 4 GPU accelerators per node and a maximum of 40 CPU cores per node.

A job script that required 8 GPU accelerators and 80 CPU cores for 20 minutes across 2 nodes could look like:

```bash
#!/bin/bash
#
#PBS -N cuda_test
#PBS -q gpu
#PBS -l select=2:ncpus=40:ngpus=4
#PBS -l walltime=0:20:0
# Budget: change 't01' to your budget code
#PBS -A t01

# Load the required modules (this assumes you compiled with GCC 6.3.0)
module load cuda
module load gcc/6.3.0
module load mpt

#PBS -l select=2:ncpus=40:ngpus=4
requests 1 node, 40 cores on that node and 4 GPU accelerators on that node (i.e. a full GPU compute node).

```

```bash
cd $PBS_O_WORKDIR
./cuda_test.x
```

```bash
mpirun -ppn 40 -n 80 ./cuda_test.x
```
The line `#PBS -l select=2:ncpus=40:ngpus=4` requests 2 nodes, 40 cores per node (80 in total) and 4 GPU accelerators per node (8 in total).
In addition to the lustre file-system Cirrus also has access to a high-capacity, object store system. This web service provides an additional place for you to store your data but it works in a different way from the file system. Normally you would not access the object store directly from within your programs but it is a good place to archive data to free up space for new calculations. The object store uses the same API as the Amazon S3 object store so many compatible clients and tools are available.

- Unlike files, objects cannot be modified or appended to. They are uploaded and downloaded as complete objects. However it is possible to replace an Object with an entirely new version.
- The object store can be accessed from anywhere with an internet connection, not just Cirrus.

**Note:** If you would like access to the object store for your project, please contact the Cirrus helpdesk: support@cirrus.ac.uk

### 10.1 Access Keys

Object store access permissions and storage quotas are based on AccessKeys. An access key consists of two parts:

1. The name of the AccessKey
2. A corresponding AccessSecret

There is also a UUID that can be used as a unique identifier for the key.

For example:

- **AccessKey** AKIA74IP98S48W5D9EPR
- **AccessSecret** xKvB51XB3gbP47T+TlkRT+DUf98BY20io0nkV9q
- **UUID** 3a36a0fef03518827ca41992e934850b8bbf7c28

These are a little bit like a randomly generated Username/Password pair which makes them difficult to guess but as you will need to store the secret in tools and scripts care needs to be taken to ensure that they are kept secret.
10.2 Buckets

Objects in the store are organised in collections called buckets. Every object has a URL of the form

https://cirrus-s3.epcc.ed.ac.uk/bucket-name/object-name

If an object is set to be “public” then anyone can download the object using a web-browser and this URL. For non-
public objects additional parameters or http headers are needed to handle authentication.

Bucket and object names should therefore be chosen to ensure these URLs are valid. Good practice is to stick to
alphanumeric characters underscores and hyphens. In particular you should avoid spaces as these cause problems
with some tools. An object name can contain slashes giving the appearance of a directory structure within the bucket.
However this is purely cosmetic. File browsing tools usually present object names with slashes as a directory hierarchy
but the object store just sees them as part of the object name.

Depending on the permissions the objects within a bucket may belong to different access-keys to the bucket itself.
However storage quotas are always calculated based on the owner of the bucket not the object.

10.3 Permissions and ACLs

Access permissions can be set on both buckets and objects. The Cirrus object store supports a combination of three
permissions.

1) Read
2) Write
3) Full-control

For buckets:

- Read permission allows you to lists its contents. You do not need this to access the object itself as long as you
  know the object name.
- Write permission allows you to add or delete objects.
- Full-control allows you to change permissions on the bucket.

For objects:

- Read permission allows the object to be downloaded.
- Full-control allows you to change permissions on the object.

These permissions can be granted to individual keys using an AccessControlList (ACL). You will usually need to
specify the key using its UUID when doing this. You can also grant permissions to two additional classes of users:

  **Authenticated users** This means any active key known to the object store.

  **All users** This means everyone with an internet connection.

You may want to set read permissions of this type when publishing public data but you should never grant write or
full-control permissions to these groups.

The simplest way of using the object-store is for each user to have a personal access-key and quota. This will allow
each user to store and retrieve their own data independently. However this makes large scale data sharing difficult.
ACLs need to be set for every object that include the key of every user that needs access. Whenever membership of
the access group changes, the ACLs of every object needs to be updated.

If you want to support data sharing you may wish to generate additional keys representing different levels of access.
For example a key that is allowed to modify a shared data-set and another that is only allowed to read it. You can
then share these keys with the appropriate groups of people. This makes managing the ACLs much less work as they
only have to reference the shared keys. When someone leaves a group you can revoke their access by changing the AccessSecret for the shared key and re-distributing the new secret to the remaining members.

ACLs should be set using the client interface to the object store - it is not possible to set ACLs through the SAFE interface.

## 10.4 Managing the Object Store from SAFE

Keys and quotas are managed through the SAFE. If a SAFE project has an allocation on the object store there will be a “Object store quotas” section in the Project Administration page for that project. Project managers can click on the button in that section to manage keys and quotas. From the quota management page you have two options:

- **New key** To create a new AccessKey
- **List keys** To show and manage existing keys.

When creating a key you need to provide a name for the key and a storage quota for the new key. The sum of all the key quotas within a project must be less than the total storage allocation of the project.

The List-keys page shows a list of the existing keys for the project. Click on one of the links to manage the corresponding key. The following options are available for each key:

- **View secret** This shows details about the key including the AccessKey name, the AccessSecret and the UUID.
- **Set permissions** This allows a project manager to share the key with selected members of the project. When a key is shared with somebody they will be able to view and download the key from the SAFE. If you want to revoke access to a key you can remove this permission then use **Regenerate** to change the AccessSecret. Other people who still have the key shared with them will be able to download the new secret as before.
- **Test** The SAFE will connect to the object store using the key and check that the key is working.
- **List Buckets** This shows the buckets owned by the key. You can also click-through to the bucket and browse its contents (using that keys permissions).
- **Change quota** This allows a project manager to change the size of the storage quota allocated to the key.
- **Lock/Unlock** An AccessKey can be locked/unlocked by a project manager. While a key is locked it cannot be used to access the object store.
- **Regenerate** A project manager can use this to change the AccessSecret. Permitted Users will be able to download the new value from the SAFE.

When a user had been given access to a key using the “Set permissions” menu the key will appear in their SAFE navigation menu under “Login accounts”->“Credentials”. This will then give them access to the following functions:

- View secret
- Test
- List Buckets

## 10.5 Browsing the Object store from your desktop

### 10.5.1 Windows: Cloudberry

There are a number of File browser UIS that can be used to browse the object store on your desktop. For example the Cloudberry browser can be used on Windows [https://www.cloudberrylab.com/explorer/amazon-s3.aspx](https://www.cloudberrylab.com/explorer/amazon-s3.aspx) and can be setup in the following way:
• Download and install the Freeware GUI from the above link.
• Select File->"New S3 compatible account"->"S3 Compatible"
• Fill in your AccessKey and AccessSecret. Use https://cirrus-s3.epcc.ed.ac.uk as the Service end-point.

10.5.2 Others: s3cmd

On non-Windows systems and for those that prefer command-line access we recommend that you install s3cmd:

• https://s3tools.org/s3cmd

This tool can also be installed in user space on other HPC systems using miniconda. Install miniconda using the command line installer as described in the Using Python chapter of this User Guide and then you can add s3cmd with:

    conda install -c conda-forge s3cmd

10.6 Using s3cmd to work with the object store on Cirrus

The Object store uses the Amazon S3 protocol so can be accessed using any of the standard tools developed to access AWS-S3. On the Cirrus command line, we have made s3cmd available via the standard Anaconda distribution. To get access to the tool, you first need to load the anaconda module:

    module load anaconda

Once the module is loaded, you can access the s3cmd tool.

10.6.1 Configure s3cmd

Note: You only need to do this once, before the first time you manipulate data on the object store.

Before you use s3cmd on Cirrus to transfer data, you need to first create a configuration file, run:

    s3cmd –configure

and use the following answers to the configuration questions:

• Access Key: use the value from SAFE
• Secret Key: use the value from SAFE
• Default Region uk-cirrus-1
• S3 Endpoint: cirrus-s3.epcc.ed.ac.uk
• DNS-style bucket+hostname:port template for accessing a bucket cirrus-s3.epcc.ed.ac.uk/
% (bucket)
• Encryption password: leave blank.
• Path to GPG program: leave blank
• Use HTTPS protocol: Yes
• HTTP Proxy server name: leave blank
• Test access with supplied credentials? Y
• *Save settings?* y to save the credential

You can re-run this command later to change any setting and it will default to your previous selections.

You can run `s3cmd --help` to see the various supported commands. We briefly describe how to create buckets, upload data from Cirrus, list buckets and their contents and download data to Cirrus in the sections below.

**Note:** Cirrus object-store does not support the CloudFront or Glacier options.

### 10.6.2 Create a bucket

Firstly, you need to create a bucket to store your data using `s3cmd mb`:

```
[auser@cirrus-login0 ~]$ s3cmd mb s3://examplebucket
Bucket 's3://examplebucket/' created
```

### 10.6.3 Upload data to the bucket

Now, you can upload data (as objects) to the bucket with `s3cmd put`:

```
[auser@cirrus-login0 ~]$ s3cmd put ~/random_2G.dat s3://examplebucket/random.dat
WARNING: Module python-magic is not available. Guessing MIME types based on file extensions.
upload: '/general/z01/z01/auser/random_2G.dat' -> 's3://examplebucket/random.dat'
  [part 1 of 137, 15MB] [1 of 1] 15728640 of 15728640 100% in 0s 22.16 MB/s done
upload: '/general/z01/z01/auser/random_2G.dat' -> 's3://examplebucket/random.dat'
  [part 2 of 137, 15MB] [1 of 1] 15728640 of 15728640 100% in 0s 25.31 MB/s done
...
upload: '/general/z01/z01/auser/random_2G.dat' -> 's3://examplebucket/random.dat'
  [part 137 of 137, 8MB] [1 of 1] 8388608 of 8388608 100% in 0s 32.80 MB/s done
```

By default, any object larger than 15MB in size is uploaded as a multipart upload - the data is split into smaller chunks and uploaded a piece at a time instead of as a single operation. In the example above, `s3://examplebucket/random.dat` is uploaded as 137 parts of default chunk size 15MB. This has the advantage that if a `s3cmd put` operation is interrupted, it’s possible to continue the operation where it left off. If uploading large objects and experiencing issues, you may need to experiment with increasing the multipart chunk size, either with using the `multipart-chunk-size-mb=N` option (where N is the desired new chunk size in megabytes), or by altering the default globally in your `.s3cfg` file in your home directory. The maximum allowed chunk size is 5GB.

Continue a paused or failed upload by passing `--upload-id= N` to the put command, plus the hash of the update to be continued. This is given at the point the initial upload is stopped, or details of all pending multipart uploads associated with a bucket can be found with `s3cmd multipart` and the name of the bucket:

```
[auser@cirrus-login0 ~]$ s3cmd multipart s3://examplebucket
s3://examplebucket/
Initiated Path Id
2019-12-12T13:22:27.000Z s3://examplebucket/random.dat
  7775611dd0c93819353abf93aa9bc7e6
```

### 10.6. Using s3cmd to work with the object store on Cirrus

67
Please be aware that incomplete multipart uploads do not expire and must be manually either continued or aborted to clear them. If not cleared then they will continue occupying that space in your project storage quota. To delete uploads from this queue rather than continue them, use `s3cmd abortmp`.

### 10.6.4 Listing buckets and the contents of buckets (objects)

You can list your buckets with `s3cmd ls`:

```
[auser@cirrus-login0 ~]$ s3cmd ls
2019-06-05 11:26 s3://examplebucket
```

and the contents of buckets (i.e. objects) with `s3cmd ls s3://<bucket>`:

```
[auser@cirrus-login0 ~]$ s3cmd ls s3://examplebucket
2019-06-05 11:28 2147483648 s3://examplebucket/random.dat
```

### 10.6.5 Downloading objects

Use the `s3cmd get` command to download data from a bucket:

```
[aturner@cirrus-login0 ~]$ s3cmd get s3://examplebucket/random.dat
download: 's3://examplebucket/random.dat' -> './random.dat' [1 of 1]
8388608 of 8388608 100% in 15s 32.80 MB/s done
```
References and further reading

11.1 Online Documentation and Resources

- GNU compiler online documentation: http://gcc.gnu.org/onlinedocs/
- MPI Home pages: http://www-unix.mcs.anl.gov/mpi/
- Free MPI implementation useful for testing: http://www.open-mpi.org/software/ompi/v1.2/
- Various HPC Workshops by NCCS: http://www.nccs.gov/user-support/training-education/workshop-archives/
- An HPC tutorial: http://www.llnl.gov/computing/hpc/training/
- HPC tutorials by NCSA http://www.citutor.org/login.html

11.2 MPI programming

- Using MPI. Gropp, Lusk, Skjellum. MIT Press. ISBN 0 262 57104 8

11.3 OpenMP programming

- Parallel Programming in OpenMP. Chandra, Kohr, Menon, Dagum, Maydan, McDonald. Morgan Kaufmann. ISBN: 1558606718
11.4 Parallel programming

- Designing and Building Parallel Programs. Ian Foster. Addison-Wesley. ISBN 0 201 57594 9 http://www.mcs.anl.gov/dbpp/
- Parallel Computing Works! Roy D. Williams, Paul C. Messina (Editor), Geoffrey Fox (Editor), Mark Fox Morgan Kaufmann Publishers; ISBN: 1558602534
- Parallel programming with MPI. Peter S. Pancheco. The complete set of C and Fortran example programs for this book are available at: http://www.cs.usfca.edu/mpi

11.5 Programming languages


11.6 Programming skills

- Debugging and Performance Tuning for Parallel Computing Systems, Simmons et al.
Altair Hyperworks

Hyperworks includes best-in-class modeling, linear and nonlinear analyses, structural and system-level optimization, fluid and multi-body dynamics simulation, electromagnetic compatibility (EMC), multiphysics analysis, model-based development, and data management solutions.

12.1 Useful Links

• Hyperworks 14 User Guide

12.2 Using Hyperworks on Cirrus

Hyperworks is licenced software so you require access to a Hyperworks licence to access the software. For queries on access to Hyperworks on Cirrus and to enable your access please contact the Cirrus helpdesk.

The standard mode of using Hyperworks on Cirrus is to use the installation of the Desktop application on your local workstation or laptop to set up your model/simulation. Once this has been done you would transfer the required files over to Cirrus using SSH and then launch the appropriate Solver program (OptiStruct, RADIOSS, MotionSolve).

Once the Solver has finished you can transfer the output back to your local system for visualisation and analysis in the Hyperworks Desktop.

12.3 Running serial Hyperworks jobs

Each of the Hyperworks Solvers can be run in serial on Cirrus in a similar way. You should construct a batch submission script with the command to launch your chosen Solver with the correct command line options.

For example, here is a job script to run a serial RADIOSS job on Cirrus:
# PBS job options (name, compute nodes, job time)
#PBS -N HW_RADIOSS_test
#PBS -l select=1:ncpus=1
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load Hyperworks module
module load altair-hwsolvers/14.0.210

# Run the RADIOSS Solver in serial
radioss box.fem

12.4 Running parallel Hyperworks jobs

Only the OptiStruct Solver currently supports parallel execution. OptiStruct supports a number of parallel execution modes of which two can be used on Cirrus:

- Shared memory (SMP) mode uses multiple cores within a single node
- Distributed memory (SPMD) mode uses multiple cores across multiple nodes via the MPI library

12.4.1 OptiStruct SMP

- OptiStruct SMP documentation

You can use up to 36 physical cores (or 72 virtual cores using HyperThreading) for OptiStruct SMP mode as these are the maximum numbers available on each Cirrus compute node.

You use the -nt option to OptiStruct to specify the number of cores to use.

For example, to run an 18-core OptiStruct SMP calculation you could use the following job script:

#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N HW_OptiStruct_SMP
#PBS -l select=1:ncpus=18
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load Hyperworks module
12.4.2 OptiStruct SPMD (MPI)

- OptiStruct SPMD documentation

There are four different parallelisation schemes for SPMD OptiStruct that are selected by different flags:

- Load decomposition (master/slave): -mpimode flag
- Domain decomposition: -ddmmode flag
- Multi-model optimisation: -mmomode flag
- Failsafe topology optimisation: -fsomode flag

You should launch OptiStruct SPMD using the standard Intel MPI mpirun command.

Note: OptiStruct does not support the use of SGI MPT, you must use Intel MPI.

Example OptiStruct SPMD job submission script:

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N HW_OptiStruct_SPMD

# Use 2 nodes for this calculation
#PBS -l select=2:ncpus=36
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load Hyperworks module and Intel MPI
module load altair-hwsolvers/14.0.210
module load intel-mpi-17

# Run the OptiStruct SPMD Solver (domain decompostion mode)
# Use 72 cores, 36 on each node (i.e. all physical cores)
mpirun -ppn 36 -n 72 $ALTAIR_HOME/hwsolvers/optistruct/bin/linux64/optistruct_14.0.211_linux64_impi box.fem -ddmmode
```
ANSYS Fluent is a computational fluid dynamics (CFD) tool. Fluent includes well-validated physical modelling capabilities to deliver fast, accurate results across the widest range of CFD and multi-physics applications.

13.1 Useful Links

- ANSYS Fluent User Guides

13.2 Using ANSYS Fluent on Cirrus

ANSYS Fluent on Cirrus is only available to researchers who bring their own licence. Other users cannot access the version centrally-installed on Cirrus.

If you have any questions regarding ANSYS Fluent on Cirrus please contact the Cirrus Helpdesk.

13.3 Running parallel ANSYS Fluent jobs

The following batch file starts Fluent in a command line mode (no GUI) and starts the Fluent batch file “inputfile”. One parameter that requires particular attention is “-t504”. In this example 14 Cirrus nodes (14 * 72 = 1008 cores) are allocated; where half of the 1008 cores are physical and the other half are virtual. To run fluent optimally on Cirrus, only the physical cores should be employed. As such, fluent’s -t flag should reflect the number of physical cores: in this example, “-t504” is employed.

```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N ANSYS_test
#PBS -l select=14:ncpus=36
#PBS -l walltime=23:04:0
```

(continues on next page)
#PBS -l place=scatter:excl
#PBS -k oe
# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]
module purge
module load mpt perfboost
module load ansys
export OMP_NUM_THREADS=1
export SGI_MPI_HOME=$MPI_ROOT
uniq $PBS_NODEFILE | cut -d . -f 1 > ~/fluent.launcher.host.txt

cd $PBS_O_WORKDIR
./fluent 3ddp -g-i inputfile.fl \n- pinfiniband -alnamd64 -r17.2.0 -t504 -mpi=intel 
- cnf=~/fluent.launcher.host.txt 
- path/lustre/sw/ansys/v172/fluent/ - ssh >& outputfile.txt

Below is the Fluent “inputfile.fl” batch script. Anything that starts with a “;” is a comment. This script does the following:

- Starts a transcript (i.e. Fluent output is redirected to a file [transcript_output_01.txt])
- Reads a case file [a case file in Fluent is a model]
- Reads a data file [a data file in Fluent is the current state of a simulation (i.e. after X iterations)]
- Prints latency and bandwidth statistics
- Prints and resets timers
- Run 50 iterations of the simulation
- Prints and resets timers
- Save the data file (so that you can continue the simulation)
- Stops the transcript
- Exits Fluent

13.4 Actual Fluent script (“inputfile.fl”):

Replace [Your Path To ] before running

; Start transcript
/file/start-transcript [Your Path To ]/transcript_output_01.txt
; Read case file
rc [Your Path To ]/200M-CFD-Benchmark.cas
; Read data file
/file/read-data [Your Path To ]/200M-CFD-Benchmark-500.dat
; Print statistics
/parallel/bandwidth
/parallel/latency
/parallel/timer/usage
/parallel/timer/reset

(continues on next page)
; Calculate 50 iterations
   it 50
; Print statistics
   /parallel/timer/usage
   /parallel/timer/reset
; Write data file
   wd [Your Path To ]/200M-CFD-Benchmark-500-new.dat
; Stop transcript
   /file/stop-transcript
; Exit Fluent
   exit
   yes
CASTEP is a leading code for calculating the properties of materials from first principles. Using density functional theory, it can simulate a wide range of properties of materials properties including energetics, structure at the atomic level, vibrational properties, electronic response properties etc. In particular it has a wide range of spectroscopic features that link directly to experiment, such as infra-red and Raman spectroscopies, NMR, and core level spectra.

14.1 Useful Links

- CASTEP User Guides
- CASTEP Tutorials
- CASTEP Licensing

14.2 Using CASTEP on Cirrus

CASTEP is only available to users who have a valid CASTEP licence.

If you have a CASTEP licence and wish to have access to CASTEP on Cirrus please submit a request through the SAFE.

Note: CASTEP versions 19 and above require a separate licence from CASTEP versions 18 and below so these are treated as two separate access requests.

14.3 Running parallel CASTEP jobs

CASTEP can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node.

For example, the following script will run a CASTEP job using 4 nodes (144 cores).
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N CASTEP_test
#PBS -l select=4:ncpus=36
#PBS -l place=scatter:excl
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load CASTEP and MPI modules
module load castep

# Set OMP_NUM_THREADS=1 to avoid unintentional threading
export OMP_NUM_THREADS=1

# Run using input in test_calc.in
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
mpiexec_mpt -ppn 36 -n 144 castep.mpi test_calc
CP2K is a quantum chemistry and solid state physics software package that can perform atomistic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems. CP2K provides a general framework for different modelling methods such as DFT using the mixed Gaussian and plane waves approaches GPW and GAPW. Supported theory levels include DFTB, LDA, GGA, MP2, RPA, semi-empirical methods (AM1, PM3, PM6, RM1, MNDO, . . . ), and classical force fields (AMBER, CHARMM, . . . ). CP2K can do simulations of molecular dynamics, metadynamics, Monte Carlo, Ehrenfest dynamics, vibrational analysis, core level spectroscopy, energy minimisation, and transition state optimisation using NEB or dimer method.

15.1 Useful Links

- CP2K Reference Manual
- CP2K HOWTOs
- CP2K FAQs

15.2 Using CP2K on Cirrus

CP2K is available through the `cp2k-mpt` module. MPI only `cp2k.popt` and MPI/OpenMP Hybrid `cp2k.pamp` binaries are available.

**IMPORTANT:** Running cp2k in hybrid OpenMP/MPI mode requires some non-standard steps. Please see the `Running CP2K in OpenMP/MPI Hybrid Mode` section below for further details.

15.3 Running parallel CP2K jobs - MPI Only

To run CP2K using MPI only, load the `cp2k-mpt` module and use the `cp2k.popt` executable.

For example, the following script will run a CP2K job using 4 nodes (144 cores):
15.4 Running Parallel CP2K Jobs - MPI/OpenMP Hybrid Mode

To run CP2K using MPI and OpenMP, load the `cp2k-mpt` module and use the `cp2k.psmmp` executable.

Due to a thread placement bug in SGI MPT's `omplace`, tool for GCC-compiled software, launching the executable must be achieved in a different way to other hybrid OpenMP/MPI codes on Cirrus.

You must first run the `placement` tool (included in the module) to produce a thread placement file, `place.txt`. For example, if you wish to use 6 threads per process, use:

```bash
export OMP_NUM_THREADS=6
placement $OMP_NUM_THREADS
```

to produce the placement file. Then launch the executable using `mpiexec_mpt` and `dplace` (instead of `omplace`) as follows:

```bash
mpiexec_mpt -n 6 dplace -p place.txt cp2k.psmmp ...
```

For example, the following script will run a CP2K job using 4 nodes, with 6 OpenMP threads per MPI process:

```bash
#!/bin/bash --login
# PBS job options (name, compute nodes, job time)
#PBS -N CP2K_test
#PBS -l select=4:ncpus=36
#PBS -l place=scatter:excl
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load CASTEP and MPI modules
module load cp2k-mpt
module load mpt
module load intel-cmkl-16

#Ensure that no libraries are inadvertently using threading
export OMP_NUM_THREADS=1

# Run using input in test.inp
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
mpiexec_mpt -ppn 36 -n 144 cp2k.popt -i test.inp
```
# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load CASTEP and MPI modules
module load cp2k-mpt
module load mpt
module load intel-cmk1-16

export OMP_NUM_THREADS=6
placement $OMP_NUM_THREADS

# Run using input in test.inp
# Notes:
# - '-ppn 6' is required to use six processes per node
# - we use 'dplace' with the placement file 'place.txt' to specify
#   thread binding
mpiexec_mpt -ppn 6 -n 24 dplace -p place.txt cp2k.psmp -i test.inp
FLACS from Gexcon is the industry standard for CFD explosion modelling and one of the best validated tools for modeling flammable and toxic releases in a technical safety context.

The Cirrus cluster is ideally suited to run multiple FLACS simulations simultaneously, via its batch system PBS. Short lasting simulations (of typically up to a few hours computing time each) can be processed efficiently and you could get a few hundred done in a day or two. In contrast, the Cirrus cluster is not particularly suited for running single big FLACS simulations with many threads: each node on Cirrus has 2x4 memory channels, and for memory-bound applications like FLACS multi-threaded execution will not scale linearly beyond eight cores. On most systems, FLACS will not scale well to more than four cores (cf. the FLACS User’s Manual), and therefore multi-core hardware is normally best used by increasing the number of simulations running in parallel rather than by increasing the number of cores per simulation.

Gexcon has two different service offerings on Cirrus: FLACS-Cloud and FLACS-HPC. From FLACS v10.7, FLACS-Cloud is the preferable way to exploit the HPC cluster, directly from the FLACS graphical user interfaces. For users who are familiar with accessing remote Linux HPC systems manually, FLACS-HPC may be an option. Both services are presented below.

16.1 FLACS-Cloud

FLACS-Cloud is a high performance computing service available right from the FLACS-Risk user interface, as well as from the FLACS RunManager. It allows you to run FLACS simulations on the high performance cloud computing infrastructure of Gexcon’s partner EPCC straight from the graphical user interfaces of FLACS – no need to manually log in, transfer data, or start jobs!

By using the FLACS-Cloud service, you can run a large number of simulations very quickly, without having to invest into in-house computing hardware. The FLACS-Cloud service scales to your your demand and facilitates running projects with rapid development cycles.

The workflow for using FLACS-Cloud is described in the FLACS User’s Manual and in the FLACS-Risk documentation; you can also find basic information in the knowledge base of the FLACS User Portal (accessible for FLACS license holders).
16.2 FLACS-HPC

Compared to FLACS-Cloud, the FLACS-HPC service is built on more traditional ways of accessing and using a remote Linux cluster. Therefore the user experience is more basic, and FLACS has to be run manually. For an experienced user, however, this way of exploiting the HPC system can be at least as efficient as FLACS-Cloud.

Follow the steps below to use the FLACS-HPC facilities on Cirrus.

Note: The instructions below assume you have a valid account on Cirrus. To get an account please first get in touch with FLACS support at flacs@gexcon.com and then see the instructions in the Tier-2 SAFE Documentation.

Note: In the instructions below you should substitute “username” by your actual Cirrus username.

16.2.1 Log into Cirrus

Log into Cirrus following the instructions at Connecting to Cirrus.

16.2.2 Upload your data to Cirrus

Transfer your data to Cirrus by following the instructions at Data Transfer Guide.

For example, to copy the scenario definition files from the current directory to the folder project_folder in your home directory on Cirrus run the following command on your local machine:

```
rsync -avz c*.dat3 username@cirrus.epcc.ed.ac.uk:project_folder
```

Note that this will preserve soft links as such; the link targets are not copied if they are outside the current directory.

16.2.3 Submit a FLACS job to the queue

To run FLACS on Cirrus you must first change to the directory where your FLACS jobs are located, use the `cd` (change directory) command for Linux. For example

```
cd projects/sim
```

Load the `flacs` module to make the application available:

```
module load flacs
```

Submit your FLACS jobs using the `qsub` command. For example:

```
qsub -A xyz -l select=1:ncpus=1 -l walltime=6:00:00 -- /lustre/sw/flacs/10.5.1/FLACS_v10.5/bin/run_runflacs -dir projects/sim 010101
```

The `-A xyz` option is obligatory and states the account `xyz` that the CPU consumption will be billed to. You can check your account in SAFE.

The `-l select=x:ncpus=y` option specifies the resource allocation for the job you are starting. The parameter `x` is the number of nodes required and the parameter `y` is the number of cores required. For a serial FLACS job you would use `-l select=1:ncpus=1`.

The maximum length of time (i.e. walltime) you want the job to run is specified with the `-l walltime=[hh:mm:ss]` option. After this time, your job will be stopped by the job scheduler. Setting a very high walltime limit may lead to your job being given lower priority and thus wait longer in the queue. The default walltime is 12 hours.
All Flacs jobs must be submitted to the flacs queue using the option `-q flacs`; the flacs queue ensures FLACS licenses are provisioned correctly for the jobs.

After the `--` which marks the beginning of the command to run, the Flacs executable is given *with its absolute path*. Having loaded the flacs module (see above) you can find the location by

```
which run_runflacs
```

The `run_runflacs` command in turn needs two arguments: first, after `-dir`, the directory where to run the job and create the output; if it is the current directory then you can pass `-dir `pwd``. Second, the job number of the FLACS scenario.

### 16.2.4 Multithreaded jobs

Multithreaded flacs simulations can be run on Cirrus with the following job submission:

```
qsub -A xyz -l select=1:ncpus=4 -l walltime=6:00:00 -q flacs -- /lustre/sw/flacs/10.5.1/FLACS_v10.5/bin/run_runflacs -dir projects/sim 010101 NumThreads=4
```

It is important to note that when submitting multithreaded flacs simulations the `ncpus=x` option must be used in order for the queueing system to allocate the correct resources. In addition to this one must also specify the number of threads used by the simulation with the `NumThreads=x` option to the `run_runflacs`.

### 16.2.5 Submit FLACS jobs from a script

In your script, change to the directory with the job files and load the flacs module as explained above.

When submitting several jobs it is advisable to add the `-N name` option to the `qsub` command, with the FLACS job number being part of the first ten characters of the name. In this way you can easily identify the jobs in the queue (see below).

During testing it has been shown that job submission to the queue runs more smoothly when there is a short delay of 5 seconds before subsequent `qsub` commands.

A script submitting the scenarios 000012, 000023 and 000117 to the queue could look like this:

```
module load flacs/10.5.1
sleep 5; qsub -A xyz -l select=1:ncpus=1 -l walltime=24:00:00 -N f-000012 -q flacs -V -- `which run_runflacs` -dir `pwd` 000012
sleep 5; qsub -A xyz -l select=1:ncpus=1 -l walltime=24:00:00 -N f-000023 -q flacs -V -- `which run_runflacs` -dir `pwd` 000023
sleep 5; qsub -A xyz -l select=1:ncpus=1 -l walltime=24:00:00 -N f-000117 -q flacs -V -- `which run_runflacs` -dir `pwd` 000117
```

This is also easy to formulate as a loop.

### 16.2.6 Monitor your jobs

You can monitor the progress of your jobs with the `qstat` command. This will list all jobs that are running or queued on the system. To list only your jobs use:

```
qstat -u username
```
16.2.7 Submitting many FLACS jobs as a job array

Running many related scenarios with the Flacs simulator is ideally suited for using job arrays, i.e. running the simulations as part of a single job.

A job script for running a job array with 128 Flacs scenarios that are located in the current directory could look like this:

```bash
#!/bin/bash --login
#PBS -l select=1:ncpus=1
#PBS -N disp2
#PBS -J 1-128
#PBS -j oe
#PBS -l walltime=48:00:00
#PBS -q flacs
#PBS -V
cd ${PBS_O_WORKDIR}
CS_FILES=(`ls -1 cs???????.dat3`) # NR_OF_JOBS=${#CS_FILES[@]}
JOB_FIRST=1
JOB_LAST=128
for (( i=0; i<$(expr ${JOB_LAST} - ${JOB_FIRST}); i++ )); do
  JOB_IDS[${i}]=${CS_FILES[$(expr $i + ${JOB_FIRST})]:2:6}
done
module load flacs
JOB_INDEX=$(( $PBS_ARRAY_INDEX - 1 ))
`which run_runflacs` ${JOB_IDS[${JOB_INDEX}]}
```

Due to the way the job scheduler interprets this script, the number of jobs has to be hard-coded in the first (non-bash) part of the job script and cannot be determined based on the number of scenarios in the current directory.

16.2.8 Transfer data from Cirrus to your local system

After your simulations are finished, transfer the data back from Cirrus following the instructions at Data Transfer Guide.

For example, to copy the result files from the directory `project_folder` in your home directory on Cirrus to the folder `/tmp` on your local machine use:

```bash
rsync -rvz --include='r[13t]***' --exclude='* ' username@cirrus.epcc.ed.ac.uk:project_folder/ /tmp
```

16.2.9 Billing for FLACS-HPC use on Cirrus

CPU time on Cirrus is measured in CPUh for each job run on a compute node, based on the number of physical cores employed. Only jobs submitted to compute nodes via `qsub` are charged. Any processing on a login node is not charged. However, using login nodes for computations other than simple pre- or post-processing is strongly discouraged.

Gexcon normally bills monthly for the use of FLACS-Cloud and FLACS-HPC, based on the Cirrus CPU usage logging.
16.3 Getting help

Get in touch with FLACS Support by email to flacs@gexcon.com if you encounter any problems. For issues related to Cirrus rather than FLACS contact the Cirrus helpdesk.
 CHAPTER 17

Gaussian

Gaussian is a general-purpose computational chemistry package.

17.1 Useful Links

- Gaussian User Guides

17.2 Using Gaussian on Cirrus

Gaussian on Cirrus is only available to University of Edinburgh researchers through the University’s site li-
cence. Users from other institutions cannot access the version centrally-installed on Cirrus.

If you wish to have access to Gaussian on Cirrus please contact the Cirrus Helpdesk.

Gaussian cannot run across multiple nodes. This means that the maximum number of cores you can use for Gaussian
jobs is 36 (the number of cores on a compute node). In reality, even large Gaussian jobs will not be able to make
effective use of more than 8 cores. You should explore the scaling and performance of your calculations on the system
before running production jobs.

17.3 Scratch Directories

Before using Gaussian for the first time, you should create a directory in your home directories to hold temporary files
used by Gaussian, e.g.

```bash
mkdir ~/g09tmp
```
17.4 Running serial Gaussian jobs

In many cases you will use Gaussian in serial mode. The following example script will run a serial Gaussian job on Cirrus (before using, ensure you have created a Gaussian scratch directory as outlined above).

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N G09_test
#PBS -l select=1:ncpus=1
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load Gaussian module
module load gaussian

# Setup the Gaussian environment
source $g09root/g09/bsd/g09.profile

# Location of the scratch directory
export GAUSS_SCRDIR=$HOME/g09tmp

# Run using input in "test0027.com"
g09 test0027
```

17.5 Running parallel Gaussian jobs

Gaussian on Cirrus can use shared memory parallelism through OpenMP by setting the `OMP_NUM_THREADS` environment variable. The number of cores requested in the job should also be modified to match.

For example, the following script will run a Gaussian job using 4 cores.

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N G09_test
#PBS -l select=1:ncpus=4
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load Gaussian module
module load gaussian

# Setup the Gaussian environment
source $g09root/g09/bsd/g09.profile
```
# Location of the scratch directory
export GAUSS_SCRDIR=$HOME/g09tmp

# Run using input in "test0027.com"
export OMP_NUM_THREADS=4
g09 test0027

17.5. Running parallel Gaussian jobs
GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the nonbonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers.

18.1 Useful Links

- GROMACS User Guides
- GROMACS Tutorials

18.2 Using GROMACS on Cirrus

GROMACS is Open Source software and is freely available to all Cirrus users. A number of versions are available:

- Serial/shared memory, single precision: gmx
- Parallel MPI/OpenMP, single precision: gmx_mpi
- GPU version, single precision: gmx

18.3 Running parallel GROMACS jobs: pure MPI

GROMACS can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node.

For example, the following script will run a GROMACS MD job using 4 nodes (144 cores) with pure MPI.
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N mdrun_test
#PBS -l select=4:ncpus=36
# Make sure you are not sharing nodes with other users
#PBS -l place=scatter:excl
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load GROMACS and MPI modules
module load gromacs
module load mpt

# Run using input in test_calc.tpr
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
export OMP_NUM_THREADS=6
mpiexec_mpt -ppn 6 -n 24 omplace -nt 6 gmx_mpi mdrun -s test_calc.tpr

18.4 Running parallel GROMACS jobs: hybrid MPI/OpenMP

The following script will run a GROMACS MD job using 4 nodes (144 cores) with 6 MPI processes per node (24 MPI processes in total) and 6 OpenMP threads per MPI process.

#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N mdrun_test
#PBS -l select=4:ncpus=36
# Make sure you are not sharing nodes with other users
#PBS -l place=scatter:excl
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load GROMACS and MPI modules
module load gromacs
module load mpt

# Run using input in test_calc.tpr
export OMP_NUM_THREADS=6
mpiexec_mpt -ppn 6 -n 24 omplace -nt 6 gmx_mpi mdrun -s test_calc.tpr
18.5 GROMACS GPU jobs

A separate build of GROMACS is provided to run on the NVIDIA GPU nodes on Cirrus (for details see *Using the Cirrus GPU Nodes*). Note also that the GPU version targets the GPU host nodes, which are Intel Skylake; this version will not run on the front end or the other non-GPU back-end nodes.

The GPU version is accessed via, e.g.,

```bash
module load gromacs-gpu/2020
```

As there are currently a limited number of GPU nodes available, a distributed memory MPI version is not available: only the ‘thread MPI’ version is available (that is, ‘gmx’ is available, but not ‘gmx_mpi’).

Further, we recommend exclusive node usage to prevent possible contention with other user jobs. An example of the form of the PBS submission script is:

```bash
#!/bin/bash --login

#PBS -q gpu
#PBS -N job-name
#PBS -l select=1:ncpus=40:ngpus=4
#PBS -l place=scatter:excl
#PBS -l walltime=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load GROMACS GPU module
module load gromacs-gpu/2020

# Invocation will depend on problem type ...
export OMP_NUM_THREADS=10
gmx mdrun -ntmpi 4 -nb gpu -pme cpu ...
```

Information on how to assign different types of calculation to the CPU or GPU appears in the GROMACS documentation under *Getting good performance from mdrun*.
HELYX is a comprehensive, general-purpose computational fluid dynamics (CFD) software package for engineering analysis and design optimisation developed by ENGYS. The package features an advanced open-source CFD simulation engine and a client-server GUI to provide a flexible and cost-effective HPC solver platform for enterprise applications.

19.1 Useful Links

- Information about HELYX
- Information about ENGYS

19.2 Using HELYX on Cirrus

HELYX is only available on Cirrus to authorised users with a valid license to use the software. For any queries regarding HELYX on Cirrus, please contact ENGYS or the Cirrus Helpdesk.

HELYX applications can be run on Cirrus in two ways:

- Manually from the command line, using a SSH terminal to access the cluster’s master node.
- Interactively from within the GUI, using the dedicated client-server node to connect remotely to the cluster.

A complete user’s guide to access HELYX on demand via Cirrus is provided by ENGYS as part of the service offering.

19.3 Running Parallel HELYX Jobs

The standard execution of HELYX applications on Cirrus is handled through the command line using a submission script to control PBSPro. A basic submission script for running multiple HELYX applications in parallel using the SGI-MPT (Message Passing Toolkit) module is included below. In this example the applications helyxHexMesh and caseSetup are run sequentially in 144 cores.
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N HELYX_MPI_Job
#PBS -l select=4:ncpus=36
#PBS -l place=scatter:excl
#PBS -l walltime=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load mpt/2.14
module load gcc/6.2.0

# Load HELYX-Core environment
export FOAM_INST_DIR=/lustre/home/y07/helyx/v3.0.2/CORE
. /lustre/home/y07/helyx/v3.0.2/CORE/HELYXcore-3.0.2/etc/bashrc

# Set the number of threads to 1
export OMP_NUM_THREADS=1

# Launch HELYX applications in parallel
export myoptions="-parallel"
jobs="helyxHexMesh caseSetup"

for job in `echo $jobs`
do
  case "$job" in
    * ) options="$myoptions" ;;
esac

  mpiexec_mpt -ppn 36 -n 144 $job $myoptions 2>&1 | tee log/$job.$PBS_JOBID.out

done

Alternatively, the user can execute most HELYX applications on Cirrus interactively via the GUI by following these simple steps:

1. Launch HELYX GUI in the local Windows or Linux machine.

2. Create a client-server connection to Cirrus using the dedicated node provided for this service. Enter the user login details and the total number of processors to be employed in the cluster for parallel execution.

3. Use the GUI in the local machine to access the remote file system in Cirrus to load a geometry, create a computational grid, set up a simulation, solve the flow, and post-process the results using the HPC resources available in the cluster. The scheduling associated with every HELYX job is handled automatically by the client-server.

4. Visualise the remote data from the local machine, perform changes in the model, and complete as many flow simulations in Cirrus as required. Disconnect the client-server at any point during execution, leave the solver running in the cluster, and resume the connection to Cirrus from another client machine to reload an existing case when needed.
LAMMPS, is a classical molecular dynamics code, and an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator. LAMMPS has potentials for solid-state materials (metals, semiconductors) and soft matter (biomolecules, polymers) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale.

## 20.1 Useful Links

- LAMMPS Mailing list details [https://lammps.sandia.gov/mail.html](https://lammps.sandia.gov/mail.html)

## 20.2 Using LAMMPS on Cirrus

LAMMPS is freely available to all Cirrus users.

## 20.3 Running parallel LAMMPS jobs

LAMMPS can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node. For example, the following script will run a LAMMPS MD job using 4 nodes (144 cores) with pure MPI.

```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N lammps_test
#PBS -l select=4:ncpus=36
# Make sure you are not sharing nodes with other users
#PBS -l place=scatter:excl
```

(continues on next page)
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
which $PBS_O_WORKDIR

# Load LAMMPS module
module load lammps

# Run using input in in.test
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
mpiexec_mpt -ppn 36 -n 144 lmp_mpi < in.test

## 20.4 Compiling LAMMPS on Cirrus

Compile instructions for LAMMPS on Cirrus can be found on GitHub:

- [Cirrus LAMMPS compile instructions](#)
MATLAB combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly.

### 21.1 Useful Links

- MATLAB Documentation

### 21.2 Using MATLAB on Cirrus

MATLAB R2018b is available on Cirrus.

This installation of MATLAB on Cirrus is covered by an Academic License - for use in teaching, academic research, and meeting course requirements at degree granting institutions only. Not for government, commercial, or other organizational use.

**If your use of MATLAB is not covered by this license then please do not use this installation.** Please contact the Cirrus Helpdesk to arrange use of your own MATLAB license on Cirrus.

This is MATLAB Version 9.5.0.1033004 (R2018b) Update 2 and provides the following toolboxes

<table>
<thead>
<tr>
<th>Toolbox</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB</td>
<td>Version 9.5</td>
</tr>
<tr>
<td>Simulink</td>
<td>Version 9.2</td>
</tr>
<tr>
<td>5G Toolbox</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Aerospace Blockset</td>
<td>Version 4.0</td>
</tr>
<tr>
<td>Aerospace Toolbox</td>
<td>Version 3.0</td>
</tr>
<tr>
<td>Antenna Toolbox</td>
<td>Version 3.2</td>
</tr>
<tr>
<td>Audio System Toolbox</td>
<td>Version 1.5</td>
</tr>
<tr>
<td>Automated Driving System Toolbox</td>
<td>Version 1.3</td>
</tr>
<tr>
<td>Bioinformatics Toolbox</td>
<td>Version 4.11</td>
</tr>
<tr>
<td>Communications Toolbox</td>
<td>Version 7.0</td>
</tr>
<tr>
<td>Computer Vision System Toolbox</td>
<td>Version 8.2</td>
</tr>
</tbody>
</table>
### Control System Toolbox  Version 10.5
### Curve Fitting Toolbox  Version 3.5.8
### DO Qualification Kit  Version 3.6
### DSP System Toolbox  Version 9.7
### Database Toolbox  Version 9.0
### Datafeed Toolbox  Version 5.8
### Deep Learning Toolbox  Version 12.0
### Econometrics Toolbox  Version 5.1
### Embedded Coder  Version 7.1
### Filter Design HDL Coder  Version 3.1.4
### Financial Instruments Toolbox  Version 2.8
### Financial Toolbox  Version 5.12
### Fixed-Point Designer  Version 6.2
### Fuzzy Logic Toolbox  Version 2.4
### GPU Coder  Version 1.2
### Global Optimization Toolbox  Version 4.0
### HDL Coder  Version 3.13
### HDL Verifier  Version 5.5
### IEC Certification Kit  Version 3.12
### Image Acquisition Toolbox  Version 5.5
### Image Processing Toolbox  Version 10.3
### Instrument Control Toolbox  Version 3.14
### LTE HDL Toolbox  Version 1.2
### LTE Toolbox  Version 3.0
### MATLAB Coder  Version 4.1
### MATLAB Distributed Computing Server  Version 6.13
### MATLAB Report Generator  Version 5.5
### Mapping Toolbox  Version 4.7
### Model Predictive Control Toolbox  Version 6.2
### Optimization Toolbox  Version 8.2
### Parallel Computing Toolbox  Version 6.13
### Partial Differential Equation Toolbox  Version 3.1
### Phased Array System Toolbox  Version 4.0
### Polyspace Bug Finder  Version 2.6
### Polyspace Code Prover  Version 9.10
### Powertrain Blockset  Version 1.4
### Predictive Maintenance Toolbox  Version 1.1
### RF Blockset  Version 7.1
### RF Toolbox  Version 3.5
### Risk Management Toolbox  Version 1.4
### Robotics System Toolbox  Version 2.1
### Robust Control Toolbox  Version 6.5
### Sensor Fusion and Tracking Toolbox  Version 1.0
### Signal Processing Toolbox  Version 8.1
### SimBiology  Version 5.8.1
### SimEvents  Version 5.5
### Simscape  Version 4.5
### Simscape Driveline  Version 2.15
### Simscape Electrical  Version 7.0
### Simscape Fluids  Version 2.5
### Simscape Multibody  Version 6.0
### Simulink 3D Animation  Version 8.1
### Simulink Check  Version 4.2
### Simulink Code Inspector  Version 3.3
### Simulink Coder  Version 9.0
### Simulink Control Design  Version 5.2
### Simulink Coverage  Version 4.2
21.3 Running MATLAB jobs

On Cirrus, MATLAB is intended to be used on the compute nodes within PBS job scripts. Use on the login nodes should be restricted to setting preferences, accessing help, and launching MDCS jobs. It is recommended that MATLAB is used without a GUI on the compute nodes, as the interactive response is slow.

21.4 Running parallel MATLAB jobs using the local cluster

The license for this installation of MATLAB provides only 32 workers via MDCS but provides 36 workers via the local cluster profile (there are 36 cores on a Cirrus compute node), so we only recommend the use of MDCS to test the configuration of distributed memory parallel computations for eventual use of your own MDCS license.

The local cluster should be used within a PBS job script - you submit a job that runs MATLAB and uses the local cluster, which is the compute node that the job is running on.

MATLAB will normally use up to the total number of cores on a node for multi-threaded operations (e.g. matrix inversions) and for parallel computations. It also make no restriction on its memory use. These features are incompatible with the shared use of nodes on Cirrus. For the local cluster, a wrapper script is provided to limit the number of cores and amount of memory used, in proportion to the number of CPUs selected in the PBS job script. Please use this wrapper instead of using MATLAB directly.

Say you have a job that requires 3 workers, each running 2 threads. As such, you should employ 3x2=6 cores. An example job script for this particular case would be

```
#PBS -N Example_MATLAB_Job
#PBS -l select=1:ncpus=6
#PBS -l walltime=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

module load matlab
```

(continues on next page)
matlab_wrapper -nodisplay < /lustre/sw/cse-matlab/examples/testp.m > testp.log

Note, for MATLAB versions R2019 and later, the `matlab_wrapper_2019` script may be required (see 2019 section below).

This would run the `testp.m` script, without a display, and exit when `testp.m` has finished. 6 CPUs are selected, which correspond to 6 cores, and the following limits would be set initially

```matlab
ncores = 6;
memory = 42GB;
```

Maximum number of computational threads (maxNumCompThreads) = 6
Preferred number of workers in a parallel pool (PreferredNumWorkers) = 6
Number of workers to start on your local machine (NumWorkers) = 6
Number of computational threads to use on each worker (NumThreads) = 1

The `testp.m` program sets `NumWorkers` to 3 and `NumThreads` to 2

```matlab
cirrus_cluster = parcluster('local');
ncores = cirrus_cluster.NumWorkers * cirrus_cluster.NumThreads;
cirrus_cluster.NumWorkers = 3;
cirrus_cluster.NumThreads = 2;
fprintf("NumWorkers = %d NumThreads = %d ncores = %d\n",
        cirrus_cluster.NumWorkers,
        cirrus_cluster.NumThreads,ncores);
if cirrus_cluster.NumWorkers * cirrus_cluster.NumThreads > ncores
    disp("NumWorkers * NumThreads > ncores");
    disp("Exiting");
    exit(1);
end
saveProfile(cirrus_cluster);
clear cirrus_cluster;

n = 3;
A = 3000;
a=zeros(A,A,n);
b=1:n;
parpool;
tic
parfor i = 1:n
    a(:,:,i) = rand(A);
end
toc
tic
parfor i = 1:n
    b(i) = max(abs(eig(a(:,:,i))));
end
toc
```

Note that `PreferredNumWorkers`, `NumWorkers` and `NumThreads` persist between MATLAB sessions but will be updated correctly if you use the wrapper each time.

`NumWorkers` and `NumThreads` can be changed (using `parcluster` and `saveProfile`) but `NumWorkers * NumThreads` should be less than or equal to the number of cores (`ncores` above). If you wish a worker to run a threaded routine in
serial, you must set \texttt{NumThreads} to 1 (the default).

If you specify exclusive node access, then all the cores and memory will be available. On the login nodes, a single core is used and memory is not limited.

### 21.5 MATLAB 2019 versions

There has been a change of configuration options for MATLAB from version R2019 and onwards that means the \texttt{-r} flag has been replaced with the \texttt{-batch} flag. To accommodate that a new job wrapper script is required to run applications. For these versions of MATLAB, if you need to use the \texttt{-r} or \texttt{-batch} flag replace this line in your PBS script, i.e.:

```bash
matlab_wrapper -nodisplay -nodesktop -batch "main_simulated_data_FINAL_clean("$\text{ind}$","$\gamma$","$\text{rw}$","$\text{PBS\_JOBID}$")"
```

with:

```bash
matlab_wrapper_2019 -nodisplay -nodesktop -batch "main_simulated_data_FINAL_clean("
"$\text{ind}$","$\gamma$","$\text{rw}$","$\text{PBS\_JOBID}$")"
```

and this should allow scripts to run normally.

### 21.6 Running parallel MATLAB jobs using MDCS

It is possible to use MATLAB on the login node to set up an MDCS PBSPro cluster profile and then launch jobs using that profile. However, this does not give per-job control of the number of cores and walltime; these are set once in the profile.

Raymond Norris from MathWorks has provided a configuration script that gives a much more flexible MDCS profile and we recommend you use this method. Instructions for using this profile are given below.

This MDCS profile can be used in MATLAB on the login node - the MDCS computations are done in PBS jobs launched using the profile.

#### 21.6.1 Configuration

Start MATLAB on the login node. Configure MATLAB to run parallel jobs on your cluster by calling \texttt{configCluster}.

For each cluster, \texttt{configCluster} only needs to be called once per version of MATLAB

```bash
configCluster
```

Jobs will now default to the cluster rather than submit to the local machine (the login node in this case).

#### 21.6.2 Configuring jobs

Prior to submitting the job, you can specify various parameters to pass to our jobs, such as walltime, e-mail, etc. Other than \texttt{ProjectCode} and \texttt{WallTime}, none of these are required to be set.

NOTE: Any parameters specified using this workflow will be persistent between MATLAB sessions
% Get a handle to the cluster.
c = parcluster('cirrus');

% Assign the project code for the job.  **[REQUIRED]**
c.AdditionalProperties.ProjectCode = 'project-code';

% Specify the walltime (e.g. 5 hours).  **[REQUIRED]**
c.AdditionalProperties.WallTime = '05:00:00';

% Specify e-mail address to receive notifications about your job.
c.AdditionalProperties.EmailAddress = 'your_name@your_address';

% Request a specific reservation to run your job.  It is better to
% use the queues rather than a reservation.
c.AdditionalProperties.Reservation = 'your-reservation';

% Set the job placement (e.g., pack, excl, scatter:excl).
% Usually the default of free is what you want.
c.AdditionalProperties.JobPlacement = 'pack';

% Request to run in a particular queue.  Usually the default (no
% specific queue requested) will route the job to the correct queue.
c.AdditionalProperties.QueueName = 'queue-name';

% If you are using GPUs, request up to 4 GPUs per node (this will
% override a requested queue name and will use the 'gpu' queue).
c.AdditionalProperties.GpusPerNode = 4;

Save changes after modifying AdditionalProperties fields

c.saveProfile

To see the values of the current configuration options, call the specific AdditionalProperties name

c.AdditionalProperties

To clear a value, assign the property an empty value (’, [], or false)

% Turn off email notifications.
c.AdditionalProperties.EmailAddress = '';

21.6.3 Interactive jobs

To run an interactive pool job on the cluster, use parpool as before. configCluster sets NumWorkers to 32 in the cluster to match the number of MDCS workers available in our TAH licence. If you have your own MDCS licence, you can change this by setting c.NumWorkers and saving the profile.

% Open a pool of 32 workers on the cluster.
p = parpool('cirrus',32);

Rather than running locally on one compute node machine, this pool can run across multiple nodes on the cluster

% Run a parfor over 1000 iterations.
parfor idz = 1:1000
  a(idz) = ...
end
Once you have finished using the pool, delete it

```matlab
% Delete the pool
p.delete
```

### 21.6.4 Serial jobs

Rather than running interactively, use the `batch` command to submit asynchronous jobs to the cluster. This is generally more useful on Cirrus, which usually has long queues. The `batch` command will return a job object which is used to access the output of the submitted job. See the MATLAB documentation for more help on `batch`.

```matlab
% Get a handle to the cluster.
c = parcluster('cirrus');

% Submit job to query where MATLAB is running on the cluster.
j = c.batch(@pwd, 1, {});

% Query job for state.
j.State

% If state is finished, fetch results.
j.fetchOutputs{:}

% Delete the job after results are no longer needed.
j.delete
```

To retrieve a list of currently running or completed jobs, call `parcluster` to retrieve the cluster object. The cluster object stores an array of jobs that were run, are running, or are queued to run. This allows you to fetch the results of completed jobs. Retrieve and view the list of jobs as shown below.

```matlab
c = parcluster('cirrus');
jobs = c.Jobs
```

Once you have identified the job you want, you can retrieve the results as you have done previously. `fetchOutputs` is used to retrieve function output arguments; if using batch with a script, use `load` instead. Data that has been written to files on the cluster needs be retrieved directly from the file system.

To view results of a previously completed job

```matlab
% Get a handle on job with ID 2.
j2 = c.Jobs(2);

% Fetch results for job with ID 2.
j2.fetchOutputs{:}

% If the job produces an error, view the error log file.
c.getDebugLog(j.Tasks(1))
```

**NOTE:** You can view a list of your jobs, as well as their IDs, using the above `c.Jobs` command.

```matlab
% Fetch results for job with ID 2.
j2.fetchOutputs{:}

% If the job produces an error, view the error log file.
c.getDebugLog(j.Tasks(1))
```

**NOTE:** When submitting independent jobs, with multiple tasks, you will have to specify the task number.
21.6.5 Parallel jobs

Users can also submit parallel workflows with batch. You can use the following example (`parallel_example.m`) for a parallel job

```matlab
function t = parallel_example(iter)

    if nargin==0, iter = 16; end

    disp('Start sim')

    t0 = tic;
    parfor idx = 1:iter
        A(idx) = idx;
        pause(2);
    end
    t = toc(t0);

    disp('Sim completed.')

end
```

Use the `batch` command again, but since you are running a parallel job, you also specify a MATLAB Pool

```matlab
% Get a handle to the cluster.
c = parcluster('cirrus');

% Submit a batch pool job using 4 workers for 16 simulations.
j = c.batch(@parallel_example, 1, {}, 'Pool', 4);

% View current job status.
j.State

% Fetch the results after a finished state is retrieved.
j.fetchOutputs{:}

ans =

8.8872
```

The job ran in 8.89 seconds using 4 workers. Note that these jobs will always request N+1 CPU cores, since one worker is required to manage the batch job and pool of workers. For example, a job that needs eight workers will consume nine CPU cores. With a MDCS licence for 32 workers, you will be able to have a pool of 31 workers.

Run the same simulation but increase the Pool size. This time, to retrieve the results later, keep track of the job ID.

NOTE: For some applications, there will be a diminishing return when allocating too many workers, as the overhead may exceed computation time.

```matlab
% Get a handle to the cluster.
c = parcluster('cirrus');

% Submit a batch pool job using 8 workers for 16 simulations.
j = c.batch(@parallel_example, 1, {}, 'Pool', 8);

% Get the job ID
id = j.ID

Id =

(continues on next page)
% Clear workspace, as though you have quit MATLAB.
clear j

Once you have a handle to the cluster, call the findJob method to search for the job with the specified job ID

```matlab
% Get a handle to the cluster.
c = parcluster('cirrus');

% Find the old job
j = c.findJob('ID', 4);

% Retrieve the state of the job.
j.State
ans
finished

% Fetch the results.
j.fetchOutputs{1};
ans =
4.7270

% If necessary, retrieve an output/error log file.
c.getDebugLog(j)
```

The job now runs 4.73 seconds using 8 workers. Run code with different number of workers to determine the ideal number to use.

Alternatively, to retrieve job results via a graphical user interface, use the Job Monitor (Parallel > Monitor Jobs).
21.6.6 Debugging

If a serial job produces an error, you can call the `getDebugLog` method to view the error log file

```matlab
j.Parent.getDebugLog(j.Tasks(1))
```

When submitting independent jobs, with multiple tasks, you will have to specify the task number. For Pool jobs, do not dereference into the job object

```matlab
j.Parent.getDebugLog(j)
```

The scheduler ID can be derived by calling `schedID`

```matlab
schedID(j)
an
25539
```

21.6.7 To learn more

To learn more about the MATLAB Parallel Computing Toolbox, check out these resources:

- Parallel Computing Coding Examples
- Parallel Computing Documentation
- Parallel Computing Overview
- Parallel Computing Tutorials
- Parallel Computing Videos
21.7 GPUs

Calculations using GPUs can be done using the *GPU nodes*. This can be done using MATLAB within a PBS job script, similar to *using the local cluster*, or can be done using the *MDCS profile*. The GPUs are shared unless you request exclusive access to the node (4 GPUs), so you may find that you share a GPU with another user.
Molpro is a comprehensive system of ab initio programs for advanced molecular electronic structure calculations, designed and maintained by H.-J. Werner and P. J. Knowles, and containing contributions from many other authors. It comprises efficient and well parallelized programs for standard computational chemistry applications, such as DFT with a large choice of functionals, as well as state-of-the art high-level coupled-cluster and multi-reference wave function methods.

22.1 Useful Links

- Molpro User Guides
- Molpro Licensing

22.2 Using Molpro on Cirrus

In order to use the Molpro binaries on Cirrus you must possess a valid Molpro licence key. Without a key you will be able to access the binaries but will not be able to run any calculations.

22.3 Running

To run Molpro you need to add the correct module to your environment; specify your licence key using the MOLPRO_KEY environment variable and make sure you specify the location for the temporary files using the TMPDIR environment variable. You can load the default Molpro module with:

```
module add molpro
```

Once you have loaded the module, the Molpro executables are available in your PATH.
22.4 Example Job Submission Script

An example Molpro job submission script is shown below.

```bash
#!/bin/bash --login
#PBS -N molpro_job
#PBS -l select=1:ncpus=36
#PBS -l walltime=1:0:0
#PBS -l place=scatter:excl

# Replace "budget" with your budget code in the line below
#PBS -A budget

# Move to directory that script was submitted from
export PBS_O_WORKDIR=$(readlink -f $PBS_O_WORKDIR)
cd $PBS_O_WORKDIR

# Load the molpro module
module add molpro

# Replace this with the value of your Molpro licence key
export MOLPRO_KEY="...your Molpro key...

# Make sure temporary files are on the /work filesystem
export TMPDIR=$PBS_O_WORKDIR

# Run Molpro using the input my_file.inp
# Requested 1 node above = 36 cores
molpro -n 36 my_file.inp
```
NAMD, recipient of a 2002 Gordon Bell Award and a 2012 Sidney Fernbach Award, is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. Based on Charm++ parallel objects, NAMD scales to hundreds of cores for typical simulations and beyond 500,000 cores for the largest simulations. NAMD uses the popular molecular graphics program VMD for simulation setup and trajectory analysis, but is also file-compatible with AMBER, CHARMM, and X-PLOR.

23.1 Useful Links

- NAMD User Guide
- NAMD Tutorials

23.2 Using NAMD on Cirrus

NAMD is freely available to all Cirrus users.

23.3 Running parallel NAMD jobs

NAMD can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node. For example, the following script will run a NAMD MD job using 4 nodes (144 cores) with pure MPI.

```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N namd_test
#PBS -l select=4:ncpus=36
# Make sure you are not sharing nodes with other users
```

(continues on next page)
#PBS -l place=scatter:excl
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load NAMD module
module load namd

# Run using input in input.namd
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
# Note: NAMD uses Intel MPI so mpirun should be used instead of
# mpiexec_mpt (which is SGI MPI)
mpirun -ppn 36 -n 144 namd2 input.namd
OpenFOAM is an open-source toolbox for computational fluid dynamics. OpenFOAM consists of generic tools to simulate complex physics for a variety of fields of interest, from fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics, electromagnetism and the pricing of financial options.

The core technology of OpenFOAM is a flexible set of modules written in C++. These are used to build solvers and utilities to perform pre- and post-processing tasks ranging from simple data manipulation to visualisation and mesh processing.

### 24.1 Available Versions

You can query the versions of OpenFOAM available on Cirrus from the command line with `module avail openfoam`.

We currently have *OpenFOAM v1706* available on Cirrus.

### 24.2 Useful Links

- OpenFOAM Documentation

### 24.3 Using OpenFOAM on Cirrus

To use OpenFOAM on Cirrus you should first load the OpenFOAM module:

```bash
module add openfoam
```

After that you need to source the `etc/bashrc` file provided by OpenFOAM:

```bash
source $OPENFOAM_CURPATH/etc/bashrc
```
You should then be able to use OpenFOAM. The above commands will also need to be added to any job/batch submission scripts you want to run OpenFOAM from.
Quantum Espresso (QE)

Quantum Espresso is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

25.1 Useful Links

- QE User Guides
- QE Tutorials

25.2 Using QE on Cirrus

QE is Open Source software and is freely available to all Cirrus users.

25.3 Running parallel QE jobs

QE can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node.

For example, the following script will run a QE pw.x job using 4 nodes (144 cores).

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N pw_test
#PBS -l select=4:ncpus=36
#PBS -1 place=scatter:excl
#PBS -1 walltime=0:20:0
```
# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Load QE and MPI modules
module load qe
module load mpt

# Run using input in test_calc.in
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
mpiexec_mpt -ppn 36 -n 144 pw.x test_calc.in
STAR-CCM+ is a computational fluid dynamics (CFD) code and beyond. It provides a broad range of validated models to simulate disciplines and physics including CFD, computational solid mechanics (CSM), electromagnetics, heat transfer, multiphase flow, particle dynamics, reacting flow, electrochemistry, aero-acoustics and rheology; the simulation of rigid and flexible body motions with techniques including mesh morphing, overset mesh and six degrees of freedom (6DOF) motion; and the ability to combine and account for the interaction between the various physics and motion models in a single simulation to cover your specific application.

### 26.1 Useful Links

- Information about STAR-CCM+ by Siemens

### 26.2 Licensing

All users must provide their own licence for STAR-CCM+. This licence can be provided as:

1. FLEXIm licence key to be installed on Cirrus

2. IP address and port of publicly accessible remote licence (your STAR-CCM+ licence server must use the same ports as our Licence Server Gateway: this is achieved by simply setting an environment variable) 3. Power on Demand (PoD) (nothing needs to be provided to Cirrus in this case)

For options 1 and 2, you should contact the Cirrus Helpdesk with the relevant details.

### 26.3 Using STAR-CCM+ on Cirrus: Interactive remote GUI Mode

A fast and responsive way of running with a GUI is to install STAR-CCM+ on your local Windows(7 or 10) or Linux workstation. You can then start your local STAR-CCM+ and connect to Cirrus in order to submit new jobs or query the status of running jobs. When you install your local version, de-activate the FLEXIm installation. FLEXIm is not required, as you will either be using the FLEXIm server on Cirrus or the Power on Demand (PoD) licence option.
You also need to setup passwordless SSH connections to Cirrus.

### 26.3.1 Jobs using FLEXlm licence server on Cirrus

Before you can use the FLEXlm server on Cirrus, you must provide us with your licence key to install on Cirrus (see above).

You can then start the STAR-CCM+ server on the compute nodes. The following script starts the server:

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N STAR-CCM_test
#PBS -l select=14:ncpus=36
#PBS -l walltime=02:00:00
#PBS -l place=scatter:excl
#PBS -k oe

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory what the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load mpt
module load starccm+

export SGI_MPI_HOME=$MPI_ROOT

uniq $PBS_NODEFILE | cut -d . -f 1 > ~/starccm.launcher.host.txt
starccm+ -server -machinefile ~/starccm.launcher.host.txt -np 504 -rsh ssh -port 42333
```

The port number “42333” should be free. If it is not free STAR-CCM+ will return with an error. You must then try to use another random port in the 42XXX range. You can then use the ‘qstat’ command to find out the first node of your application.

### 26.3.2 Jobs using remote licence server

The documentation for this option is currently under construction.

### 26.3.3 Jobs using Power on Demand (PoD) licences

You can then start the STAR-CCM+ server on the compute nodes. The following script starts the server:

```bash
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N STAR-CCM_test
#PBS -l select=14:ncpus=36
#PBS -l walltime=02:00:00
#PBS -l place=scatter:excl
#PBS -k oe

# Replace [budget code] below with your project code (e.g. t01)
```

(continues on next page)
#PBS -A [budget code]

# Change to the directory what the job was submitted from
cd $PBS_O_WORKDIR

# Load any required modules
module load mpt
module load starccm+

export SGI_MPI_HOME=$MPI_ROOT
export PATH=$STARCCM_EXE:$PATH
export LM_LICENSE_FILE=2999@192.168.191.10
export CDLMD_LICENSE_FILE=2999@192.168.191.10

uniq $PBS_NODEFILE | cut -d . -f 1 > ~/starccm.launcher.host.txt
starccm+ -power -podkey <PODkey> -licpath 2999@192.168.191.10 -server -machinefile ~/starccm.launcher.host.txt -np 504 -rsh ssh -port 42333

You should replace “<PODkey>” with your PoD licence key.

### 26.3.4 Local Star-CCM+ client configuration

Start your local STAR-CCM+ application and connect to your server. Click on the File -> “Connect to Server...” option and use the following settings:

- **Host**: name of first Cirrus compute node (use ‘qtsat’, e.g. r1i0n32)
- **Port**: the number that you specified in the submission script

Select the “Connect through SSH tunnel” option and use:

- **SSH Tunnel Host**: cirrus-login0.epcc.ed.ac.uk
- **SSH Tunnel Host Username**: use your Cirrus username
- **SSH Options**: -agent

Your local STAR-CCM+ client should now be connected to the remote server. You should be able to run a new simulation or interact with an existing one.
The Vienna Ab initio Simulation Package (VASP) is a computer program for atomic scale materials modelling, e.g. electronic structure calculations and quantum-mechanical molecular dynamics, from first principles.

VASP computes an approximate solution to the many-body Schrödinger equation, either within density functional theory (DFT), solving the Kohn-Sham equations, or within the Hartree-Fock (HF) approximation, solving the Roothaan equations. Hybrid functionals that mix the Hartree-Fock approach with density functional theory are implemented as well. Furthermore, Green’s functions methods (GW quasiparticles, and ACFDT-RPA) and many-body perturbation theory (2nd-order Møller-Plesset) are available in VASP.

In VASP, central quantities, like the one-electron orbitals, the electronic charge density, and the local potential are expressed in plane wave basis sets. The interactions between the electrons and ions are described using norm-conserving or ultrasoft pseudopotentials, or the projector-augmented-wave method.

To determine the electronic groundstate, VASP makes use of efficient iterative matrix diagonalisation techniques, like the residual minimisation method with direct inversion of the iterative subspace (RMM-DIIS) or blocked Davidson algorithms. These are coupled to highly efficient Broyden and Pulay density mixing schemes to speed up the self-consistency cycle.

### 27.1 Useful Links

- VASP Manual
- VASP Licensing

### 27.2 Using VASP on Cirrus

VASP is only available to users who have a valid VASP licence.

If you have a VASP licence and wish to have access to VASP on Cirrus please contact the Cirrus Helpdesk.
27.3 Running parallel VASP jobs

VASP can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node.

To access VASP you should load the `vasp` module in your job submission scripts:

```
module add vasp
```

Once loaded, the executables are called:

- `vasp_std` - Multiple k-point version
- `vasp_gam` - GAMMA-point only version
- `vasp_ncl` - Non-collinear version

All 5.4.* executables include the additional MD algorithms accessed via the `MDALGO` keyword.

You can access the LDA and PBE pseudopotentials for VASP on Cirrus at:

```
/lustre/home/y07/vasp5/5.4.4-intel17-mpt214/pot
```

The following script will run a VASP job using 4 nodes (144 cores).

```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N VASP_test
#PBS -l select=4:ncpus=36
#PBS -l place=scatter:excl
#PBS -l walltime=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Change to the directory that the job was submitted from
#PBS_O_WORKDIR

# Load VASP module
module load vasp

# Run standard VASP executable
# Note: '-ppn 36' is required to use all physical cores across
# nodes as hyperthreading is enabled by default
mpiexec_mpt -ppn 36 -n 144 vasp_std
```
The Intel MKL libraries contain a variety of optimised numerical libraries including BLAS, LAPACK, and ScaLAPACK.

### 28.1 Intel Compilers

#### 28.1.1 BLAS and LAPACK

To use MKL libraries with the Intel compilers you first need to load the Intel compiler module and the Intel tools module:

```bash
module load intel-compilers-17
module load intel-tools-17
```

To include MKL you specify the `-mkl` option on your compile and link lines. For example, to compile a single source file, Fortran program with MKL you could use:

```bash
ifort -c -mkl -o lapack_prb.o lapack_prb.f90
ifort -o lapack_prb.o
```

The `-mkl` flag without any options builds against the threaded version of MKL. If you wish to build against the serial version of MKL, you would use `-mkl=sequential`.

#### 28.1.2 ScaLAPACK

The distributed memory linear algebra routines in ScaLAPACK require MPI in addition to the compilers and MKL libraries. On Cirrus, this is usually provided by SGI MPT.

```bash
module load intel-compilers-17
module load intel-tools-17
module load mpt
```
Once you have the modules loaded you need to use the SGI versions of BLACS at link time to include ScaLAPACK. Remember to use the MPI versions of the compilers:

```bash
mpif90 -c -o linsolve.o linsolve.f90
mpif90 -o linsolve.x linsolve.o -L$(MKLROOT)/lib/intel64 -lmkl_scalapack_lpm64 -lmkl_intel_lpm64 -lmkl_sequential -lmkl_core -lmkl_blacs_sgammt_lpm64 -lpthread -lm -ldl
```

### 28.2 GNU Compiler

#### 28.2.1 BLAS and LAPACK

To use MKL libraries with the GNU compiler you first need to load the GNU compiler module and Intel tools module:

```bash
module load gcc
module load intel-tools-16
```

To include MKL you need to explicitly link against the MKL libraries. For example, to compile a single source file, Fortran program with MKL you could use:

```bash
gfortran -c -o lapack_prb.o lapack_prb.f90
gfortran -o lapack_prb.x lapack_prb.o -L$MKLROOT/lib/intel64 -lmkl_gf_lpm64 -lmkl_core -lmkl_sequential
```

This will build against the serial version of MKL, to build against the threaded version use:

```bash
gfortran -c -o lapack_prb.o lapack_prb.f90
gfortran -fopenmp -o lapack_prb.x lapack_prb.o -L$MKLROOT/lib/intel64 -lmkl_gf_lpm64 -lmkl_core -lmkl_gnu_thread
```

#### 28.2.2 ScaLAPACK

The distributed memory linear algebra routines in ScaLAPACK require MPI in addition to the compilers and MKL libraries. On Cirrus, this is usually provided by SGI MPT.

```bash
module load gcc
module load intel-tools-16
module load mpt
```

Once you have the modules loaded you need to link against two additional libraries to include ScaLAPACK. Remember to use the MPI versions of the compilers:

```bash
mpif90 -f90=gfortran -c -o linsolve.o linsolve.f90
mpif90 -f90=gfortran -o linsolve.x linsolve.o -L$(MKLROOT)/lib/intel64 -lmkl_scalapack_lpm64 -lmkl_intel_lpm64 -lmkl_sequential -lmkl_core -lmkl_blacs_sgammt_lpm64 -lpthread -lm -ldl
```

#### 28.2.3 ILP vs LP libraries

If you look in the $MKLROOT/lib/intel64 directory then you will see ILP and LP libraries, in the above we were linking against the LP libraries and you can choose either. ILP use a 64 bit integer type, whereas LP use a 32 bit integer type. For very large arrays then ILP is the best choice (as it can index far more data), but there are some limitations. For more information see the Intel documentation here.
Allinea’s Forge tool suite is installed on Cirrus and DDT, which is a debugging tool for scalar, multi-threaded and large-scale parallel applications, is available for use in debugging your codes. To compile your code for debugging you will usually want to specify the `-O0` option to turn off all code optimisation (as this can produce a mismatch between source code line numbers and debugging information) and `-g` to include debugging information in the compiled executable. To use this package you will need to load the Allinea Forge module and execute `forge`:

```
module load allinea
forge
```

### 29.1 Debugging runs on the login nodes

You can execute and debug your MPI code on the login node which is useful for immediate development work with short, simple runs to avoid having to wait in the queue. Firstly ensure you have loaded the `mpt` library, then start `forge` and click `Run`. Fill in the necessary details of your code under the application pane, then check the MPI tick box, specify the number of MPI processes you wish to run and ensure implementation is set to `SGI MPT (2.10+)`. If this is not set correctly then you can update the configuration via clicking the `Change` button and selecting this option on the `MPI/UPC Implementation` field of the system pane. When you are happy with this hit `Run` to start.

### 29.2 Debugging runs on the compute nodes

This involves DDT submitting your job to the queue, this then running and as soon as the compute nodes start executing you will drop into the debug session and be able to interact with your code. Similarly to running on the login node, fill in details of your application and ensure that MPI is ticked. But now change the implementation from `SGI MPT (2.10+) to SGI MPT (2.10+, batch)` as we are running via the batch system. Then check the `Submit to Queue` tick box and click the `Configure` button. In the settings window that pops up you can specify the submission template, one has been prepared one for Cirrus at `/lustre/sw/allinea/forge-7.0.0/templates/cirrus.qtf` which we suggest you use - although you are very free to chose another one and/or specialise this as you require. Back on the run page, click the `Parameters` button and fill in the maximum wallclock time, the budget to charge to and the total number of virtual cores required which determine the number of nodes and are provided as an argument to
the `-l select=` PBS option. Back on the `run` dialog ensure look at the `MPI` pane, ensure the `number of processes` and `processes per node` settings are correct and then hit `Submit`.

### 29.3 Memory debugging with DDT

If you are dynamically linking your code and debugging it on the login node then this is fine (just ensure that the `Preload the memory debugging library` option is ticked in the `Details` pane.) If you are dynamically linking but intending to debug running on the compute nodes, or statically linking then you need to include the compile option `-Wl, --allow-multiple-definition` and explicitly link your executable with Allinea’s memory debugging library. The exactly library to link against depends on your code; `-ldmalloc` (for no threading with C), `-ldmallocth` (for threading with C), `-ldmallocxx` (for no threading with C++) or `-ldmallocthcxx` (for threading with C++). The library locations are all set up when the `allinea` module is loaded so these libraries should be found without further arguments.

### 29.4 Getting further help on DDT

- DDT website
- DDT support page
- DDT user guide