Cirrus Documentation

Release 1.2

EPCC

Nov 24, 2022

CIRRUS USER GUIDE

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 38 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 (Cascade Lake) 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. 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

ONE

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

TWO

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.

Before following the process below, we assume you have set up an account on Cirrus through the EPCC SAFE. Documentation on how to do this can be found at:

> SAFE Guide for Users

This section covers the basic connection methods.

2.1 Access credentials

To access Cirrus, you need to use two credentials: your password **and** an SSH key pair protected by a passphrase. You can find more detailed instructions on how to set up your credentials to access Cirrus from Windows, macOS and Linux below.

2.1.1 SSH Key Pairs

You will need to generate an SSH key pair protected by a passphrase to access Cirrus.

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:

```
$ 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 |
```

(continues on next page)

(continued from previous page)

```
|oE .   .         |
|o =     .   S    |
|.    +.+     .   |
|.  oo            |
|.  .             |
| ..              |
+-----------------+
```

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

2.1.2 Upload public part of key pair to SAFE

You should now upload the public part of your SSH key pair to the SAFE by following the instructions at:

Login to SAFE. Then:

1. Go to the Menu *Login accounts* and select the Cirrus account you want to add the SSH key to
2. On the subsequent Login account details page click the *Add Credential* button
3. Select *SSH public key* as the Credential Type and click *Next*
4. Either copy and paste the public part of your SSH key into the *SSH Public key* box or use the button to select the public key file on your computer.
5. Click *Add* to associate the public SSH key part with your account

Once you have done this, your SSH key will be added to your Cirrus account.

Remember, you will need to use both an SSH key and password to log into Cirrus so you will also need to collect your initial password before you can log into Cirrus. We cover this next.

2.1.3 Initial passwords

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

2.1.4 Changing passwords

You may now change your password on the Cirrus machine itself using the *passwd* command or when you are prompted the first time you login. 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.

Note: When you first log into Cirrus, you will be prompted to change your initial password. This is a three step process:

1. When promoted to enter your *ldap password*: Re-enter the password you retrieved from SAFE
2. When prompted to enter your new password: type in a new password
3. When prompted to re-enter the new password: re-enter the new password

Your password has now been changed

2.1.5 Password Expiration

Passwords on Cirrus will expire after two years. When this happens, you will be required to update your password. This will be done by following the same steps as above.

Note: You will still be prompted to enter your current password first before changing your 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 MacOS

Linux distributions and MacOS 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.

MacOS 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
```

You will first be prompted for the passphrase associated with your SSH key pair. Once you have entered your passphrase successfully, you will then be prompted for your password. You need to enter both correctly to be able to access Cirrus.

Note: If your SSH key pair is not stored in the default location (usually `~/.ssh/id_rsa`) on your local system, you may need to specify the path to the private part of the key with the `-i` option to `ssh`. For example, if your key is in a file called `keys/id_rsa_cirrus` you would use the command `ssh -i keys/id_rsa_cirrus username@login.cirrus.ac.uk` to log in.

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 MacOS systems do not have an X window system. Users should install the XQuartz package to allow for SSH with X11 forwarding on MacOS 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 the SSH configuration file

Typing in the full command to login or transfer data to Cirrus can become tedious as it often has to be repeated many times. You can use the SSH configuration file, usually located on your local machine at `.ssh/config` to make things a bit more convenient.

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 username
```

(remember to replace `username` with your actual username!).

The `Host cirrus` line defines a short name for the entry. In this case, instead of typing `ssh username@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)

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

```
-bash-4.1$ ssh cirrus
```

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. You may find the `IdentityFile` option useful if you have to manage multiple SSH key pairs for different systems as this allows you to specify which SSH key to use for each system.

Note: There is a known bug with Windows ssh-agent. If you get the error message: `Warning: agent returned different signature type ssh-rsa (expected rsa-sha2-512)`, you will need to either specify the path to your ssh key in the command line (using the `-i` option as described above) or add the path to your SSH config file by using the `IdentityFile` option.

2.4 Accessing Cirrus from more than 1 machine

It is common for users to want to access Cirrus from more than one local machine (e.g. a desktop linux, and a laptop) - this can be achieved through use of an `~/.ssh/authorized_keys` file on Cirrus to hold the additional keys you generate. Note that if you want to access Cirrus via another remote service, see the next section, SSH forwarding.

You need to consider one of your local machines as your primary machine - this is the machine you should connect to Cirrus with using the instructions further up this page, adding your public key to SAFE.

On your second local machine, generate a new SSH key pair. Copy the public key to your primary machine (e.g. by email, USB stick, or cloud storage); the default location for this on a Linux or MacOS machine will be `~/.ssh/id_rsa.pub`. If you are a Windows user using MobaXTerm, you should export the public key it generates to OpenSSH format (`Conversions > Export OpenSSH Key`). You should never move the private key off the machine on which it was generated.

Once back on your primary machine, you should copy the public key from your secondary machine to Cirrus using:

```
scp id_rsa.pub <user>@login.cirrus.ac.uk:id_secondary.pub
```

You should then log into Cirrus, as normal: `ssh <user>@login.cirrus.ac.uk`, and then:

- check to see if the `.ssh` directory exists, using `ls -la ~`
- if it doesn't, create it, and apply appropriate permissions:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
```

- and then create an authorized_keys file, and add the public key from your secondary machine in one go:

```
cat ~/id_secondary.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
rm ~/id_secondary.pub
```

You can then repeat this process for any more local machines you want to access Cirrus from, omitting the `mkdir` and `chmod` lines as the relevant files and directories will already exist with the correct permissions. You don't need to add the public key from your primary machine in your *authorized_keys* file, because Cirrus can find this in SAFE.

Note that the permissions on the `.ssh` directory must be set to 700 (Owner can read, can write and can execute but group and world do not have access) and on the `authorized_keys` file must be 600 (Owner can read and write but group and world do not have access). Keys will be ignored if this is not the case.

2.5 SSH forwarding (to use Cirrus from a second remote machine)

If you want to access Cirrus from a machine you already access remotely (e.g. to copy data from Cirrus onto a different cluster), you can *forward* your local Cirrus SSH keys so that you don't need to create a new key pair on the intermediate machine.

If your local machine is MacOS or Linus, add your Cirrus SSH key to the SSH Agent:

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

(If you created your key with a different name, replace `id_rsa` in the command with the name of your private key file). You will be prompted for your SSH key's passphrase.

You can then use the `-A` flag when connecting to your intermediate cluster:

```
ssh -A <user>@<host>
```

Once on the intermediate cluster, you should be able to SSH to Cirrus directly:

```
ssh <user>@login.cirrus.ac.uk
```

2.6 SSH debugging tips

If you find you are unable to connect via SSH there are a number of ways you can try and diagnose the issue. Some of these are collected below - if you are having difficulties connecting we suggest trying these before contacting the Cirrus service desk.

2.6.1 Can you connect to the login node?

Try the command `ping -c 3 login.cirrus.ac.uk`. If you successfully connect to the login node, the output should include:

```
--- login.dyn.cirrus.ac.uk ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 38ms
```

(the ping time '38ms' is not important). If not all packets are received there could be a problem with your internet connection, or the login node could be unavailable.

2.6.2 SSH key

If you get the error message `Permission denied (publickey)` this can indicate a problem with your SSH key. Some things to check:

- Have you uploaded the key to SAFE? Please note that if the same key is reuploaded SAFE will not map the "new" key to cirrus. If for some reason this is required, please delete the key first, then reupload.
- Is ssh using the correct key? You can check which keys are being found and offered by ssh using `ssh -vvv`. If your private key has a non-default name you can use the `-i` flag to provide it to ssh, i.e. `ssh -i path/to/key username@login.cirrus.ac.uk`.
- Are you entering the passphrase correctly? You will be asked for your private key's passphrase first. If you enter it incorrectly you will usually be asked to enter it again, and usually up to three times in total, after which ssh will fail with `Permission denied (publickey)`. If you would like to confirm your passphrase without attempting to connect, you can use `ssh-keygen -y -f /path/to/private/key`. If successful, this command will print the corresponding public key. You can also use this to check it is the one uploaded to SAFE.
- Are permissions correct on the ssh key? One common issue is that the permissions are incorrect on the either the key file, or the directory it's contained in. On Linux/MacOS for example, if your private keys are held in `~/.ssh/` you can check this with `ls -al ~/.ssh`. This should give something similar to the following output:

```
$ ls -al ~/.ssh/
drwx------.  2 user group    48 Jul 15 20:24 .
drwx------. 12 user group  4096 Oct 13 12:11 ..
-rw-------.  1 user group   113 Jul 15 20:23 authorized_keys
-rw-------.  1 user group 12686 Jul 15 20:23 id_rsa
-rw-r--r--.  1 user group  2785 Jul 15 20:23 id_rsa.pub
-rw-r--r--.  1 user group  1967 Oct 13 14:11 known_hosts
```

The important section here is the string of letters and dashes at the start, for the lines ending in `.`, `id_rsa`, and `id_rsa.pub`, which indicate permissions on the containing directory, private key, and public key respectively. If your permissions are not correct, they can be set with `chmod`. Consult the table below for the relevant `chmod` command. On Windows, permissions are handled differently but can be set by right-clicking on the file and selecting Properties > Security > Advanced. The user, SYSTEM, and Administrators should have `Full control`, and no other permissions should exist for both public and private key files, and the containing folder.

Target	Permissions	`chmod` Code
Directory	`drwx------`	700
Private Key	`-rw-------`	600
Public Key	`-rw-r--r--`	644

`chmod` can be used to set permissions on the target in the following way: `chmod <code> <target>`. So for example to set correct permissions on the private key file `id_rsa_cirrus` one would use the command `chmod 600 id_rsa_cirrus`.

Note: Unix file permissions can be understood in the following way. There are three groups that can have file permissions: (owning) *users*, (owning) *groups*, and *others*. The available permissions are *read*, *write*, and *execute*. The first character indicates whether the target is a file `-`, or directory `d`. The next three characters indicate the owning user's permissions. The first character is `r` if they have read permission, `-` if they don't, the second character is `w` if they have write permission, `-` if they don't, the third character is `x` if they have execute permission, `-` if they don't. This pattern is then repeated for *group*, and *other* permissions. For example the pattern `-rw-r--r--` indicates that the owning user can read and write the file, members of the owning group can read it, and anyone else can also read it. The `chmod` codes are constructed by treating the user, group, and owner permission strings as binary numbers, then converting them to decimal. For example the permission string `-rwx------` becomes `111 000 000 -> 700`.

2.6.3 Password

If you are having trouble entering your password consider using a password manager, from which you can copy and paste it. This will also help you generate a secure password. If you need to reset your password, instructions for doing so can be found here.

Windows users please note that `Ctrl+V` does not work to paste in to PuTTY, MobaXterm, or PowerShell. Instead use `Shift+Ins` to paste. Alternatively, right-click and select 'Paste' in PuTTY and MobaXterm, or simply right-click to paste in PowerShell.

2.6.4 SSH verbose output

Verbose debugging output from `ssh` can be very useful for diagnosing the issue. In particular, it can be used to distinguish between problems with the SSH key and password - further details are given below. To enable verbose output add the `-vvv` flag to your SSH command. For example:

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

The output is lengthy, but somewhere in there you should see lines similar to the following:

```
debug1: Next authentication method: publickey
debug1: Offering public key: RSA SHA256:<key-hash> <path_to_private_key>
debug3: send_pubkey_test
debug3: send packet: type 50
```

(continues on next page)

(continued from previous page)

```
debug2: we sent a publickey packet, wait for reply
debug3: receive packet: type 60
debug1: Server accepts key: pkalg ssh-rsa vlen 2071
debug2: input_userauth_pk_ok: fp SHA256:<key-hash>
debug3: sign_and_send_pubkey: RSA SHA256:<key-hash>
Enter passphrase for key '<path_to_private_key>':
debug3: send packet: type 50
debug3: receive packet: type 51
Authenticated with partial success.
```

Most importantly, you can see which files ssh has checked for private keys, and you can see if any key is accepted. The line `Authenticated with partial success` indicates that the SSH key has been accepted, and you will next be asked for your password. By default ssh will go through a list of standard private key files, as well as any you have specified with `-i` or a config file. This is fine, as long as one of the files mentioned is the one that matches the public key uploaded to SAFE.

If you do not see `Authenticated with partial success` anywhere in the verbose output, consider the suggestions under *SSH key* above. If you do, but are unable to connect, consider the suggestions under *Password* above.

The equivalent information can be obtained in PuTTY or MobaXterm by enabling all logging in settings.

CHAPTER

THREE

DATA MANAGEMENT AND TRANSFER

This section covers the storage and file systems available on the system; the different ways that you can transfer data to and from Cirrus; and how to transfer backed up data from prior to the March 2022 Cirrus upgrade.

In all cases of data transfer, users should use the Cirrus login nodes.

3.1 Cirrus file systems and storage

The Cirrus service, like many HPC systems, has a complex structure. There are a number of different data storage types available to users:

- Home file system
- Work file systems
- Solid state storage

Each type of storage has different characteristics and policies, and is suitable for different types of use.

There are also two different types of node available to users:

- Login nodes
- Compute nodes

Each type of node sees a different combination of the storage types. The following table shows which storage options are available on different node types:

Storage	Login nodes	Compute nodes	Notes
Home	yes	no	No backup
Work	yes	yes	No backup
Solid state	yes	yes	No backup

3.1.1 Home file system

Every project has an allocation on the home file system and your project's space can always be accessed via the path `/home/[project-code]`. The home file system is approximately 1.5 PB in size and is implemented using the Ceph technology. This means that this storage is not particularly high performance but are well suited to standard operations like compilation and file editing. This file systems is visible from the Cirrus login nodes.

There are currently no backups of any data on the home file system.

Quotas on home file system

All projects are assigned a quota on the home file system. The project PI or manager can split this quota up between groups of users if they wish.

You can view any home file system quotas that apply to your account by logging into SAFE and navigating to the page for your Cirrus login account.

1. Log into SAFE
2. Use the "Login accounts" menu and select your Cirrus login account
3. The "Login account details" table lists any user or group quotas that are linked with your account. (If there is no quota shown for a row then you have an unlimited quota for that item, but you may still may be limited by another quota.)

Quota and usage data on SAFE is updated twice daily so may not be exactly up to date with the situation on the system itself.

From the command line

Some useful information on the current contents of directories on the `/home` file system is available from the command line by using the Ceph command `getfattr`. This is to be preferred over standard Unix commands such as `du` for reasons of efficiency.

For example, the number of entries (files plus directories) in a home directory can be queried via

```
$ cd
$ getfattr -n ceph.dir.entries .
# file: .
ceph.dir.entries="33"
```

The corresponding attribute `rentries` gives the recursive total in all subdirectories, that is, the total number of files and directories:

```
$ getfattr -n ceph.dir.rentries .
# file: .
ceph.dir.rentries="1619179"
```

Other useful attributes (all prefixed with `ceph.dir.`) include `files` which is the number of ordinary files, `subdirs` the number of subdirectories, and `bytes` the total number of bytes used. All these have a corresponding recursive version, respectively: `rfiles`, `rsubdirs`, and `rbytes`.

A full path name can be specified if required.

3.1.2 Work file system

Every project has an allocation on the work file system and your project's space can always be accessed via the path `/work/[project-code]`. The work file system is approximately 400 TB in size and is implemented using the Lustre parallel file system technology. They are designed to support data in large files. The performance for data stored in large numbers of small files is probably not going to be as good.

There are currently no backups of any data on the work file system.

Ideally, the work file system should only contain data that is:

- actively in use;

- recently generated and in the process of being saved elsewhere; or
- being made ready for up-coming work.

In practice it may be convenient to keep copies of datasets on the work file system that you know will be needed at a later date. However, make sure that important data is always backed up elsewhere and that your work would not be significantly impacted if the data on the work file system was lost.

If you have data on the work file system that you are not going to need in the future please delete it.

Quotas on the work file system

Tip: The capacity of the home file system is much larger than the work file system so you should store most data on home and only move data to work that you need for current running work.

As for the home file system, all projects are assigned a quota on the work file system. The project PI or manager can split this quota up between groups of users if they wish.

You can view any work file system quotas that apply to your account by logging into SAFE and navigating to the page for your Cirrus login account.

1. Log into SAFE
2. Use the "Login accounts" menu and select your Cirrus login account
3. The "Login account details" table lists any user or group quotas that are linked with your account. (If there is no quota shown for a row then you have an unlimited quota for that item, but you may still may be limited by another quota.)

Quota and usage data on SAFE is updated twice daily so may not be exactly up to date with the situation on the system itself.

You can also examine up to date quotas and usage on the Cirrus system itself using the `lfs quota` command. To do this:

Change directory to the work directory where you want to check the quota. For example, if I wanted to check the quota for user `auser` in project `t01` then I would:

```
[auser@cirrus-login1 auser]$ cd /work/t01/t01/auser

[auser@cirrus-login1 auser]$ lfs quota -hu auser .
Disk quotas for usr auser (uid 68826):
     Filesystem    used   quota   limit   grace   files   quota   limit   grace
               .  5.915G      0k      0k       -   51652       0       0       -
uid 68826 is using default block quota setting
uid 68826 is using default file quota setting
```

the quota and limit of 0k here indicate that no user quota is set for this user.

To check your project (group) quota, you would use the command:

```
[auser@cirrus-login1 auser]$ lfs quota -hg t01 .
Disk quotas for grp t01 (gid 37733):
     Filesystem    used   quota   limit   grace   files   quota   limit   grace
           .  958.3G      0k  13.57T       - 1427052       0       0       -
gid 37733 is using default file quota setting
```

the limit of `13.57T` indicates the quota for the group.

3.1.3 Solid state storage

More information on using the solid state storage can be found in the *Solid state storage* section of the user guide.

The solid state storage is not backed up.

3.2 Accessing Cirrus data from before March 2022

Prior to the March 2022 Cirrus upgrade,all user date on the `/lustre/sw` filesystem was archived. Users can access their archived data from the Cirrus login nodes in the `/home-archive` directory. Assuming you are user `auser` from project `x01`, your pre-rebuild archived data can be found in:

```
/home-archive/x01/auser
```

The data in the `/home-archive` file system is **read only** meaning that you will not be able to create, edit, or copy new information to this file system.

To make archived data visible from the compute nodes, you will need to copy the data from the `/home-archive` file system to the `/home` file system. Assuming again that you are user `auser` from project `x01` and that you were wanting to copy data from `/home-archive/x01/auser/directory_to_copy` to `/home/x01/x01/auser/destination_directory`, you would do this by running:

```
cp -r /home-archive/x01/auser/directory_to_copy \
   /home/x01/x01/auser/destination_directory
```

Note that the project code appears once in the path for the old home archive and twice in the path on the new /home file system.

Note: The capacity of the home file system is much larger than the work file system so you should move data to home rather than work.

3.3 Data transfer

3.3.1 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.

3.3.2 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 login 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 a *ssh-key* as documented in the User Guide at *Connecting to Cirrus*

3.3.3 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 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 assuming 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.

- Only transfer those files that are required. Consider which data you really need to keep.
- 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).
- Compress data before sending it, e.g. using gzip.

3.3.4 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@login.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@login.cirrus.ac.uk:[destination]
```

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

3.3.5 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@login.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.)

CHAPTER

FOUR

FILE AND RESOURCE MANAGEMENT

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://safe.epcc.ed.ac.uk).

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 selet *Report generator*.

4.3 Disk quotas

Disk quotas on Cirrus are managed via SAFE

For live disk usage figures on the Lustre `/work` file system, use

```
lfs quota -hu <username> /work

lfs quota -hg <groupname> /work
```

4.4 Backup policies

The `/home` file system is not backed up.

The `/work` file system is not backed up.

The solid-state storage `/scratch/space1` file system is not backed up.

We strongly advise that you keep copies of any critical data on on an alternative system that is fully backed up.

4.5 Sharing data with other Cirrus users

How you share data with other Cirrus users depends on whether or not they belong to the same project as you. Each project has two shared folders that can be used for sharing data.

4.5.1 Sharing data with Cirrus users in your project

Each project has an inner shared folder on the `/home` and `/work` filesystems:

```
/home/[project code]/[project code]/shared

/work/[project code]/[project code]/shared
```

This folder has read/write permissions for all project members. You can place any data you wish to share with other project members in this directory. For example, if your project code is `x01` the inner shared folder on the `/work` file system would be located at `/work/x01/x01/shared`.

4.5.2 Sharing data with all Cirrus users

Each project also has an outer shared folder on the `/home` and `/work` filesystems:

```
/home/[project code]/shared

/work/[project code]/shared
```

It is writable by all project members and readable by any user on the system. You can place any data you wish to share with other Cirrus users who are not members of your project in this directory. For example, if your project code is `x01` the outer shared folder on the `/work` file system would be located at `/work/x01/shared`.

4.6 File permissions and security

You should check the permissions of any files that you place in the shared area, especially if those files were created in your own Cirrus account. Files of the latter type are likely to be readable by you only.

The chmod command below shows how to make sure that a file placed in the outer shared folder is also readable by all Cirrus users.

```
chmod a+r /work/x01/shared/your-shared-file.txt
```

Similarly, for the inner shared folder, chmod can be called such that read permission is granted to all users within the x01 project.

```
chmod g+r /work/x01/x01/shared/your-shared-file.txt
```

If you're sharing a set of files stored within a folder hierarchy the chmod is slightly more complicated.

```
chmod -R a+Xr /work/x01/shared/my-shared-folder
chmod -R g+Xr /work/x01/x01/shared/my-shared-folder
```

The `-R` option ensures that the read permission is enabled recursively and the `+X` guarantees that the user(s) you're sharing the folder with can access the subdirectories below my-shared-folder.

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.

4.7 File types

4.7.1 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.

4.7.2 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/
- HDF: 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.

4.8 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.

4.9 Common I/O patterns

There is a number of I/O patterns that are frequently used in applications:

4.9.1 Single file, single writer (Serial I/O)

A common approach is to funnel all the I/O through a single master process. Although this has the advantage of producing a single file, the fact that only a single client is doing all the I/O means that it gains little benefit from the parallel file system.

4.9.2 File-per-process (FPP)

One of the first parallel strategies people use for I/O is for each parallel process to write to its own file. This is a simple scheme to implement and understand but has the disadvantage that, at the end of the calculation, the data is spread across many different files and may therefore be difficult to use for further analysis without a data reconstruction stage.

4.9.3 Single file, multiple writers without collective operations

There are a number of ways to achieve this. For example, many processes can open the same file but access different parts by skipping some initial offset; parallel I/O libraries such as MPI-IO, HDF5 and NetCDF also enable this.

Shared-file I/O has the advantage that all the data is organised correctly in a single file making analysis or restart more straightforward.

The problem is that, with many clients all accessing the same file, there can be a lot of contention for file system resources.

4.9.4 Single Shared File with collective writes (SSF)

The problem with having many clients performing I/O at the same time is that, to prevent them clashing with each other, the I/O library may have to take a conservative approach. For example, a file may be locked while each client is accessing it which means that I/O is effectively serialised and performance may be poor.

However, if I/O is done collectively where the library knows that all clients are doing I/O at the same time, then reads and writes can be explicitly coordinated to avoid clashes. It is only through collective I/O that the full bandwidth of the file system can be realised while accessing a single file.

4.10 Achieving efficient I/O

This section provides information on getting the best performance out of the `/work` parallel file system on Cirrus when writing data, particularly using parallel I/O patterns.

You may find that using the *Solid state storage* gives better performance than `/work` for some applications and IO patterns.

4.10.1 Lustre

The Cirrus `/work` file system use Lustre as a parallel file system technology. The Lustre file system provides POSIX semantics (changes on one node are immediately visible on other nodes) and can support very high data rates for appropriate I/O patterns.

4.10.2 Striping

One of the main factors leading to the high performance of `/work` Lustre file systems is the ability to stripe data across multiple Object Storage Targets (OSTs) in a round-robin fashion. Files are striped when the data is split up in chunks that will then be stored on different OSTs across the `/work` file system. Striping might improve the I/O performance because it increases the available bandwidth since multiple processes can read and write the same files simultaneously. However striping can also increase the overhead. Choosing the right striping configuration is key to obtain high performance results.

Users have control of a number of striping settings on Lustre file systems. Although these parameters can be set on a per-file basis they are usually set on directory where your output files will be written so that all output files inherit the settings.

Default configuration

The file system on Cirrus has the following default stripe settings:

- A default stripe count of 1
- A default stripe size of 1 MiB (1048576 bytes)

These settings have been chosen to provide a good compromise for the wide variety of I/O patterns that are seen on the system but are unlikely to be optimal for any one particular scenario. The Lustre command to query the stripe settings for a directory (or file) is `lfs getstripe`. For example, to query the stripe settings of an already created directory `res_dir`:

```
$ lfs getstripe res_dir/
res_dir
stripe_count:   1 stripe_size:    1048576 stripe_offset:  -1
```

Setting Custom Striping Configurations

Users can set stripe settings for a directory (or file) using the `lfs setstripe` command. The options for `lfs setstripe` are:

- `[--stripe-count|-c]` to set the stripe count; 0 means use the system default (usually 1) and -1 means stripe over all available OSTs.
- `[--stripe-size|-s]` to set the stripe size; 0 means use the system default (usually 1 MB) otherwise use k, m or g for KB, MB or GB respectively
- `[--stripe-index|-i]` to set the OST index (starting at 0) on which to start striping for this file. An index of -1 allows the MDS to choose the starting index and it is strongly recommended, as this allows space and load balancing to be done by the MDS as needed.

For example, to set a stripe size of 4 MiB for the existing directory `res_dir`, along with maximum striping count you would use:

```
$ lfs setstripe -s 4m -c -1 res_dir/
```

CHAPTER

FIVE

APPLICATION DEVELOPMENT ENVIRONMENT

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:

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

--------------------------------------- /lustre/sw/modulefiles --------------------------
↪-------------
intel-compilers-18/18.05.274  intel-compilers-19/19.0.0.117
```

If you want more info on any of the modules, you can use the `module help` command:

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

----------------------------------------------------------------------
Module Specific Help for /usr/share/Modules/modulefiles/mpt/2.25:

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, e.g.:

```
[user@cirrus-login0 ~]$ module list
Currently Loaded Modulefiles:
1) git/2.35.1(default)                       6) gcc/8.2.0(default)
2) singularity/3.7.2(default)                7) intel-cc-18/18.0.5.274
3) epcc/utils                                8) intel-fc-18/18.0.5.274
4) /scratch/sw/modulefiles/epcc/setup-env    9) intel-compilers-18/18.05.274
5) intel-license                            10) mpt/2.25
```

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-18 into the development environment:

```
module load intel-compilers-18
```

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

```
module load intel-compilers-18/18.0.5.274
```

will load version 18.0.2.274 for you, regardless of the default.

If a module loading file cannot be accessed within 10 seconds, a warning message will appear: `Warning: Module system not loaded`.

If you want to clean up, `module remove` will remove a loaded module:

```
module remove intel-compilers-18
```

(or `module rm intel-compilers-18` or `module unload intel-compilers-18`) will unload what ever version of intel-compilers-18 (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 18 of the Intel compilers; the following command will change to version 19:

```
module swap intel-compilers-18 intel-compilers-19
```

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-*` and `intel-*/compilers` modules, where * references the version. For example, to load the 2019 release, you would run:

```
module load intel-compilers-19
```

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

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

See the extended section below for further details of available Intel compiler versions and tools.

5.2.2 GCC Compiler Suite

The GCC compiler suite is accessed by loading the `gcc/*` modules, where * again is the version. For example, to load version 8.2.0 you would run:

```
module load gcc/8.2.0
```

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

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

5.3 Compiling MPI codes

MPI on Cirrus is currently provided by the HPE MPT library.

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

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 Intel compilers module you intend to use (e.g. `intel-compilers-19`):

```
module load intel-compilers-19
```

The compiler wrappers are then available as

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

Note:

The MPT compiler wrappers use GCC by default rather than the Intel compilers:

When compiling C applications you must also specify that `mpicc` should use the `icc` compiler with, for example, `mpicc -cc=icc`. Similarly, 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 `mpicc -cc=icc -v` or `mpicxx -cxx=icpc -v` to see which compiler is actually being called.

Alternatively, you can set the environment variables `MPICC_CC=icc` and/or `MPICXX=icpc` to ensure the correct base compiler is used:

```
export MPICC_CC=icc
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 compilers with MPT.

5.3.2 Using Intel MPI

Although HPE MPT remains the default MPI library and we recommend that first attempts at building code follow that route, you may also choose to use Intel MPI if you wish. To use these, load the appropriate `intel-mpi` module, for example `intel-mpi-19`:

```
module load intel-mpi-19
```

Please note that the name of the wrappers to use when compiling with Intel MPI depends on whether you are using the Intel compilers or GCC. You should make sure that you or any tools use the correct ones when building software.

Note: Although Intel MPI is available on Cirrus, HPE MPT remains the recommended and default MPI library to use when building applications.

Note: Using Intel MPI 18 can cause warnings in your output similar to `no hfi units are available` or `The /dev/hfi1_0 device failed to appear`. These warnings can be safely ignored, or, if you would prefer to prevent them, you may add the line

```
export I_MPI_FABRICS=shm:ofa
```

to your job scripts after loading the Intel MPI 18 module.

Note: When using Intel MPI 18, you should always launch MPI tasks with `srun`, the supported method on Cirrus. Launches with `mpirun` or `mpiexec` will likely fail.

Using Intel Compilers and Intel MPI

After first loading Intel MPI, you should next load the appropriate `intel-compilers` module (e.g. `intel-compilers-19`):

```
module load intel-compilers-19
```

You may then use the following MPI compiler wrappers:

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

Using GCC Compilers and Intel MPI

After loading Intel MPI, you should next load the `gcc` module you wish to use:

```
module load gcc
```

You may then use these MPI compiler wrappers:

- `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++ `man 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.

5.6 Intel modules and tools

There are a number of different Intel compiler versions available, and there is also a slight difference in the way different versions appear.

A full list is available via `module avail intel`.

The different available compiler versions are:

- `intel-*/18.0.5.274` Intel 2018 Update 4
- `intel-*/19.0.0.117` Intel 2019 Initial release
- `intel-19.5/*` Intel 2019 Update 5
- `intel-20.4/*` Intel 2020 Update 4

We recommend the most up-to-date version in the first instance, unless you have particular reasons for preferring an older version.

For a note on Intel compiler version numbers, see this Intel page

The different module names (or parts thereof) indicate:

- `cc` C/C++ compilers only

- `cmkl` MKL libraries (see Software Libraries section)
- `compilers` Both C/C++ and Fortran compilers
- `fc` Fortran compiler only
- `itac` Intel Trace Analyze and Collector
- `mpi` Intel MPI
- `pxse` Intel Parallel Studio (all Intel modules)
- `tbb` Thread Building Blocks
- `vtune` VTune profiler - note that in older versions (`intel-*/18.0.5.274`, `intel-*/19.0.0.117` VTune is launched as `amplxe-gui` for GUI or `amplxe-cl` for CLI use)

CHAPTER

SIX

RUNNING JOBS ON CIRRUS

As with most HPC services, Cirrus uses a scheduler to manage access to resources and ensure that the thousands of different users of system are able to share the system and all get access to the resources they require. Cirrus uses the Slurm software to schedule jobs.

Writing a submission script is typically the most convenient way to submit your job to the scheduler. 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.

Hint: If you have any questions on how to run jobs on Cirrus do not hesitate to contact the Cirrus Service Desk.

You typically interact with Slurm by issuing Slurm commands from the login nodes (to submit, check and cancel jobs), and by specifying Slurm directives that describe the resources required for your jobs in job submission scripts.

6.1 Basic Slurm commands

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

- `sinfo` - Get information on the partitions and resources available
- `sbatch jobscript.slurm` - Submit a job submission script (in this case called: `jobscript.slurm`) to the scheduler
- `squeue` - Get the current status of jobs submitted to the scheduler
- `scancel 12345` - Cancel a job (in this case with the job ID `12345`)

We cover each of these commands in more detail below.

6.1.1 `sinfo`: information on resources

`sinfo` is used to query information about available resources and partitions. Without any options, `sinfo` lists the status of all resources and partitions, e.g.

```
 [auser@cirrus-login3 ~]$ sinfo

PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard       up   infinite    280   idle r1i0n[0-35],r1i1n[0-35],r1i2n[0-35],r1i3n[0-
↪35],r1i4n[0-35],r1i5n[0-35],r1i6n[0-35],r1i7n[0-6,9-15,18-24,27-33]
```

(continues on next page)

(continued from previous page)

```
gpu-skylake    up   infinite      2   idle r2i3n[0-1]
gpu-cascade    up   infinite     36   idle r2i4n[0-8],r2i5n[0-8],r2i6n[0-8],r2i7n[0-8]
```

6.1.2 `sbatch`: submitting jobs

`sbatch` is used to submit a job script to the job submission system. The script will typically contain one or more `srun` commands to launch parallel tasks.

When you submit the job, the scheduler provides the job ID, which is used to identify this job in other Slurm commands and when looking at resource usage in SAFE.

```
[auser@cirrus-login3 ~]$ sbatch test-job.slurm
Submitted batch job 12345
```

6.1.3 `squeue`: monitoring jobs

`squeue` without any options or arguments shows the current status of all jobs known to the scheduler. For example:

```
[auser@cirrus-login3 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             1554  comp-cse CASTEP_a  auser  R       0:03      2 r2i0n[18-19]
```

will list all jobs on Cirrus.

The output of this is often overwhelmingly large. You can restrict the output to just your jobs by adding the `-u $USER` option:

```
[auser@cirrus-login3 ~]$ squeue -u $USER
```

6.1.4 `scancel`: deleting jobs

`scancel` is used to delete a jobs from the scheduler. If the job is waiting to run it is simply cancelled, if it is a running job then it is stopped immediately. You need to provide the job ID of the job you wish to cancel/stop. For example:

```
[auser@cirrus-login3 ~]$ scancel 12345
```

will cancel (if waiting) or stop (if running) the job with ID `12345`.

6.2 Resource Limits

Note: If you have requirements which do not fit within the current QoS, please contact the Service Desk and we can discuss how to accommodate your requirements.

There are different resource limits on Cirrus for different purposes. There are three different things you need to specify for each job:

- The amount of *primary resource* you require (more information on this below)

- The *partition* that you want to use - this specifies the nodes that are eligible to run your job
- The *Quality of Service (QoS)* that you want to use - this specifies the job limits that apply

Each of these aspects are described in more detail below.

The *primary resources* you request are *compute* resources: either CPU cores on the standard compute nodes or GPU cards on the GPU compute nodes. Other node resources: memory on the standard compute nodes; memory and CPU cores on the GPU nodes are assigned pro rata based on the primary resource that you request.

Warning: On Cirrus, you cannot specify the memory for a job using the `--mem` options to Slurm (e.g. `--mem`, `--mem-per-cpu`, `--mem-per-gpu`). The amount of memory you are assigned is calculated from the amount of primary resource you request.

6.2.1 Primary resources on standard (CPU) compute nodes

The *primary resource* you request on standard compute nodes are CPU cores. The maximum amount of memory you are allocated is computed as the number of CPU cores you requested multiplied by 1/36th of the total memory available (as there are 36 CPU cores per node). So, if you request the full node (36 cores), then you will be allocated a maximum of all of the memory (256 GB) available on the node; however, if you request 1 core, then you will be assigned a maximum of 256/36 = 7.1 GB of the memory available on the node.

Note: Using the `--exclusive` option in jobs will give you access to the full node memory even if you do not explicitly request all of the CPU cores on the node.

Note: You will not generally have access to the full amount of memory resource on the the node as some is retained for running the operating system and other system processes.

6.2.2 Primary resources on GPU nodes

The *primary resource* you request on standard compute nodes are GPU cards. The maximum amount of memory and CPU cores you are allocated is computed as the number of GPU cards you requested multiplied by 1/4 of the total available (as there are 4 GPU cards per node). So, if you request the full node (4 GPU cards), then you will be allocated a maximum of all of the memory (384 GB) available on the node; however, if you request 1 GPU card, then you will be assigned a maximum of 384/4 = 96 GB of the memory available on the node.

Note: Using the `--exclusive` option in jobs will give you access to all of the CPU cores and the full node memory even if you do not explicitly request all of the GPU cards on the node.

Warning: In order to run jobs on the GPU nodes your budget must have positive GPU hours *and* core hours associated with it. However, only your GPU hours will be consumed when running these jobs.

6.2.3 Partitions

On Cirrus, compute nodes are grouped into partitions. You will have to specify a partition using the `--partition` option in your submission script. The following table has a list of active partitions on Cirrus.

Table 1: Cirrus Partitions

Partition	Description	Total nodes available	Notes
standard	CPU nodes with 2x 18-core Intel Broadwell processors	280	
gpu-cascade	GPU nodes with 4x Nvidia V100 GPU and 2x 20-core Intel Cascade Lake processors	36	
gpu-skylake	GPU nodes with 4x Nvidia V100 GPU and 2x 20-core Intel Skylake processors	2	Only available for short test/development jobs

You can list the active partitions using

```
sinfo
```

Note: you may not have access to all the available partitions.

6.2.4 Quality of Service (QoS)

On Cirrus Quality of Service (QoS) is used alongside partitions to set resource limits. The following table has a list of active QoS on Cirrus.

Table 2: Cirrus QoS

QoS Name	Jobs Running Per User	Jobs Queued Per User	Max Wall-time	Max Size	Applies to Partitions	Notes
standard	No limit	500 jobs	4 days	2520 cores (70 nodes/25%)	standard	
capability	1 job	4 jobs	24 hours	228 nodes (8192+ cores/81%) or 144 GPUs	standard, gpu-cascade	
long	5 jobs	20 jobs	14 days	16 nodes or 8 GPUs	standard, gpu-cascade	
high-priority	10 jobs	20 jobs	4 days	140 nodes	standard	
gpu	No limit	128 jobs	4 days	64 GPUs (16 nodes/40%)	gpu-skylake, gpu-cascade	
short	1 job	2 jobs	20 minutes	2 nodes or 4 GPUs	standard, gpu-skylake, gpu-cascade	

You can find out the QoS that you can use by running the following command:

```
sacctmgr show assoc user=$USER cluster=cirrus format=cluster,account,user,qos%50
```

6.3 Troubleshooting

6.3.1 Slurm error handling

MPI jobs

Users of MPI codes may wish to ensure termination of all tasks on the failure of one individual task by specifying the `--kill-on-bad-exit` argument to `srun`. E.g.,

```
srun -n 36 --kill-on-bad-exit ./my-mpi-program
```

This can prevent effective "hanging" of the job until the wall time limit is reached.

Automatic resubmission

Jobs that fail are not automatically resubmitted by Slurm on Cirrus. Automatic resubmission can be enabled for a job by specifying the `--requeue` option to `sbatch`.

6.3.2 Slurm error messages

An incorrect submission will cause Slurm to return an error. Some common problems are listed below, with a suggestion about the likely cause:

- `sbatch: unrecognized option <text>`
  - One of your options is invalid or has a typo. `man sbatch` to help.
- `error: Batch job submission failed: No partition specified or system default partition`

  A `--partition=` option is missing. You must specify the partition (see the list above). This is most often `--partition=standard`.
- `error: invalid partition specified: <partition>`

  `error: Batch job submission failed: Invalid partition name specified`

  Check the partition exists and check the spelling is correct.
- `error: Batch job submission failed: Invalid account or account/partition combination specified`

  This probably means an invalid account has been given. Check the `--account=` options against valid accounts in SAFE.
- `error: Batch job submission failed: Invalid qos specification`

  A QoS option is either missing or invalid. Check the script has a `--qos=` option and that the option is a valid one from the table above. (Check the spelling of the QoS is correct.)
- `error: Your job has no time specification (--time=)...`

  Add an option of the form `--time=hours:minutes:seconds` to the submission script. E.g., `--time=01:30:00` gives a time limit of 90 minutes.
- **`error: QOSMaxWallDurationPerJobLimit`** `error: Batch job submission failed: Job violates accounting/QOS policy (job submit limit, user's size and/or time limits)`

The script has probably specified a time limit which is too long for the corresponding QoS. E.g., the time limit for the short QoS is 20 minutes.

6.3.3 Slurm queued reasons

The `squeue` command allows users to view information for jobs managed by Slurm. Jobs typically go through the following states: PENDING, RUNNING, COMPLETING, and COMPLETED. The first table provides a description of some job state codes. The second table provides a description of the reasons that cause a job to be in a state.

Table 3: Slurm Job State codes

Status	Code	Description
PENDING	PD	Job is awaiting resource allocation.
RUNNING	R	Job currently has an allocation.
SUSPENDED	S	Job currently has an allocation.
COMPLETING	CG	Job is in the process of completing. Some processes on some nodes may still be active.
COMPLETED	CD	Job has terminated all processes on all nodes with an exit code of zero.
TIMEOUT	TO	Job terminated upon reaching its time limit.
STOPPED	ST	Job has an allocation, but execution has been stopped with SIGSTOP signal. CPUS have been retained by this job.
OUT_OF_MEMORY	OOM	Job experienced out of memory error.
FAILED	F	Job terminated with non-zero exit code or other failure condition.
NODE_FAIL	NF	Job terminated due to failure of one or more allocated nodes.
CANCELLED	CA	Job was explicitly cancelled by the user or system administrator. The job may or may not have been initiated.

For a full list of see Job State Codes

Table 4: Slurm Job Reasons

Reason	Description
Priority	One or more higher priority jobs exist for this partition or advanced reservation.
Resources	The job is waiting for resources to become available.
BadConstraints	The job's constraints can not be satisfied.
BeginTime	The job's earliest start time has not yet been reached.
Dependency	This job is waiting for a dependent job to complete.
Licenses	The job is waiting for a license.
WaitingForScheduling	No reason has been set for this job yet. Waiting for the scheduler to determine the appropriate reason.
Prolog	Its PrologSlurmctld program is still running.
JobHeldAdmin	The job is held by a system administrator.
JobHeldUser	The job is held by the user.
JobLaunchFailure	The job could not be launched. This may be due to a file system problem, invalid program name, etc.
NonZeroExitCode	The job terminated with a non-zero exit code.
InvalidAccount	The job's account is invalid.
InvalidQOS	The job's QOS is invalid.
QOSUsageThreshold	Required QOS threshold has been breached.
QOSJobLimit	The job's QOS has reached its maximum job count.
QOSResourceLimit	The job's QOS has reached some resource limit.
QOSTimeLimit	The job's QOS has reached its time limit.
NodeDown	A node required by the job is down.
TimeLimit	The job exhausted its time limit.
ReqNodeNotAvail	Some node specifically required by the job is not currently available. The node may currently be in use, reserved for another job, in an advanced reservation, DOWN, DRAINED, or not responding. Nodes which are DOWN, DRAINED, or not responding will be identified as part of the job's "reason" field as "UnavailableNodes". Such nodes will typically require the intervention of a system administrator to make available.

For a full list of see Job Reasons

6.4 Output from Slurm jobs

Slurm places standard output (STDOUT) and standard error (STDERR) for each job in the file `slurm_<JobID>.out`. This file appears in the job's working directory once your job starts running.

Note: This file is plain text and can contain useful information to help debugging if a job is not working as expected. The Cirrus Service Desk team will often ask you to provide the contents of this file if you contact them for help with issues.

6.5 Specifying resources in job scripts

You specify the resources you require for your job using directives at the top of your job submission script using lines that start with the directive `#SBATCH`.

Note: Options provided using `#SBATCH` directives can also be specified as command line options to `srun`.

If you do not specify any options, then the default for each option will be applied. As a minimum, all job submissions must specify the budget that they wish to charge the job too, the partition they wish to use and the QoS they want to use with the options:

- `--account=<budgetID>` your budget ID is usually something like `t01` or `t01-test`. You can see which budget codes you can charge to in SAFE.
- `--partition=<partition>` The partition specifies the set of nodes you want to run on. More information on available partitions is given above.
- `--qos="QoS"` The QoS specifies the limits to apply to your job. More information on available QoS are given above.

Other common options that are used are:

- `--time=<hh:mm:ss>` the maximum walltime for your job. *e.g.* For a 6.5 hour walltime, you would use `--time=6:30:0`.
- `--job-name=<jobname>` set a name for the job to help identify it in Slurm command output.

Other not so common options that are used are:

- `--switches=max-switches{@max-time-to-wait}` optimum switches and max time to wait for them. The scheduler will wait indefinitely when attempting to place these jobs. Users can override this indefinite wait. The scheduler will deliberately place work to clear space for these jobs, so we don't foresee the indefinite wait nature to be an issue.

In addition, parallel jobs will also need to specify how many nodes, parallel processes and threads they require.

- `--exclusive` to ensure that you have exclusive access to a compute node
- `--nodes=<nodes>` the number of nodes to use for the job.
- `--tasks-per-node=<processes per node>` the number of parallel processes (e.g. MPI ranks) per node.
- `--cpus-per-task=<threads per task>` the number of threads per parallel process (e.g. number of OpenMP threads per MPI task for hybrid MPI/OpenMP jobs). **Note:** you must also set the `OMP_NUM_THREADS` environment variable if using OpenMP in your job and usually add the `--cpu-bind=cores` option to `srun`

Note: For parallel jobs, you should request exclusive node access with the `--exclusive` option to ensure you get the expected resources and performance.

6.6 srun: Launching parallel jobs

If you are running parallel jobs, your job submission script should contain one or more `srun` commands to launch the parallel executable across the compute nodes. As well as launching the executable, `srun` also allows you to specify the distribution and placement (or *pinning*) of the parallel processes and threads.

If you are running MPI jobs that do not also use OpenMP threading, then you should use `srun` with no additional options. `srun` will use the specification of nodes and tasks from your job script, `sbatch` or `salloc` command to launch the correct number of parallel tasks.

If you are using OpenMP threads then you will generally add the `--cpu-bind=cores` option to `srun` to bind threads to cores to obtain the best performance.

Note: See the example job submission scripts below for examples of using `srun` for pure MPI jobs and for jobs that use OpenMP threading.

6.7 Example parallel job submission scripts

A subset of example job submission scripts are included in full below.

Hint: Do not replace `srun` with `mpirun` in the following examples. Although this might work under special circumstances, it is not guaranteed and therefore not supported.

6.7.1 Example: job submission script for MPI parallel job

A simple MPI job submission script to submit a job using 4 compute nodes and 36 MPI ranks per node for 20 minutes would look like:

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=Example_MPI_Job
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# We use the "standard" partition as we are running on CPU nodes
#SBATCH --partition=standard
# We use the "standard" QoS as our runtime is less than 4 days
#SBATCH --qos=standard

# Load the default HPE MPI environment
module load mpt
```

(continues on next page)

(continued from previous page)

```
# Change to the submission directory
cd $SLURM_SUBMIT_DIR

# 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
#   srun picks up the distribution from the sbatch options
srun ./my_mpi_executable.x
```

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

By default, srun will launch an MPI job that uses all of the cores you have requested via the "nodes" and "tasks-per-node" options. If you want to run fewer MPI processes than cores you will need to change the script.

For example, to run this program on 128 MPI processes you have two options:

- set `--tasks-per-node=32` for an even distribution across nodes (this may not always be possible depending on the exact combination of nodes requested and MPI tasks required)
- set the number of MPI tasks explicitly using `#SBATCH --ntasks=128`

Note: If you specify `--ntasks` explicitly and it is not compatible with the value of `tasks-per-node` then you will get a warning message from srun such as `srun: Warning: can't honor --ntasks-per-node set to 36`.

In this case, srun does the sensible thing and allocates MPI processes as evenly as it can across nodes. For example, the second option above would result in 32 MPI processes on each of the 4 nodes.

See above for a more detailed discussion of the different `sbatch` options.

Note on MPT task placement

By default, `mpt` will distribute processss to physical cores (cores 0-17 on socket 0, and cores 18-35 on socket 1) in a cyclic fashion. That is, rank 0 would be placed on core 0, task 1 on core 18, rank 2 on core 1, and so on (in a single-node job). This may be undesirable. Block, rather than cyclic, distribution can be obtained with

```
#SBATCH --distribution=block:block
```

The `block:block` here refers to the distribution on both nodes and sockets. This will distribute rank 0 for core 0, rank 1 to core 1, rank 2 to core 2, and so on.

6.7.2 Example: 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 36.

In the example below, we are using 4 nodes for 6 hours. There are 8 MPI processes in total (2 MPI processes per node) and 18 OpenMP threads per MPI process. This results in all 36 physical cores per node being used.

Note: the use of the `--cpu-bind=cores` option to generate the correct affinity settings.

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=Example_MPI_Job
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --ntasks=8
#SBATCH --tasks-per-node=2
#SBATCH --cpus-per-task=18

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# We use the "standard" partition as we are running on CPU nodes
#SBATCH --partition=standard
# We use the "standard" QoS as our runtime is less than 4 days
#SBATCH --qos=standard

# Load the default HPE MPI environment
module load mpt

# Change to the submission directory
cd $SLURM_SUBMIT_DIR

# 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

srun --cpu-bind=cores ./my_mixed_executable.x arg1 arg2
```

6.7.3 Example: job submission script for OpenMP parallel job

A simple OpenMP job submission script to submit a job using 1 compute nodes and 36 threads for 20 minutes would look like:

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=Example_OpenMP_Job
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=1
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=36

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# We use the "standard" partition as we are running on CPU nodes
#SBATCH --partition=standard
# We use the "standard" QoS as our runtime is less than 4 days
#SBATCH --qos=standard

# Load any required modules
module load mpt

# Change to the submission directory
cd $SLURM_SUBMIT_DIR

# Set the number of threads to the CPUs per task
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Launch the parallel job
#   Using 36 threads per node
#   srun picks up the distribution from the sbatch options
srun --cpu-bind=cores ./my_openmp_executable.x
```

This will run your executable "my_openmp_executable.x" in parallel on 36 threads. Slurm will allocate 1 node to your job and srun will place 36 threads (one per physical core).

See above for a more detailed discussion of the different `sbatch` options

6.8 Job arrays

The Slurm 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 Slurm 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, the following script runs 56 subjobs, with the subjob index as the only argument to the executable. Each subjob requests a single node and uses all 36 cores on the node by placing 1 MPI process per core and specifies 4 hours maximum runtime per subjob:

```
#!/bin/bash
# Slurm job options (name, compute nodes, job time)

#SBATCH --name=Example_Array_Job
#SBATCH --time=04:00:00
#SBATCH --exclusive
#SBATCH --nodes=1
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1
#SBATCH --array=0-55

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# We use the "standard" partition as we are running on CPU nodes
#SBATCH --partition=standard
# We use the "standard" QoS as our runtime is less than 4 days
#SBATCH --qos=standard

# Load the default HPE MPI environment
module load mpt

# Change to the submission directory
cd $SLURM_SUBMIT_DIR

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

srun /path/to/exe $SLURM_ARRAY_TASK_ID
```

6.8.2 Submitting a job array

Job arrays are submitted using `sbatch` in the same way as for standard jobs:

```
sbatch job_script.pbs
```

6.9 Job chaining

Job dependencies can be used to construct complex pipelines or chain together long simulations requiring multiple steps.

Note: The `--parsable` option to `sbatch` can simplify working with job dependencies. It returns the job ID in a format that can be used as the input to other commands.

For example:

```
jobid=$(sbatch --parsable first_job.sh)
sbatch --dependency=afterok:$jobid second_job.sh
```

or for a longer chain:

```
jobid1=$(sbatch --parsable first_job.sh)
jobid2=$(sbatch --parsable --dependency=afterok:$jobid1 second_job.sh)
jobid3=$(sbatch --parsable --dependency=afterok:$jobid1 third_job.sh)
sbatch --dependency=afterok:$jobid2,afterok:$jobid3 last_job.sh
```

6.10 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 small/short MPI jobs. However, 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 way to achieve this on Cirrus is to use an interactive jobs.

Interactive jobs via SLURM take two slightly different forms. The first uses `srun` directly to allocate resource to be used interactively; the second uses both `salloc` and `srun`.

6.10.1 Using srun

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

A convenient way to do this is as follows.

```
[user@cirrus-login1]$ srun --exclusive --nodes=1 --time=00:20:00 --partition=standard --
↪qos=standard --account=z04 --pty /usr/bin/bash --login
[user@r1i0n14]$
```

This requests the exclusive use of one node for the given time (here, 20 minutes). The `--pty /usr/bin/bash --login` requests an interactive login shell be started. (Note the prompt has changed.) Interactive commands can then be used as normal and will execute on the compute node. When no longer required, you can type `exit` or CTRL-D to release the resources and return control to the front end shell.

```
[user@r1i0n14]$ exit
logout
[user@cirrus-login1]$
```

Note that the new interactive shell will reflect the environment of the original login shell. If you do not wish this, add the `--export=none` argument to `srun` to provide a clean login environment.

Within an interactive job, one can use `srun` to launch parallel jobs in the normal way, e.g.,

```
[user@r1i0n14]$ srun -n 2 ./a.out
```

In this context, one could also use `mpirun` directly. Note we are limited to the 36 cores of our original `--nodes=1` `srun` request.

6.10.2 Using `salloc` with `srun`

This approach uses the``salloc`` command to reserve compute nodes and then `srun` to launch relevant work.

To submit a request for a job reserving 2 nodes (72 physical cores) for 1 hour you would issue the command:

```
[user@cirrus-login1]$ salloc --exclusive --nodes=2 --tasks-per-node=36 --cpus-per-task=1␣
↪--time=01:00:00  --partition=standard --qos=standard --account=t01
salloc: Granted job allocation 8699
salloc: Waiting for resource configuration
salloc: Nodes r1i7n[13-14] are ready for job
[user@cirrus-login1]$
```

Note that this starts a new shell on the login node associated with the allocation (the prompt has not changed). The allocation may be released by exiting this new shell.

```
[user@cirrus-login1]$ exit
salloc: Relinquishing job allocation 8699
[user@cirrus-login1]$
```

While the allocation lasts you will be able to run parallel jobs on the compute nodes by issuing the `srun` command in the normal way. The resources available are those specified in the original `salloc` command. For example, with the above allocation,

```
$ srun ./mpi-code.out
```

will run 36 MPI tasks per node on two nodes.

If your allocation reaches its time limit, it will automatically be termintated and the associated shell will exit. To check that the allocation is still running, use `squeue`:

```
[user@cirrus-login1]$ squeue -u user
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              8718  standard     bash    user   R       0:07      2 r1i7n[18-19]
```

Choose a time limit long enough to allow the relevant work to be completed.

The `salloc` method may be useful if one wishes to associate operations on the login node (e.g., via a GUI) with work in the allocation itself.

6.11 Reservations

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.

Reservations require justification. They will only be approved if the request could not be fulfilled with the standard queues. For example, you require a job/jobs to run at a particular time e.g. for a demonstration or course.

Note: Reservation requests must be submitted at least 120 hours in advance of the reservation start time. We cannot guarantee to meet all reservation requests due to potential conflicts with other demands on the service but will do our best to meet all requests.

Reservations will be charged at 1.5 times the usual rate and our policy is that they will be charged the full rate for the entire reservation at the time of booking, whether or not you use the nodes for the full time. In addition, you will not be refunded the compute time if you fail to use them due to a job crash unless this crash is due to a system failure.

To request a reservation please contact the Cirrus Service Desk. You need to provide the following:

- The start time and date of the reservation.
- The end time and date of the reservation.
- The project code for the reservation.
- The number of nodes/cores/GPU required.
- Your justification for the reservation – this must be provided or the request will be rejected.

Your request will be checked by the Cirrus User Administration team and, if approved, you will be provided a reservation ID which can be used on the system. To submit jobs to a reservation, you need to add `--reservation=<reservation ID>` and `--qos=reservation` options to your job submission script or Slurm job submission command.

Note: You must have at least 1 CPUh - and 1 GPUh for reservations on GPU nodes - to be able to submit jobs to reservations.

Tip: You can submit jobs to a reservation as soon as the reservation has been set up; jobs will remain queued until the reservation starts.

6.12 Serial jobs

Unlike parallel jobs, serial jobs will generally not need to specify the number of nodes and exclusive access (unless they want access to all of the memory on a node. You usually only need the `--ntasks=1` specifier. For example, a serial job submission script could look like:

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=Example_Serial_Job
#SBATCH --time=0:20:0
#SBATCH --ntasks=1
```

(continues on next page)

(continued from previous page)

```
# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# We use the "standard" partition as we are running on CPU nodes
#SBATCH --partition=standard
# We use the "standard" QoS as our runtime is less than 4 days
#SBATCH --qos=standard

# Change to the submission directory
cd $SLURM_SUBMIT_DIR

# Enforce threading to 1 in case underlying libraries are threaded
export OMP_NUM_THREADS=1

# Launch the serial job
#   Using 1 thread
srun --cpu-bind=cores ./my_serial_executable.x
```

Note: Remember that you will be allocated memory based on the number of tasks (i.e. CPU cores) that you request. You will get ~7.1 GB per task/core. If you need more than this for your serial job then you should ask for the number of tasks you need for the required memory (or use the `--exclusive` option to get access to all the memory on a node) and launch specifying a single task using `srun --ntasks=1 --cpu-bind=cores`.

6.13 Temporary files and `/tmp` in batch jobs

Applications which normally read and write temporary files from `/tmp` may require some care in batch jobs on Cirrus. As the size of `/tmp` on backend nodes is relatively small (< 150 MB), applications should use a different location to prevent possible failures. This is relevant for both CPU and GPU nodes.

Note also that the default value of the variable `TMPDIR` in batch jobs is a memory-resident file system location specific to the current job (typically in the `/dev/shm` directory). Files here reduce the available capacity of main memory on the node.

It is recommended that applications with significant temporary file space requirement should use the *Solid state storage*. E.g., a submission script might contain:

```
export TMPDIR="/scratch/space1/x01/auser/$SLURM_JOBID.tmp"
mkdir -p $TMPDIR
```

to set the standard temporary directory to a unique location in the solid state storage. You will also probably want to add a line to clean up the temporary directory at the end of your job script, e.g.

```
rm -r $TMPDIR
```

Tip: Applications should not hard-code specific locations such as `/tmp`. Parallel applications should further ensure that there are no collisions in temporary file names on separate processes/nodes.

CHAPTER

SEVEN

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 (e.g. Infiniband), file systems (e.g. Lustre) and potentially other resources (such as 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 of Singularity).

7.1 Useful Links

- Singularity website
- Singularity documentation archive

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 principle, 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.

1. Interactively on the login nodes
2. Interactively on compute nodes
3. As serial processes within a non-interactive batch script
4. As parallel processes within a non-interactive batch script

We provide information on each of these scenarios. First, we describe briefly how to get existing images onto Cirrus so that 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.

For example, to retrieve an image from SingularityHub on Cirrus we can simply issue a Singularity command to pull the image.

```
[user@cirrus-login1 ~]$ module load singularity
[user@cirrus-login1 ~]$ singularity pull hello-world.sif shub://vsoch/hello-world
```

The image located at the `shub` URI is written to a Singularity Image File (SIF) called `hello-world.sif`.

7.3.2 Interactive use on the login nodes

The container represented by the image file can be run on the login node like so.

```
[user@cirrus-login1 ~]$ singularity run hello-world.sif
RaawwWWWWWRRRR!! Avocado!
[user@cirrus-login1 ~]$
```

We can also `shell` into the container.

```
[user@cirrus-login1 ~]$ singularity shell hello-world.sif
Singularity> ls /
bin  boot  dev  environment  etc  home  lib  lib64  lustre  media  mnt  opt  proc  rawr.
↪sh  root  run  sbin  singularity  srv  sys  tmp  usr  var
Singularity> exit
exit
[user@cirrus-login1 ~]$
```

For more information see the Singularity documentation.

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 first have to submit an interactive serial job to get interactive access to the compute node.

First though, move to a suitable location on `/work` and re-pull the `hello-world` image. This step is necessary as the compute nodes do not have access to the `/home` file system.

```
[user@cirrus-login1 ~]$ cd ${HOME/home/work}
[user@cirrus-login1 ~]$ singularity pull hello-world.sif shub://vsoch/hello-world
```

Now reserve a full node to work on interactively by issuing an `salloc` command, see below.

```
[user@cirrus-login1 ~]$ salloc --exclusive --nodes=1 \
  --tasks-per-node=36 --cpus-per-task=1 --time=00:20:00 \
  --partition=standard --qos=standard --account=[budget code]
salloc: Pending job allocation 14507
salloc: job 14507 queued and waiting for resources
salloc: job 14507 has been allocated resources
salloc: Granted job allocation 14507
salloc: Waiting for resource configuration
salloc: Nodes r1i0n8 are ready for job
[user@cirrus-login1 ~]$ ssh r1i0n8
```

Note the prompt has changed to show you are on a compute node. Once you are logged in to the compute node (you may need to submit your account password), move to a suitable location on `/work` as before. You can now use the `hello-world` image in the same way you did on the login node.

```
[user@r1i0n8 ~]$ cd ${HOME/home/work}
[user@r1i0n8 ~]$ singularity shell hello-world.sif
Singularity> exit
exit
[user@r1i0n8 ~]$ exit
logout
Connection to r1i0n8 closed.
[user@cirrus-login1 ~]$ exit
exit
salloc: Relinquishing job allocation 14507
salloc: Job allocation 14507 has been revoked.
[user@cirrus-login1 ~]$
```

Note we used `exit` to leave the interactive container shell and then called `exit` twice more to close 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 `hello-world.sif` we built above on Cirrus would be as follows.

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

# job options (name, compute nodes, job time)
#SBATCH --job-name=hello-world
#SBATCH --ntasks=1
#SBATCH --exclusive
#SBATCH --time=0:20:0
#SBATCH --partition=standard
#SBATCH --qos=standard

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]

# Load any required modules
module load singularity

# Run the serial executable
srun --cpu-bind=cores singularity run ${HOME/home/work}/hello-world.sif
```

Submit this script using the `sbatch` command and once the job has finished, you should see `RaawwWWWWWRRRR!! Avocado!` in the Slurm output file.

7.3.5 Parallel processes within a non-interactive batch script

Running a Singularity container on the compute nodes isn't too different from launching a normal parallel application. The submission script below shows that the `srun` command now contains an additional `singularity` clause.

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

# job options (name, compute nodes, job time)
#SBATCH --job-name=[name of application]
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1
#SBATCH --exclusive
#SBATCH --time=0:20:0
#SBATCH --partition=standard
#SBATCH --qos=standard

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]

# Load any required modules
module load mpt
```

(continues on next page)

(continued from previous page)

```
module load singularity

# The host bind paths for the Singularity container.
BIND_ARGS=/scratch/sw,/opt/hpe,/etc/libibverbs.d,/path/to/input/files

# The file containing environment variable settings that will allow
# the container to find libraries on the host, e.g., LD_LIBRARY_PATH .
ENV_PATH=/path/to/container/environment/file

CONTAINER_PATH=/path/to/singularity/image/file

APP_PATH=/path/to/containerized/application/executable
APP_PARAMS=[application parameters]

srun --distribution=block:block --hint=nomultithread \
    singularity exec --bind ${BIND_ARGS} --env-file ${ENV_PATH} ${IMAGE_PATH}
        ${APP_PATH} ${APP_PARAMS}
```

The script above runs a containerized application such that each of the four nodes requested is fully populated. In general, the containerized application's input and output will be read from and written to a location on the host; hence, it is necessary to pass a suitable bind path to singularity (`/path/to/input/files`).

Note: The paths in the submission script that begin `/path/to` should be provided by the user. All but one of these paths are host specific. The exception being `APP_PATH`, which should be given a path relative to the container file system.

If the Singularity image file was built according to the Bind model, you will need to specify certain paths (`--bind`) and environment variables (`--env-file`) that allow the containerized application to find the required MPI libraries.

Otherwise, if the image follows the Hybrid model and so contains its own MPI implementation, you instead need to be sure that the containerized MPI is compatible with the host MPI, the one loaded in the submission script. In the example above, the host MPI is HPE MPT 2.25, but you could also use OpenMPI (with `mpirun`), either by loading a suitable `openmpi` module or by referencing the paths to an OpenMPI installation that was built locally (i.e., within your Cirrus work folder).

7.4 Creating Your Own Singularity Images

You can create Singularity images by importing from DockerHub or Singularity Hub directly to Cirrus. 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 definition file in order 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 and HPE MPI libraries.

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.

- Installing Singularity on Linux

7.4.2 Accessing Cirrus Modules from Inside a Container

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.

Below, is an example Singularity definition file that builds 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
    echo 'module() { eval `/usr/bin/modulecmd bash $*`; }' >> /etc/bashrc
    yum install wget -y
    yum install which -y
    yum install squashfs-tools -y
```

If we save this definition to a file called `centos7.def`, we can use the following `build` command to build the image (remember this command must be run on a system where you have root access, not on Cirrus).

```
me@my-system:~> sudo singularity build centos7.sif centos7.def
```

The resulting image file (`centos7.sif`) can then be copied to Cirrus using scp; such an image already exists on Cirrus and can be found in the `/scratch/sw/singularity/images` folder.

When you use that image interactively on Cirrus you must start with a login shell and also bind `/scratch/sw` so that the container can see all the module files, see below.

```
[user@cirrus-login1 ~]$ module load singularity
[user@cirrus-login1 ~]$ singularity exec -B /scratch/sw \
  /scratch/sw/singularity/images/centos7.sif \
    /bin/bash --login
```

(continues on next page)

(continued from previous page)

```
Singularity> module avail intel-compilers

------------------------- /scratch/sw/modulefiles ----------------------
intel-compilers-18/18.05.274  intel-compilers-19/19.0.0.117
Singularity> exit
logout
[user@cirrus-login1 ~]$
```

7.4.3 Altering a Container on Cirrus

A container image file is immutable but it is possible to alter the image if you convert the file to a sandbox. The sandbox is essentially a directory on the host system that contains the full container file hierarchy.

You first run the `singularity build` command to perform the conversion followed by a `shell` command with the `--writable` option. You are now free to change the files inside the container sandbox.

```
user@cirrus-login1 ~]$ singularity build --sandbox image.sif.sandbox image.sif
user@cirrus-login1 ~]$ singularity shell -B /scratch/sw --writable image.sif.sandbox
Singularity>
```

In the example above, the `/scratch/sw` bind path is specified, allowing you to build code that links to the Cirrus module libraries.

Finally, once you are finished with the sandbox you can exit and convert back to the original image file.

```
Singularity> exit
exit
user@cirrus-login1 ~]$ singularity build --force image.sif image.sif.sandbox
```

Note: Altering a container in this way will cause the associated definition file to be out of step with the current image. Care should be taken to keep a record of the commands that were run within the sandbox so that the image can be reproduced.

CHAPTER

EIGHT

USING PYTHON

Python on Cirrus is provided by the Anaconda distribution. The Python 3 version of the distribution is available.

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 the Cirrus Anaconda Modules

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

```
module load anaconda/python3
```

for Python 3 (v3.9.7).

You can verify the current version of Python with:

```
[user@cirrus-login1 ~]$ module load anaconda/python3
[user@cirrus-login1 ~]$ python3 --version
Python 3.9.7 :: Anaconda, Inc.
```

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

- 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-login1 ~]$ module load anaconda
[user@cirrus-login1 ~]$ conda list
# packages in environment at /scratch/sw/anaconda/anaconda3-2021.11:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0           py39h06a4308_0
```

(continues on next page)

(continued from previous page)

```
_libgcc_mutex             0.1                        main
_openmp_mutex             4.5                      1_gnu
alabaster                 0.7.12               pyhd3eb1b0_0
anaconda                  2021.11                    py39_0
anaconda-client           1.9.0               py39h06a4308_0
anaconda-navigator        2.1.1                      py39_0
anaconda-project          0.10.1               pyhd3eb1b0_0
anyio                     2.2.0               py39h06a4308_1
appdirs                   1.4.4                pyhd3eb1b0_0
argh                      0.26.2              py39h06a4308_0
argon2-cffi               20.1.0              py39h27cfd23_1
...
```

8.1.2 Adding packages to the Anaconda distribution

Packages cannot be added to the central Anaconda distribution 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 the next section.

8.2 Accessing the Cirrus Miniconda3 Modules

There are a number of Miniconda3 modules available on Cirrus that support Python-based parallel codes. In fact, each module provides a suite of packages pertinent to parallel processing and numerical analysis, e.g., dask, ipyparallel, jupyter, matplotlib, numpy, pandas and scipy.

8.2.1 mpi4py for CPU

For example, the `python/3.9.13` module provides mpi4py 3.1.3 linked with OpenMPI 4.1.4.

The scripts below demonstrate how to run a simple MPI Broadcast example (`numpy-broadcast.py`) across two compute nodes.

```
#!/usr/bin/env python

"""
Parallel Numpy Array Broadcast
"""

import mpi4py.rc
mpi4py.rc.initialize = False

from mpi4py import MPI
import numpy as np
import sys

MPI.Init()

comm = MPI.COMM_WORLD
```

(continues on next page)

(continued from previous page)

```
size = comm.Get_size()
rank = comm.Get_rank()
name = MPI.Get_processor_name()

arraySize = 100
if rank == 0:
    data = np.arange(arraySize, dtype='i')
else:
    data = np.empty(arraySize, dtype='i')

comm.Bcast(data, root=0)

if rank == 0:
    sys.stdout.write(
        "Rank %d of %d (%s) has broadcast %d integers.\n"
        % (rank, size, name, arraySize))
else:
    sys.stdout.write(
        "Rank %d of %d (%s) has received %d integers.\n"
        % (rank, size, name, arraySize))

    arrayBad = False
    for i in range(100):
        if data[i] != i:
            arrayBad = True
            break

    if arrayBad:
        sys.stdout.write(
            "Error, rank %d array is not as expected.\n"
            % (rank))

MPI.Finalize()
```

The purpose of the `mpi4py.rc.initialize = False` line above is to turn off the automatic MPI initialization that would otherwise happen as a result of `from mpi4py import MPI` - the MPI initialization is invoked explicitly by calling `MPI.Init()`.

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=broadcast
#SBATCH --time=00:20:00
#SBATCH --exclusive
#SBATCH --partition=standard
#SBATCH --qos=standard
#SBATCH --account=[budget code]
#SBATCH --nodes=2
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

module load python/3.9.13
```

(continues on next page)

(continued from previous page)

```
export OMPI_MCA_mca_base_component_show_load_errors=0

srun numpy-broadcast.py
```

The Slurm submission script contains an OpenMPI MCA setting that prevents false errors from being recorded in the output file.

Please see the mpi4py online docs for more coding examples.

8.2.2 mpi4py for GPU

There's also an mpi4py module (again using OpenMPI 4.1.4) that is tailored for CUDA 11.6 on the Cirrus GPU nodes, `python/3.9.13-gpu`. We show below an example that features an MPI reduction performed on a CuPy array (`cupy-allreduce.py`).

```
#!/usr/bin/env python

"""
Reduce-to-all CuPy Arrays
"""

import mpi4py.rc
mpi4py.rc.initialize = False

from mpi4py import MPI
import cupy as cp
import sys

MPI.Init()

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()
name = MPI.Get_processor_name()

sendbuf = cp.arange(10, dtype='i')
recvbuf = cp.empty_like(sendbuf)
assert hasattr(sendbuf, '__cuda_array_interface__')
assert hasattr(recvbuf, '__cuda_array_interface__')
cp.cuda.get_current_stream().synchronize()
comm.Allreduce(sendbuf, recvbuf)

assert cp.allclose(recvbuf, sendbuf*size)

sys.stdout.write(
    "%d (%s): recvbuf = %s\n"
    % (rank, name, str(recvbuf)))

MPI.Finalize()
```

```
#!/bin/bash

#SBATCH --job-name=allreduce
#SBATCH --time=00:20:00
#SBATCH --exclusive
#SBATCH --partition=gpu-cascade
#SBATCH --qos=gpu
#SBATCH --account=[budget code]
#SBATCH --nodes=2
#SBATCH --gres=gpu:4

module load python/3.9.13-gpu

export CUPY_CACHE_DIR=${HOME/home/work}/.cupy/kernel_cache

export OMPI_MCA_mpi_warn_on_fork=0
export OMPI_MCA_mca_base_component_show_load_errors=0

srun --ntasks=8 --tasks-per-node=4 --cpus-per-task=1 cupy-allreduce.py
```

By default, the CuPy cache will be located within the user's home directory. And so, as `/home` is not accessible from the GPU nodes, it is necessary to set `CUPY_CACHE_DIR` such that the cache is on the `/work` file system instead.

8.2.3 Machine Learning frameworks

There are several more Python-based modules that also target the Cirrus GPU nodes. These include two machine learning frameworks, `pytorch/1.12.1-gpu` and `tensorflow/2.9.1-gpu`. Both modules are Python virtual environments that extend `python/3.9.13-gpu`. The MPI comms is handled by the Horovod 0.25.0 package along with the NVIDIA Collective Communications Library v2.11.4.

A full package list for these environments can be obtained by loading the module of interest and then running `pip list`.

Note: The Cirrus compute nodes cannot access the `/home` file system, which means you may need to run `export XDG_CACHE_HOME=${HOME/home/work}` if you're working from within an interactive session as that export command will ensure the pip cache is located off `/work`.

Please click on the link indicated to see examples of how to use the PyTorch and TensorFlow modules .

More detail on the Cirrus GPU nodes can be found at https://cirrus.readthedocs.io/en/main/user-guide/gpu.html .

8.3 Custom Miniconda3 Environments

To setup a custom Python environment including packages that are not in the central installation, the simplest method is to install Miniconda locally within your project area on Cirrus.

8.3.1 Installing Miniconda3

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

```
[user@cirrus-login1 ~]$ 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 installer.

Once you have downloaded the installer, you can run it via `bash`.

```
[user@cirrus-login1 ~]$ bash Miniconda3-latest-Linux-x86_64.sh
```

Note that the installer will prompt you for a number of choices which include the license agreement to which you must answer *yes*.

```
Do you accept the license terms? [yes|no]
[no] >>> yes
```

The installer will prompt for the install location, the default being your home directory, to install in `/scratch` or `/work` please change the install location here. Remember, the Cirrus compute nodes do not have access to `/home`.

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

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

[/home/t01/t01/user/miniconda3] >>> /work/t01/t01/user/miniconda3
```

The final question will be about initialization. If you wish to use only Miniconda and no other python environments (such as the central Anaconda modules), you may want to answer *yes* at this point, otherwise we would suggest answering *no*.

```
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>> no

You have chosen to not have conda modify your shell scripts at all.
To activate conda's base environment in your current shell session:

eval "$(/work/t01/t01/user/miniconda3/bin/conda shell.YOUR_SHELL_NAME hook)"

To install conda's shell functions for easier access, first activate, then:

conda init

If you'd prefer that conda's base environment not be activated on startup,
   set the auto_activate_base parameter to false:

conda config --set auto_activate_base false
```

(continues on next page)

(continued from previous page)

```
Thank you for installing Miniconda3!
```

If you have answered *no*, the instructions above should be followed to activate the base conda environment. This can be done in a number of ways.

- Perform the shell `eval` command manually as required.

```
$ eval "$(/work/t01/t01/user/miniconda3/bin/conda shell.bash hook)"
```

- Add the shell `eval` command to a script, which can then be invoked when required, e.g., `source ~/miniconda-init.sh`.

Answering *yes* to the initialization question will mean that the shell command is effectively injected into your `.bashrc` file, and will be executed whenever you login to your Cirrus account. In this case, you may at a later date wish to issue a command that prevents the conda base environment from being activated at login.

```
$ conda config --set auto_activate_base false
```

If not activated automatically at login, the conda base environment can instead be activated in the usual way.

```
[user@cirrus-login1 ~]$ conda activate
(base) [user@cirrus-login1 ~]$ conda list
# packages in environment at /work/t01/t01/user/miniconda3:
#
# Name                    Version                   Build  Channel
_libgcc_mutex             0.1                        main
_openmp_mutex             4.5                       1_gnu
brotlipy                  0.7.0         py39h27cfd23_1003
ca-certificates           2021.7.5             h06a4308_1
certifi                   2021.5.30        py39h06a4308_0
cffi                      1.14.6           py39h400218f_0
chardet                   4.0.0         py39h06a4308_1003
conda                     4.10.3           py39h06a4308_0
...
(base) [user@cirrus-login1 ~]$  conda deactivate
[user@cirrus-login1 ~]$
```

8.3.2 Installing packages into Miniconda3

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, see below for two examples.

```
[user@cirrus-login1 ~]$ conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done

... package details omitted ...

Proceed ([y]/n)? y

...
```

(continues on next page)

(continued from previous page)

```
[user@cirrus-login0 ~]$ conda list
# packages in environment at /work/t01/t01/user/miniconda3:
#
# Name                    Version                   Build  Channel
_libgcc_mutex             0.1                 conda_forge    conda-forge
_openmp_mutex             4.5                       1_llvm    conda-forge
blas                      1.0                         mkl
brotlipy                  0.7.0         py39h27cfd23_1003
ca-certificates           2021.10.8            ha878542_0    conda-forge
cairo                     1.16.0            ha00ac49_1009    conda-forge
certifi                   2021.10.8        py39hf3d152e_1    conda-forge
cffi                      1.15.0           py39h4bc2ebd_0    conda-forge
chardet                   4.0.0         py39h06a4308_1003
conda                     4.10.3           py39hf3d152e_3    conda-forge
conda-package-handling    1.7.3            py39h27cfd23_1
cryptography              3.4.7            py39hd23ed53_0
font-ttf-dejavu-sans-mono 2.37                hab24e00_0    conda-forge
font-ttf-inconsolata      3.000               h77eed37_0    conda-forge
font-ttf-source-code-pro  2.038               h77eed37_0    conda-forge
font-ttf-ubuntu           0.83                hab24e00_0    conda-forge
fontconfig                2.13.1            hba837de_1005    conda-forge
fonts-conda-ecosystem     1                            0    conda-forge
fonts-conda-forge         1                            0    conda-forge
freetype                  2.10.4              h0708190_1    conda-forge
gettext                   0.19.8.1          h73d1719_1008    conda-forge
icu                       69.1                h9c3ff4c_0    conda-forge
idna                      2.10               pyhd3eb1b0_0
intel-openmp              2021.4.0          h06a4308_3561
ld_impl_linux-64          2.36.1              hea4e1c9_2    conda-forge
libffi                    3.4.2               h9c3ff4c_4    conda-forge
libgcc-ng                 11.2.0             h1d223b6_11    conda-forge
libgirepository           1.70.0              hb520f89_1    conda-forge
libglib                   2.70.0              h174f98d_1    conda-forge
libiconv                  1.16                h516909a_0    conda-forge
libpng                    1.6.37              h21135ba_2    conda-forge
libstdcxx-ng              11.2.0             he4da1e4_11    conda-forge
libuuid                   2.32.1            h7f98852_1000    conda-forge
libxcb                    1.13              h7f98852_1003    conda-forge
libxml2                   2.9.12              h885dcf4_1    conda-forge
libzlib                   1.2.11            h36c2ea0_1013    conda-forge
llvm-openmp               12.0.1              h4bd325d_1    conda-forge
mkl                       2021.4.0           h06a4308_640
mkl-service               2.4.0            py39h7f8727e_0
mkl_fft                   1.3.1            py39hd3c417c_0
mkl_random                1.2.2            py39h51133e4_0
ncurses                   6.2                 he6710b0_1
numpy                     1.21.2           py39h20f2e39_0
numpy-base                1.21.2           py39h79a1101_0
...
```

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-login1 ~]$ conda install -c conda-forge pygobject
```

8.4 Note on Default Python

System versions of python occur in the default PATH if no action has been taken.

```
[user@cirrus-login1]$ which python2

[user@cirrus-login1]$ which python3
/usr/bin/python3
```

These should not be used. Use either an Anaconda or a Miniconda version.

8.5 Using JupyterLab on Cirrus

It is possible to view and run JupyterLab on both the login and compute nodes of Cirrus. Please note, you can test notebooks on the login nodes, but please don't attempt to run any computationally intensive work (such jobs will be killed should they reach the login node CPU limit).

If you want to run your JupyterLab on a compute node, you will need to enter an interactive session; otherwise you can start from a login node prompt.

1. As described above, load the Anaconda module on Cirrus using `module load anaconda/python3`.
2. Run `export JUPYTER_RUNTIME_DIR=$(pwd)`.
3. Start the JupyterLab server by running `jupyter lab --ip=0.0.0.0 --no-browser` - once it's started, you will see some lines resembling the following output.

   ```
   Or copy and paste one of these URLs:
       ...
    or http://127.0.0.1:8888/lab?token=<string>
   ```

   You will need the URL shown above for step 6.

4. Please skip this step if you are connecting from Windows. If you are connecting from Linux or macOS, open a new terminal window, and run the following command.

   ```
   ssh <username>@login.cirrus.ac.uk -L<port_number>:<node_id>:<port_number>
   ```

   where <username> is your username, <port_number> is as shown in the URL from the Jupyter output and <node_id> is the name of the node we're currently on. On a login node, this will be `cirrus-login1`, or similar; on a compute node, it will be a mix of numbers and letters such as `r2i5n5`.

   Note: If, when you connect in the new terminal, you see a message of the form *channel_setup_fwd_listener_tcpip: cannot listen to port: 8888*, it means port 8888 is already in use. You need to go back to step 3 (kill the existing jupyter lab) and retry with a new explicit port number by adding the `--port=N` option. The port number `N` can be in the range 5000-65535. You should then use the same port number in place of 8888.

5. Please skip this step if you are connecting from Linux or macOS. If you are connecting from Windows, you should use MobaXterm to configure an SSH tunnel as follows.

   5.1. Click on the `Tunnelling` button above the MobaXterm terminal. Create a new tunnel by clicking on `New SSH tunnel` in the window that opens.

   5.2. In the new window that opens, make sure the `Local port forwarding` radio button is selected.

   5.3. In the `forwarded port` text box on the left under `My computer with MobaXterm`, enter the port number indicated in the Jupyter server output.

   5.4. In the three text boxes on the bottom right under `SSH server` enter `login.cirrus.ac.uk`, your Cirrus username, and then `22`.

   5.5. At the top right, under `Remote server`, enter the name of the Cirrus login or compute node that you noted earlier followed by the port number (e.g. *8888*).

   5.6. Click on the `Save` button.

   5.7. In the tunnelling window, you will now see a new row for the settings you just entered. If you like, you can give a name to the tunnel in the leftmost column to identify it. Click on the small key icon close to the right for the new connection to tell MobaXterm which SSH private key to use when connecting to Cirrus. You should tell it to use the same `.ppk` private key that you normally use.

   5.8. The tunnel should now be configured. Click on the small start button (like a play > icon) for the new tunnel to open it. You'll be asked to enter your Cirrus password – please do so.

6. Now, if you open a browser window on your local machine, you should be able to navigate to the URL from step 3, and this should display the JupyterLab server.

   - Please note, you will get a connection error if you haven't used the correct node name in step 4 or 5.

If you are on a compute node, the JupyterLab server will be available for the length of the interactive session you have requested.

You can also run Jupyter sessions using the centrally-installed Miniconda3 modules available on Cirrus. For example, the following link provides instructions for how to setup a Jupyter server on a GPU node.

https://github.com/hpc-uk/build-instructions/tree/main/pyenvs/ipyparallel

CHAPTER

NINE

USING THE CIRRUS GPU NODES

Cirrus has 38 GPU compute nodes each equipped with 4 NVIDIA V100 (Volta) GPU cards. This section of the user guide gives some details of the hardware; it also covers how to compile and run standard GPU applications.

The GPU cards on Cirrus do not support graphics rendering tasks; they are set to *compute cluster* mode and so only support computational tasks.

9.1 Hardware details

All of the Cirrus GPU nodes contain four Tesla V100-SXM2-16GB (Volta) cards. Each card has 16GB of high-bandwidth memory, `HBM`, often referred to as device memory. Maximum device memory bandwidth is in the region of 900 GB per second. Each card has 5,120 CUDA cores and 640 Tensor cores.

There are two GPU Slurm partitions installed on Cirrus. The first called `gpu-skylake` features two GPU nodes that each have Intel Skylake processors. These nodes are only available for short testing/development jobs via the `short` QoS. The remaining 36 nodes form the `gpu-cascade` partition and have the slightly more recent Intel Cascade Lake architecture. Users concerned with host performance should add the specific compilation options appropriate for the processor.

In both cases, the host node has two 20-core sockets (2.5 GHz) and a total of 384 GB host memory (192 GB per socket). Each core supports two threads in hardware.

For further details of the V100 architecture see, https://www.nvidia.com/en-gb/data-center/tesla-v100/ .

9.2 Compiling software for the GPU nodes

9.2.1 NVIDIA HPC SDK

NVIDIA now make regular releases of a unified HPC SDK which provides the relevant compilers and libraries needed to build and run GPU programs. Versions of the SDK are available via the module system.

```
$ module avail nvidia/nvhpc
```

NVIDIA encourage the use of the latest available version, unless there are particular reasons to use earlier versions. The default version is therefore the latest module version present on the system.

Each release of the NVIDIA HPC SDK may include several different versions of the CUDA toolchain. For example, the `nvidia/nvhpc/21.2` module comes with CUDA 10.2, 11.0 and 11.2. Only one of these CUDA toolchains can be active at any one time and for `nvhpc/21.2` this is CUDA 11.2.

Here is a list of available HPC SDK versions, and the corresponding version of CUDA:

Module	Supported CUDA Version
nvidia/nvhpc/22.2	CUDA 11.6
nvidia/nvhpc/21.9	CUDA 11.4
nvidia/nvhpc/21.2	CUDA 11.2

To load the latest NVIDIA HPC SDK use

```
$ module load nvidia/nvhpc
```

The following sections provide some details of compilation for different programming models.

9.2.2 CUDA

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

Programs, typically written in C or C++, are compiled with `nvcc`. As well as `nvcc`, a host compiler is required. By default, a `gcc` module is added when `nvidia/nvhpc` is loaded.

Compile your source code in the usual way.

```
nvcc -arch=sm_70 -o cuda_test.x cuda_test.cu
```

Note: The `-arch=sm_70` compile option ensures that the binary produced is compatible with the NVIDIA Volta architecture.

Using CUDA with Intel compilers

You can load either the Intel 18 or Intel 19 compilers to use with `nvcc`.

```
module unload gcc
module load intel-compilers-19
```

You can now use `nvcc -ccbin icpc` to compile your source code with the Intel C++ compiler `icpc`.

```
nvcc -arch=sm_70 -ccbin icpc -o cuda_test.x cuda_test.cu
```

9.2.3 Compiling OpenACC code

OpenACC is a directive-based approach to introducing parallelism into either C/C++ or Fortran codes. A code with OpenACC directives may be compiled like so.

```
$ module load nvidia/nvhpc
$ nvc program.c
```

```
$ nvc++ program.cpp
```

Note that `nvc` and `nvc++` are distinct from the NVIDIA CUDA compiler `nvcc`. They provide a way to compile standard C or C++ programs without explicit CUDA content. See `man nvc` or `man nvc++` for further details.

9.2.4 CUDA Fortran

CUDA Fortran provides extensions to standard Fortran which allow GPU functionality. CUDA Fortran files (with file extension `.cuf`) may be compiled with the NVIDIA Fortran compiler.

```
$ module load nvidia/nvhpc
$ nvfortran program.cuf
```

See `man nvfortran` for further details.

9.2.5 OpenMP for GPUs

The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran and can offload computation from the host (i.e. CPU) to one or more target devices (such as the GPUs on Cirrus). OpenMP code can be compiled with the NVIDIA compilers in a similar manner to OpenACC. To enable this functionality, you must add `-mp=gpu` to your compile command.

```
$ module load nvidia/nvhpc
$ nvc++ -mp=gpu program.cpp
```

You can specify exactly which GPU to target with the `-gpu` flag. For example, the Volta cards on Cirrus use the flag `-gpu=cc70`.

During development it can be useful to have the compiler report information about how it is processing OpenMP pragmas. This can be enabled by the use of `-Minfo=mp`, see below.

```
nvc -mp=gpu -Minfo=mp testprogram.c
main:
24, #omp target teams distribute parallel for thread_limit(128)
24, Generating Tesla and Multicore code
Generating "nvkernel_main_F1L88_2" GPU kernel
26, Loop parallelized across teams and threads(128), schedule(static)
```

9.3 Submitting jobs to the GPU nodes

To run a GPU job, a SLURM submission must specify a GPU partition and a quality of service (QoS) as well as the number of GPUs required. You specify the number of GPU cards you want using the `--gres=gpu:N` option, where `N` is typically 1, 2 or 4.

Note: As there are 4 GPUs per node, each GPU is associated with 1/4 of the resources of the node, i.e., 10/40 physical cores and roughly 91/384 GB in host memory.

Allocations of host resources are made pro-rata. For example, if 2 GPUs are requested, `sbatch` will allocate 20 cores and around 190 GB of host memory (in addition to 2 GPUs). Any attempt to use more than the allocated resources will result in an error.

This automatic allocation by SLURM for GPU jobs means that the submission script should not specify options such as `--ntasks` and `--cpus-per-task`. Such a job submission will be rejected. See below for some examples of how to use host resources and how to launch MPI applications.

If you specify the `--exclusive` option, you will automatically be allocated all host cores and all memory from the node irrespective of how many GPUs you request. This may be needed if the application has a large host memory requirement.

If more than one node is required, exclusive mode `--exclusive` and `--gres=gpu:4` options must be included in your submission script. It is, for example, not possible to request 6 GPUs other than via exclusive use of two nodes.

Warning: In order to run jobs on the GPU nodes your budget must have positive GPU hours *and* positive CPU core hours associated with it. However, only your GPU hours will be consumed when running these jobs.

9.3.1 Partitions

Your job script must specify a partition. The following table has a list of relevant GPU partitions on Cirrus.

Table 1: Cirrus Partitions

Partition	Description	Maximum Job Size (Nodes)
gpu-cascade	GPU nodes with Cascade Lake processors	36
gpu-skylake	GPU nodes with Skylake processors	2

9.3.2 Quality of Service (QoS)

Your job script must specify a QoS relevant for the GPU nodes. Available QoS specifications are as follows.

Table 2: GPU QoS

QoS Name	Jobs Running Per User	Jobs Queued Per User	Max Wall-time	Max Size	GPU Partition
gpu	No limit	128 jobs	4 days	64 GPUs	gpu-cascade
long	5 jobs	20 jobs	14 days	8 GPUs	gpu-cascade
short	1 job	2 jobs	20 minutes	4 GPUs or 2 nodes	gpu-skylake

9.4 Examples

9.4.1 Job submission script using one GPU on a single node

A job script that requires 1 GPU accelerator and 10 CPU cores for 20 minutes would look like the following.

```
#!/bin/bash
#
#SBATCH --partition=gpu-cascade
#SBATCH --qos=gpu
#SBATCH --gres=gpu:1
#SBATCH --time=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
```

(continues on next page)

(continued from previous page)

```
# Load the required modules
module load nvidia/nvhpc

srun ./cuda_test.x
```

This will execute one host process with access to one GPU. If we wish to make use of the 10 host cores in this allocation, we could use host threads via OpenMP.

```
export OMP_NUM_THREADS=10
export OMP_PLACES=cores

srun --ntasks=1 --cpus-per-task=10 --hint=nomultithread ./cuda_test.x
```

The launch configuration is specified directly to `srun` because, for the GPU partitions, it is not possible to do this via `sbatch`.

9.4.2 Job submission script using multiple GPUs on a single node

A job script that requires 4 GPU accelerators and 40 CPU cores for 20 minutes would appear as follows.

```
#!/bin/bash
#
#SBATCH --partition=gpu-cascade
#SBATCH --qos=gpu
#SBATCH --gres=gpu:4
#SBATCH --time=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]

# Load the required modules
module load nvidia/nvhpc

srun ./cuda_test.x
```

A typical MPI application might assign one device per MPI process, in which case we would want 4 MPI tasks in this example. This would again be specified directly to `srun`.

```
srun --ntasks=4 ./mpi_cuda_test.x
```

9.4.3 Job submission script using multiple GPUs on multiple nodes

See below for a job script that requires 8 GPU accelerators for 20 minutes.

```
#!/bin/bash
#
#SBATCH --partition=gpu-cascade
#SBATCH --qos=gpu
#SBATCH --gres=gpu:4
```

(continues on next page)

(continued from previous page)

```
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --time=00:20:00

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]

# Load the required modules
module load nvidia/nvhpc

srun ./cuda_test.x
```

An MPI application with four MPI tasks per node would be launched as follows.

```
srun --ntasks=8 --tasks-per-node=4 ./mpi_cuda_test.x
```

Again, these options are specified directly to `srun` rather than being declared as `sbatch` directives.

Attempts to oversubscribe an allocation (10 cores per GPU) will fail, and generate an error message.

```
srun: error: Unable to create step for job 234123: More processors requested
than permitted
```

9.5 Debugging GPU applications

Applications may be debugged using `cuda-gdb`. This is an extension of `gdb` which can be used with CUDA. We assume the reader is familiar with `gdb`.

First, compile the application with the `-g -G` flags in order to generate debugging information for both host and device code. Then, obtain an interactive session like so.

```
$ srun --nodes=1 --partition=gpu-skylake --qos=short --gres=gpu:1 \
       --time=0:20:0 --account=[budget code] --pty /bin/bash
```

Next, load the NVIDIA HPC SDK module and start `cuda-gdb` for your application.

```
$ module load nvidia/nvhpc
$ cuda-gdb ./my-application.x
NVIDIA (R) CUDA Debugger
...
(cuda-gdb)
```

Debugging then proceeds as usual. One can use the help facility within `cuda-gdb` to find details on the various debugging commands. Type `quit` to end your debug session followed by `exit` to close the interactive session.

Note, it may be necessary to set the temporary directory to somewhere in the user space (e.g., `export TMPDIR=$(pwd)/tmp`) to prevent unexpected internal CUDA driver errors.

For further information on CUDA-GDB, see https://docs.nvidia.com/cuda/cuda-gdb/index.html.

9.6 Profiling GPU applications

NVIDIA provide two useful tools for profiling performance of applications: Nsight Systems and Nsight Compute; the former provides an overview of application performance, while the latter provides detailed information specifically on GPU kernels.

9.6.1 Using Nsight Systems

Nsight Systems provides an overview of application performance and should therefore be the starting point for investigation. To run an application, compile as normal (including the `-g` flag) and then submit a batch job.

```
#!/bin/bash

#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --partition=gpu-skylake
#SBATCH --qos=short
#SBATCH --gres=gpu:1

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]

module load nvidia/nvhpc

srun -n 1 nsys profile -o prof1 ./my_application.x
```

The run should then produce an additional output file called, in this case, `prof1.qdrep`. The recommended way to view the contents of this file is to download the NVIDIA Nsight package to your own machine (you do not need the entire HPC SDK). Then copy the `.qdrep` file produced on Cirrus so that if can be viewed locally.

Note, a profiling run should probably be of a short duration so that the profile information (contained in the `.qdrep` file) does not become prohibitively large.

Details of the download of Nsight Systems and a user guide can be found via the links below.

https://developer.nvidia.com/nsight-systems

https://docs.nvidia.com/nsight-systems/UserGuide/index.html

If your code was compiled with the tools provided by `nvidia/nvhpc/21.2` you should download and install Nsight Systems v2020.5.1.85.

9.6.2 Using Nsight Compute

Nsight Compute may be used in a similar way as Nsight Systems. A job may be submitted like so.

```
#!/bin/bash

#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --partition=gpu-skylake
#SBATCH --qos=short
```

(continues on next page)

(continued from previous page)

```
#SBATCH --gres=gpu:1

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]

module load nvidia/nvhpc

srun -n 1 nv-nsight-cu-cli --section SpeedOfLight_RooflineChart \
                           -o prof2 -f ./my_application.x
```

In this case, a file called `prof2.ncu-rep` should be produced. Again, the recommended way to view this file is to download the Nsight Compute package to your own machine, along with the `.ncu-rep` file from Cirrus. The `--section` option determines which statistics are recorded (typically not all hardware counters can be accessed at the same time). A common starting point is `--section MemoryWorkloadAnalysis`.

Consult the NVIDIA documentation for further details.

https://developer.nvidia.com/nsight-compute

https://docs.nvidia.com/nsight-compute/2021.2/index.html

Nsight Compute v2021.3.1.0 has been found to work for codes compiled using `nvhpc` versions 21.2 and 21.9.

9.7 Compiling and using GPU-aware MPI

For applications using message passing via MPI, considerable improvements in performance may be available by allowing device memory references in MPI calls. This allows replacement of relevant host device transfers by direct communication within a node via NVLink. Between nodes, MPI communication will remain limited by network latency and bandwidth.

A version of OpenMPI with both CUDA-aware MPI support and SLURM support is available via

```
$ module load openmpi/4.1.4-cuda-11.6
$ module load nvidia/nvhpc-nompi/22.2
```

The location of the MPI include files and libraries must then be specified explicitly, e.g.,

```
$ nvcc -I${MPI_HOME}/include my_program.cu -L${MPI_HOME}/lib -lmpi
```

This will produce an executable in the usual way.

9.7.1 Run time

A batch script to use such an executable might be:

```
#!/bin/bash

#SBATCH --time=00:20:00

#SBATCH --nodes=1
#SBATCH --partition=gpu-cascade
#SBATCH --qos=gpu
```

(continues on next page)

(continued from previous page)

```
#SBATCH --gres=gpu:4

module load openmpi/4.1.2-cuda-11.6

export OMP_NUM_THREADS=1

# Note the addition
export OMPI_MCA_pml=ob1

srun --ntasks=4 --cpus-per-task=10 --hint=nomultithread ./my_program
```

Note the addition of the environment variable `OMPI_MCA_pml=ob1` is required for correct operation. As before, MPI and placement options should be directly specified to `srun` and not via `SBATCH` directives.

CHAPTER

TEN

SOLID STATE STORAGE

In addition to the Lustre file system, the Cirrus login and compute nodes have access to a shared, high-performance, solid state storage system (also known as RPOOL). This storage system is network mounted and shared across the login nodes and GPU compute nodes in a similar way to the normal, spinning-disk Lustre file system but has different performanc characteristics.

The solid state storage has a maximum usable capacity of 256 TB which is shared between all users.

10.1 Backups, quotas and data longevity

There are no backups of any data on the solid state storage so you should ensure that you have copies of critical data elsewhere.

In addition, the solid state storage does not currently have any quotas (user or group) enabled so all users are potentially able to access the full 256 TB capacity of the storage system. We ask all users to be considerate in their use of this shared storage system and to delete any data on the solid state storage as soon as it no longer needs to be there.

We monitor the usage of the storage system by users and groups and will potentially remove data that is stopping other users getting fair access to the storage and data that has not been actively used for long periods of time.

10.2 Accessing the solid-state storage

You access the solid-state storage at `/scratch/space1` on both the login nodes and on the compute nodes.

Everybody has access to be able to create directories and add data so we suggest that you create a directory for your project and/or user to avoid clashes with files and data added by other users. For example, if my project is `t01` and my username is `auser` then I could create a directory with

```
mkdir -p /scratch/space1/t01/auser
```

When these directories are initially created they will be *world-readable*. If you do not want users from other projects to be able to see your data, you should change the permissions on your new directory. For example, to restrict the directory so that only other users in your project can read the data you would use:

```
chmod -R o-rwx /scratch/space1/t01
```

10.3 Copying data to/from solid-state storage

You can move data to/from the solid-state storage in a number of different ways:

- By copying to/from another Cirrus file system - either interactively on login nodes or as part of a job submission script
- By transferring directly to/from an external host via the login nodes

10.3.1 Local data transfer

The most efficient tool for copying to/from the Cirrus file systems (*/home*, */work*) to the solid state storage is generally the `cp` command, e.g.

```
cp -r /path/to/data-dir /scratch/space1/t01/auser/
```

where `/path/to/data-dir` should be replaced with the path to the data directory you are wanting to copy and assuming, of course, that you have setup the `t01/auser` subdirectories as described above).

Note: If you are transferring data from your `/work` directory, these commands can also be added to job submission scripts running on the compute nodes to move data as part of the job. If you do this, remember to include the data transfer time in the overall walltime for the job.

Data from your `/home` directory is not available from the compute nodes and must therefore be transferred from a login node.

10.3.2 Remote data transfer

You can transfer data directly to the solid state storage from external locations using `scp` or `rsync` in exactly the same way as you would usually do to transfer data to Cirrus. Simply substitute the path to the location on the solid state storage for that you would normally use for Cirrus. For example, if you are on the external location (e.g. your laptop), you could use something like:

```
scp -r data_dir user@login.cirrus.ac.uk:/scratch/space1/t01/auser/
```

You can also use commands such as `wget` and `curl` to pull data from external locations directly to the solid state storage.

Note: You cannot transfer data from external locations in job scripts as the Cirrus compute nodes do not have external network access.

CHAPTER

ELEVEN

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/
- An MPI tutorial: http://www-unix.mcs.anl.gov/mpi/tutorial/gropp/talk.html
- HPC tutorials by NCSA http://www.citutor.org/login.html

11.2 MPI programming

- MPI: The Complete Reference. Snir, Otto, Huss-Lederman, Walker and Dongarra. MIT Press. ISBN 0 262 69184 1
- MPI: The Complete Reference, volume 2. Gropp et al. MIT Press. ISBN 0262571234
- 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

- Practical Parallel Programming. Gregory V. Wilson. MIT Press. ISBN 0 262 23186 7
- 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

- Fortran90/95 Explained. Metcalf and Reid. Oxford Science Publications. ISBN 0 19 851888 9
- Fortran 90 Programming. Ellis, Philips, Lahey. Addison-Wesley. ISBN 0-201-54446-6
- Programmers Guide to Fortran90. Brainerd, Goldberg, Adams. Unicomp. ISBN 0-07-000248-7
- The High Performance Fortran Handbook. Koelbel, Loveman, Schreiber, Steele, Zosel. ISBN 0-262-11185-3 / 0-262-61094-9
- Parallel Programming using C++. G.V.Wilson and P Lu. MIT Press. ISBN 0 262 73118 5

11.6 Programming skills

- Debugging and Performance Tuning for Parallel Computing Systems, Simmons et al.
- Foundations of Parallel Programming, A Machine-independent Approach, Lewis.

CHAPTER

TWELVE

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 transsfer 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:

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=HW_RADIOSS_test
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=1
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=1
```

(continues on next page)

(continued from previous page)

```
# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Set the number of threads to the CPUs per task
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Load Hyperworks module
module load altair-hwsolvers/14.0.210

# Launch the parallel job
#   Using 36 threads per node
#   srun picks up the distribution from the sbatch options
srun --cpu-bind=cores 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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=HW_OptiStruct_SMP
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=1
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=36

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
```

(continues on next page)

(continued from previous page)

```
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load Hyperworks module
module load altair-hwsolvers/14.0.210

# Launch the parallel job
#   Using 36 threads per node
#   srun picks up the distribution from the sbatch options
srun --cpu-bind=cores --ntasks=18 optistruct box.fem -nt 18
```

12.4.2 OptiStruct SPMD (MPI)

- OptiStruct SPMD documentation

There are four different parallelisation schemes for SPMD OptStruct 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:

```
 #!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=HW_OptiStruct_SPMD
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=2
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

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

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

(continues on next page)

(continued from previous page)

```
export OMP_NUM_THREADS=1

# Run the OptStruct SPMD Solver (domain decomposition mode)
#   Use 72 cores, 36 on each node (i.e. all physical cores)
#   srun picks up the distribution from the sbatch options
srun --ntasks=72 $ALTAIR_HOME/hwsolvers/optistruct/bin/linux64/optistruct_14.0.211_
↪linux64_impi box.fem -ddmmode
```

CHAPTER

THIRTEEN

ANSYS FLUENT

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=ANSYS_test
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
```

(continues on next page)

(continued from previous page)

```
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load Ansys
module purge
module load ansys/19.0

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

scontrol show hostnames $SLURM_NODELIST > ~/fluent.launcher.host.txt

# Launch the parallel job
./fluent 3ddp -g -i inputfile.fl \
  -pinfiniband -alnamd64 -t504 -pib    \
  -cnf=~/fluent.launcher.host.txt      \
  -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
```

(continues on next page)

(continued from previous page)

```
/parallel/latency
/parallel/timer/usage
/parallel/timer/reset
; 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
```

CHAPTER

FOURTEEN

CASTEP

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 proprieties 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

 # Slurm job options (name, compute nodes, job time)
 #SBATCH --job-name=CASTEP_Example
 #SBATCH --time=1:0:0
 #SBATCH --exclusive
 #SBATCH --nodes=4
 #SBATCH --tasks-per-node=36
```

(continues on next page)

(continued from previous page)

```
 #SBATCH --cpus-per-task=1

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load CASTEP version 18 module
module load castep/18

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

# Run using input in test_calc.in
srun castep.mpi test_calc
```

CHAPTER

FIFTEEN

CP2K

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` module. MPI only `cp2k.popt` and MPI/OpenMP Hybrid `cp2k.psmp` binaries are available.

15.3 Running parallel CP2K jobs - MPI Only

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

For example, the following script will run a CP2K job using 4 nodes (144 cores):

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=CP2K_test
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1
```

(continues on next page)

(continued from previous page)

```
# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-
↪skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load CP2K
module load cp2k

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

# Run using input in test.inp
srun cp2k.popt -i test.inp
```

15.4 Running Parallel CP2K Jobs - MPI/OpenMP Hybrid Mode

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

For example, the following script will run a CP2K job using 8 nodes, with 2 OpenMP threads per MPI process:

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=CP2K_test
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=8
#SBATCH --tasks-per-node=18
#SBATCH --cpus-per-task=2

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-
↪skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load CP2K
module load cp2k

# Set the number of threads to 2
export OMP_NUM_THREADS=2

# Run using input in test.inp
srun cp2k.psmp -i test.inp
```

CHAPTER

SIXTEEN

ELEMENTS

ELEMENTS is a computational fluid dynamics (CFD) software tool based on the HELYX® package developed by ENGYS. The software features an advanced open-source CFD simulation engine and a client-server GUI to provide a flexible and cost-effective HPC solver platform for automotive and motorsports design applications, including a dedicated virtual wind tunnel wizard for external vehicle aerodynamics and other proven methods for UHTM, HVAC, aeroacoustics, etc.

16.1 Useful Links

- Information about ELEMENTS
- Information about ENGYS

16.2 Using ELEMENTS on Cirrus

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

ELEMENTS 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 ELEMENTS GUI, using the dedicated client-server node to connect remotely to the cluster.

A complete user's guide to access ELEMENTS on demand via Cirrus is provided by ENGYS as part of this service.

16.3 Running ELEMENTS Jobs in Parallel

The standard execution of ELEMENTS applications on Cirrus is handled through the command line using a submission script to control Slurm. A basic submission script for running multiple ELEMENTS applications in parallel using the SGI-MPT (Message Passing Toolkit) module is included below. In this example the applications `helyxHexMesh`, `caseSetup` and `helyxAero` are run sequentially using 4 nodes (144 cores).

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=Test
#SBATCH --time=1:00:00
```

(continues on next page)

(continued from previous page)

```
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=36
#SBATCH --cpus-per-task=1
#SBATCH --output=test.out

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=t01

# Replace [partition name] below with your partition name (e.g. standard)
#SBATCH --partition=standard


# Replace [QoS name] below with your QoS name (e.g. commercial)
#SBATCH --qos=commercial

# Load any required modules
module load gcc
module load mpt

# Load the HELYX-Core environment v3.5.0 (select version as needed, e.g. 3.5.0)
source /scratch/sw/elements/v3.5.0/CORE/HELYXcore-3.5.0/platforms/activeBuild.shrc

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

for job in `echo $jobs`
do

  case "$job" in
   *                )   options="$myoptions" ;;
  esac

  srun $job $myoptions 2>&1 | tee log/$job.$SLURM_JOB_ID.out

done
```

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

1. Launch ELEMENTS GUI in your local Windows or Linux machine.
2. Create a client-server connection to Cirrus using the dedicated node provided for this service in the GUI. Enter your Cirrus 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 Slurm scheduling associated with every ELEMENTS job is handled automatically by the client-server.
4. Visualise the remote data from your local machine, perform changes to the model and complete as many flow simulations in Cirrus as required, all interactively from within the GUI.

5. Disconnect the client-server at any point during execution, leave a utility or solver running in the cluster, and resume the connection to Cirrus from another client machine to reload an existing case in the GUI when needed.

CHAPTER

SEVENTEEN

FLACS

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. 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. 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.

17.1 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).

17.2 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.

17.2.1 Log into Cirrus

Log into Cirrus following the instructions at *Connecting to Cirrus*.

17.2.2 Upload your data to Cirrus

Transfer your data to Cirrus by following the instructions at *Data Management and Transfer*.

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.

17.2.3 FLACS license manager

In order to use FLACS a valid license is required. To check the availability of a license, a license manager is used. To be able to connect to the license manager from the batch system, users wishing to use FLACS should add the following file as `~/.hasplm/hasp_104628.ini` (that is, in their home directory)

```
; copy this file (vendor is gexcon) to ~/.hasplm/hasp_104628.ini
aggressive = 0
broadcastsearch = 0
serveraddr = cirrus-services1
disable_IPv6 = 1
```

17.2.4 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
```

The usual way to submit work to the queue system is to write a submission script, which would be located in the working directory. This is a standard bash shell script, a simple example of which is given here:

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

#SBATCH --job-name=test_flacs_1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=02:00:00
#SBATCH --partition=standard
#SBATCH --qos=standard

module load flacs-cfd/21.2

run_runflacs 012345
```

The script has a series of special comments (introduced by *#SBATCH*) which give information to the queue system to allow the system to allocate space for the job and to execute the work. These are discussed in more detail below.

The `flacs` module is loaded to make the application available. Note that you should specify the specific version you require:

```
module load flacs-cfd/21.2
```

(Use `module avail flacs` to see which versions are available.) The appropriate FLACS commands can then be executed in the usual way.

Submit your FLACS jobs using the `sbatch` command, e.g.:

```
$ sbatch --account=i123 script.sh
Submitted batch job 157875
```

The `--account=i123` option is obligatory and states that account `i123` will be used to record the CPU time consumed by the job, and result in billing to the relevant customer. You will need your project account code here to replace `i123`. You can check your account details in SAFE.

The name of the submission script here is `script.sh`. The queue system returns a unique job id (here `157875`) to identify the job. For example, the standard output here will appear in a file named `slurm-157875.out` in the current working directory.

17.2.5 Options for FLACS jobs

The `#SBATCH` lines in the script above set various parameters which control execution of the job. The first is `--job-name` just provides a label which will be associated with the job.

The parameter `--ntasks=1` is the number of tasks or processes involved in the job. For a serial FLACS job you would use `--ntasks=1`. The

The maximum length of time (i.e. wall clock time) you want the job to run is specified with the `--time=hh:mm:ss` option. After this time, your job will be terminated by the job scheduler. The default time limit is 12 hours. It is useful to have an estimate of how long your job will take to be able to specify the correct limit (which can take some experience). Note that shorter jobs can sometimes be scheduled more quickly by the system.

Multithreaded FLACS simulations can be run on Cirrus with the following job submission, schematically:

```
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
...
```

(continues on next page)

(continued from previous page)

```
run_runflacs -dir projects/sim 010101 NumThreads=4
```

When submitting multithreaded FLACS simulations the `--cpus-per-task` option should be used in order for the queue system to allocate the correct resources (here 4 threads running on 4 cores). In addition, one must also specify the number of threads used by the simulation with the `NumThreads=4` option to the run_runflacs.

One can also specify the OpenMP version of FLACS explicitly via, e.g.,

```
export OMP_NUM_THREADS=20

run_runflacs version _omp <run number> NumThreads=20
```

See the FLACS manual for further details.

17.2.6 Monitor your jobs

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

```
squeue -u username
```

17.2.7 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.

Note you must determine ahead of time the number of scenarios involved. This determines the number of array elements, which must be specified at the point of job submission. The number of array elements is specified by `--array` argument to `sbatch`.

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

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

# Recall that the resource specification is per element of the array
# so this would give four instances of one task (with one thread per
# task --cpus-per-task=1).

#SBATCH --array=1-128

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=02:00:00
#SBATCH --account=z04

#SBATCH --partition=standard
#SBATCH --qos=commercial

# Abbreviate some SLURM variables for brevity/readability
```

(continues on next page)

(continued from previous page)

```
TASK_MIN=${SLURM_ARRAY_TASK_MIN}
TASK_MAX=${SLURM_ARRAY_TASK_MAX}
TASK_ID=${SLURM_ARRAY_TASK_ID}
TASK_COUNT=${SLURM_ARRAY_TASK_COUNT}

# Form a list of relevant files, and check the number of array elements
# matches the number of cases with 6-digit identifiers.

CS_FILES=(`ls -1 cs??????.dat3`)

if test "${#CS_FILES[@]}" -ne "${TASK_COUNT}";
then
  printf "Number of files is:       %s\n" "${#CS_FILES[@]}"
  printf "Number of array tasks is: %s\n" "${TASK_COUNT}"
  printf "Do not match!\n"
fi

# All tasks loop through the entire list to find their specific case.

for (( jid = $((${TASK_MIN})); jid <= $((${TASK_MAX})); jid++ ));
do
  if test "${TASK_ID}" -eq "${jid}";
  then
      # File list index with offset zero
      file_id=$((${jid} - ${TASK_MIN}))
      # Form the substring file_id (recall syntax is :offset:length)
      my_file=${CS_FILES[${file_id}]}
      my_file_id=${my_file:2:6}
  fi
done

printf "Task %d has file %s id %s\n" "${TASK_ID}" "${my_file}" "${my_file_id}"

module load flacs-cfd/21.2
`which run_runflacs` ${my_file_id}
```

17.2.8 Transfer data from Cirrus to your local system

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

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:

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

17.2.9 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 `sbatch` 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.

17.3 Getting help

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

CHAPTER

EIGHTEEN

GAUSSIAN

Gaussian is a general-purpose computational chemistry package.

18.1 Useful Links

- Gaussian User Guides

18.2 Using Gaussian on Cirrus

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

If you wish to have access to Gaussian on Cirrus please request access via SAFE

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.

18.3 Scratch Directories

You will typically add lines to your job submission script to create a scratch directory on the solid state storage for temporary Gaussian files. e.g.:

```
export GAUSS_SCRDIR="/scratch/space1/x01/auser/$SLURM_JOBID.tmp"
mkdir -p $GAUSS_SCRDIR
```

You should also add a line at the end of your job script to remove the scratch directory. e.g.:

```
rm -r $GAUSS_SCRDIR
```

18.4 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).

```
#!/bin/bash

# job options (name, compute nodes, job time)
#SBATCH --job-name=G16_test
#SBATCH --ntasks=1
#SBATCH --time=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load Gaussian module
module load gaussian

# Setup the Gaussian environment
source $g16root/g16/bsd/g16.profile

# Location of the scratch directory
export GAUSS_SCRDIR="/scratch/space1/x01/auser/$SLURM_JOBID.tmp"
mkdir -p $GAUSS_SCRDIR

# Run using input in "test0027.com"
g16 test0027

# Remove the temporary scratch directory
rm -r $GAUSS_SCRDIR
```

18.5 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.

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

# job options (name, compute nodes, job time)
#SBATCH --job-name=G16_test
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --time=0:20:0

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

(continues on next page)

(continued from previous page)

```
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load Gaussian module
module load gaussian

# Setup the Gaussian environment
source $g16root/g16/bsd/g16.profile

# Location of the scratch directory
export GAUSS_SCRDIR="/scratch/space1/x01/auser/$SLURM_JOBID.tmp"
mkdir -p $GAUSS_SCRDIR

# Run using input in "test0027.com"
export OMP_NUM_THREADS=4
g16 test0027

# Remove the temporary scratch directory
rm -r $GAUSS_SCRDIR
```

CHAPTER

NINETEEN

GROMACS

GROMACS 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.

19.1 Useful Links

- GROMACS User Guides
- GROMACS Tutorials

19.2 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

19.3 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 2 nodes (72 cores) with pure MPI.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=gmx_test
#SBATCH --nodes=2
#SBATCH --tasks-per-node=36
#SBATCH --time=0:25:0
# Make sure you are not sharing nodes with other users
#SBATCH --exclusive
```

(continues on next page)

(continued from previous page)

```
# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load GROMACS module
module load gromacs

# Run using input in test_calc.tpr
export OMP_NUM_THREADS=1
srun gmx_mpi mdrun -s test_calc.tpr
```

19.4 Running parallel GROMACS jobs: hybrid MPI/OpenMP

The following script will run a GROMACS MD job using 2 nodes (72 cores) with 6 MPI processes per node (12 MPI processes in total) and 6 OpenMP threads per MPI process.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=gmx_test
#SBATCH --nodes=2
#SBATCH --tasks-per-node=6
#SBATCH --cpus-per-task=6
#SBATCH --time=0:25:0
# Make sure you are not sharing nodes with other users
#SBATCH --exclusive

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load GROMACS and MPI modules
module load gromacs

# Run using input in test_calc.tpr
export OMP_NUM_THREADS=6
srun gmx_mpi mdrun -s test_calc.tpr
```

19.5 GROMACS GPU jobs

The following script will run a GROMACS GPU MD job using 1 node (40 cores and 4 GPUs). The job is set up to run on *<MPI task count>* MPI processes, and *<OMP thread count>* OMP threads – you will need to change these variables when running your script.

Note: Unlike the base version of GROMACS, the GPU version comes with only MDRUN installed. For any pre- and post-processing, you will need to use the non-GPU version of GROMACS.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=gmx_test
#SBATCH --nodes=1
#SBATCH --time=0:25:0
#SBATCH --exclusive

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]
#SBATCH --gres=gpu:4

# Load GROMACS and MPI modules
module load gromacs/2022.1-gpu

# Run using input in test_calc.tpr
export OMP_NUM_THREADS=<OMP thread count>
srun --ntasks=<MPI task count> --cpus-per-task=<OMP thread count> \
     gmx_mpi mdrun -ntomp <OMP thread count> -s test_calc.tpr
```

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

CHAPTER

TWENTY

HELYX®

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.

20.1 Useful Links

- Information about HELYX
- Information about ENGYS

20.2 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 HELYX 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 this service.

20.3 Running HELYX Jobs in Parallel

The standard execution of HELYX applications on Cirrus is handled through the command line using a submission script to control Slurm. 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`, `caseSetup` and `helyxSolve` are run sequentially using 4 nodes (144 cores).

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=Test
#SBATCH --time=1:00:00
#SBATCH --exclusive
#SBATCH --nodes=4
```

(continues on next page)

(continued from previous page)

```
#SBATCH --ntasks-per-node=36
#SBATCH --cpus-per-task=1
#SBATCH --output=test.out

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=t01

# Replace [partition name] below with your partition name (e.g. standard)
#SBATCH --partition=standard

# Replace [QoS name] below with your QoS name (e.g. commercial)
#SBATCH --qos=commercial

# Load any required modules
module load gcc
module load mpt

# Load the HELYX-Core environment v3.5.0 (select version as needed, e.g. 3.5.0)
source /scratch/sw/helyx/v3.5.0/CORE/HELYXcore-3.5.0/platforms/activeBuild.shrc

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

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

for job in `echo $jobs`
do

   case "$job" in
    *                )   options="$myoptions" ;;
   esac

   srun $job $myoptions 2>&1 | tee log/$job.$SLURM_JOB_ID.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 your local Windows or Linux machine.
2. Create a client-server connection to Cirrus using the dedicated node provided for this service in the GUI. Enter your Cirrus 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 Slurm scheduling associated with every HELYX job is handled automatically by the client-server.
4. Visualise the remote data from your local machine, perform changes to the model and complete as many flow simulations in Cirrus as required, all interactively from within the GUI.
5. Disconnect the client-server at any point during execution, leave a utility or solver running in the cluster, and

resume the connection to Cirrus from another client machine to reload an existing case in the GUI when needed.

CHAPTER

TWENTYONE

LAMMPS

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.

21.1 Useful Links

- LAMMPS Documentation https://lammps.sandia.gov/doc/Manual.html
- LAMMPS Mailing list details https://lammps.sandia.gov/mail.html

21.2 Using LAMMPS on Cirrus

LAMMPS is freely available to all Cirrus users.

21.3 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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=lammps_Example
#SBATCH --time=00:20:00
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
```

(continues on next page)

(continued from previous page)

```
#SBATCH --qos=[qos name]

# Load LAMMPS module
module load lammps

# Run using input in in.test
 srun lmp_mpi < in.test
```

21.4 Compiling LAMMPS on Cirrus

Compile instructions for LAMMPS on Cirrus can be found on GitHub:

- Cirrus LAMMPS compile instructions

CHAPTER

TWENTYTWO

MATLAB

MATLAB combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly.

22.1 Useful Links

- MATLAB Documentation

22.2 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

```
MATLAB                                  Version 9.5
Simulink                                Version 9.2
5G Toolbox                              Version 1.0
Aerospace Blockset                      Version 4.0
Aerospace Toolbox                       Version 3.0
Antenna Toolbox                         Version 3.2
Audio System Toolbox                    Version 1.5
Automated Driving System Toolbox        Version 1.3
Bioinformatics Toolbox                  Version 4.11
Communications Toolbox                  Version 7.0
Computer Vision System Toolbox          Version 8.2
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
```

(continues on next page)

(continued from previous page)

```
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
Simulink Design Optimization                  Version 3.5
Simulink Report Generator                     Version 5.5
Simulink Requirements                         Version 1.2
```

(continues on next page)

(continued from previous page)

```
Simulink Test                              Version 2.5
Stateflow                                  Version 9.2
Statistics and Machine Learning Toolbox Version 11.4
Symbolic Math Toolbox                      Version 8.2
System Identification Toolbox              Version 9.9
Text Analytics Toolbox                     Version 1.2
Trading Toolbox                            Version 3.5
Vehicle Dynamics Blockset                  Version 1.1
Vehicle Network Toolbox                    Version 4.1
Vision HDL Toolbox                         Version 1.7
WLAN Toolbox                               Version 2.0
Wavelet Toolbox                            Version 5.1
```

22.3 Running MATLAB jobs

On Cirrus, MATLAB is intended to be used on the compute nodes within Slurm 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.

22.4 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 Slurm 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 Slurm 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

```
#SBATCH --job-name=Example_MATLAB_Job
#SBATCH --time=0:20:0
#SBATCH --nodes=1
#SBATCH --tasks-per-node=6
#SBATCH --cpus-per-task=1


# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]
```

(continues on next page)

(continued from previous page)

```
module load matlab

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

```
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

```
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
```

(continues on next page)

(continued from previous page)

```
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 *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.

22.5 MATLAB 2019 versions

There has been a change of configuration options for MATLAB from version R2019 and onwards that means the *-r* flag has been replaced with the *-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 *-r* or *-batch* flag replace this line in your Slurm script, i.e.:

```
matlab_wrapper -nodisplay -nodesktop -batch "main_simulated_data_FINAL_clean("$ind","
↪$gamma","$rw",'"$SLURM_JOB_ID"')
```

with:

```
matlab_wrapper_2019 -nodisplay -nodesktop -batch "main_simulated_data_FINAL_clean("$ind",
↪"$gamma","$rw",'"$SLURM_JOB_ID"')
```

and this should allow scripts to run normally.

22.6 Running parallel MATLAB jobs using MDCS

It is possible to use MATLAB on the login node to set up an MDCS Slurm 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.

This MDCS profile can be used in MATLAB on the login node - the MDCS computations are done in Slurm jobs launched using the profile.

22.6.1 Configuration

Start MATLAB on the login node. Configure MATLAB to run parallel jobs on your cluster by calling *configCluster*. For each cluster, *configCluster* only needs to be called once per version of MATLAB

```
configCluster
```

Jobs will now default to the cluster rather than submit to the local machine (the login node in this case).

22.6.2 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 *ProjectCode* and *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 = '';
```

22.6.3 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 idx = 1:1000
  a(idx) = ...
end
```

Once you have finished using the pool, delete it

```
% Delete the pool
p.delete
```

22.6.4 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*

```
% 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

```
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

```
% Get a handle on job with ID 2.
j2 = c.Jobs(2);
```

NOTE: You can view a list of your jobs, as well as their IDs, using the above *c.Jobs* command

```
% 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.

22.6.5 Parallel jobs

Users can also submit parallel workflows with batch. You can use the following example (*parallel_example.m*) for a parallel job

```
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.')
```

Use the *batch* command again, but since you are running a parallel job, you also specify a MATLAB Pool

```
% 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.

```
% 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 =

4
```

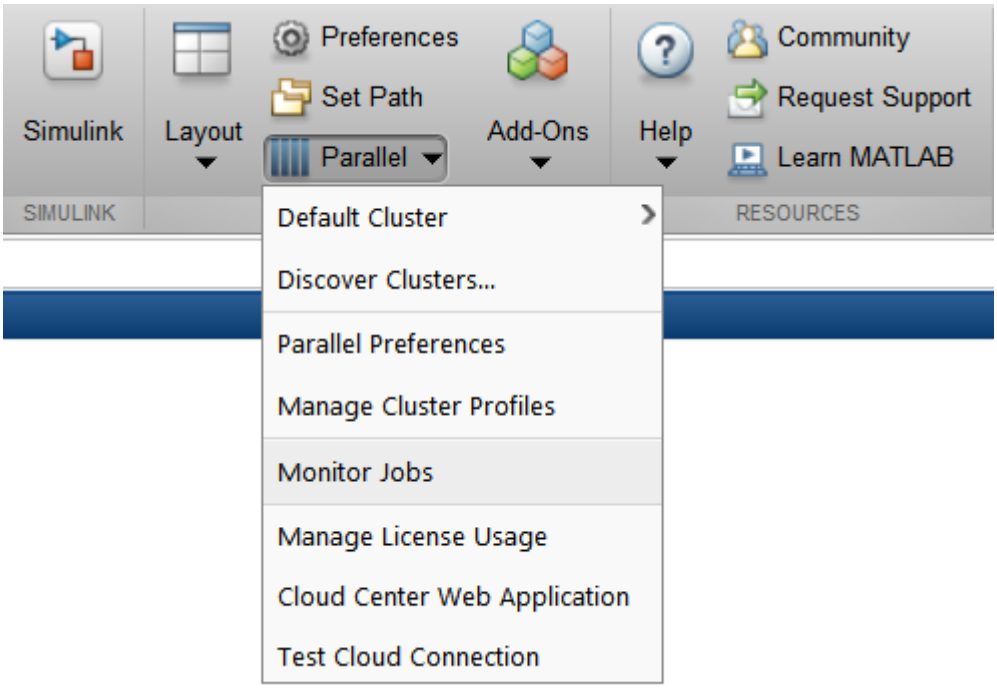
```
% Clear workspace, as though you have quit MATLAB.
clear j
```

Once you have a handle to the cluster, call the *findJob* method to search for the job with the specified job ID

```
% 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{:};

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).



22.6.6 Debugging

If a serial job produces an error, you can call the *getDebugLog* method to view the error log file

```
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

```
j.Parent.getDebugLog(j)
```

The scheduler ID can be derived by calling *schedID*

```
schedID(j)

ans

25539
```

22.6.7 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

- Parallel Computing Webinars

22.7 GPUs

Calculations using GPUs can be done using the *GPU nodes*. This can be done using MATLAB within a Slurm 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.

CHAPTER

TWENTYTHREE

NAMD

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 across 2 nodes (72 cores) with 2 tasks per node and 18 cores per task, one of which is reserved for communications.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=NAMD_Example
#SBATCH --time=01:00:00
#SBATCH --exclusive
#SBATCH --nodes=2
#SBATCH --tasks-per-node=2
#SBATCH --cpus-per-task=18
#SBATCH --account=[budget code]
#SBATCH --partition=standard
#SBATCH --qos=standard
```

(continues on next page)

(continued from previous page)

```
module load namd/2.14

export OMP_NUM_THREADS=18
export OMP_PLACES=cores

srun namd2 +setcpuaffinity +isomalloc_sync +ppn 17 +pemap 1-17,19-35 +commap 0,18 input.
↪namd
```

NAMD can also be run without SMP.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=NAMD_Example
#SBATCH --time=01:00:00
#SBATCH --exclusive
#SBATCH --nodes=2
#SBATCH --account=[budget code]
#SBATCH --partition=standard
#SBATCH --qos=standard

module load namd/2.14-nosmp

srun namd2 +setcpuaffinity +isomalloc_sync input.namd
```

And, finally, there's also a GPU version. The example below runs ten NAMD worker threads on one GPU.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=NAMD_Example
#SBATCH --time=01:00:00
#SBATCH --nodes=1
#SBATCH --account=[budget code]
#SBATCH --partition=gpu-cascade
#SBATCH --qos=gpu
#SBATCH --gres=gpu:1

module load namd/2.14-gpu

export OMP_NUM_THREADS=10
export OMP_PLACES=cores

srun --distribution=block:block --hint=nomultithread \
    namd2 +setcpuaffinity +isomalloc_sync +idlepoll \
        +ppn ${OMP_NUM_THREADS} +devices 0 input.namd
```

CHAPTER

TWENTYFOUR

OPENFOAM

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

OpenFOAM comes in a number of different flavours. The two main releases are from https://openfoam.org/ and from https://www.openfoam.com/.

You can query the versions of OpenFOAM are currently available on Cirrus from the command line with `module avail openfoam`.

Versions from https://openfoam.org/ are typically `v8` etc, while versions from https://www.openfoam.com/ are typically `v2006` (released June 2020).

24.2 Useful Links

- OpenFOAM Documentation

24.3 Using OpenFOAM on Cirrus

Any batch script which intends to use OpenFOAM should first load the appropriate `openfoam` module. You then need to source the `etc/bashrc` file provided by OpenFOAM to set all the relevant environment variables. The relevant command is printed to screen when the module is loaded. For example, for OpenFOAM v8:

```
module add openfoam/v8.0
source ${FOAM_INSTALL_PATH}/etc/bashrc
```

You should then be able to use OpenFOAM in the usual way.

24.4 Example Batch Submisison

The following example batch submission script would run OpenFOAM on two nodes, with 36 MPI tasks per node.

```
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=36
#SBATCH --exclusive
#SBATCH --time=00:10:00

#SBATCH --partition=standard
#SBATCH --qos=standard

# Load the openfoam module and source the bashrc file

module load openfoam/v8.0
source ${FOAM_INSTALL_PATH}/etc/bashrc

# Compose OpenFOAM work in the usual way, except that parallel
# executables are launched via srun. For example:

srun interFoam -parallel
```

A SLURM submission script would usually also contain an account token of the form

```
#SBATCH --account=your_account_here
```

where the *your_account_here* should be replaced by the relevant token for your account. This is available from SAFE with your budget details.

CHAPTER

TWENTYFIVE

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).

```
#!/bin/bash
#
# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=pw_test
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --time=0:20:0
# Make sure you are not sharing nodes with other users
#SBATCH --exclusive


# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]
```

(continues on next page)

(continued from previous page)

```
# Load QE and MPI modules
module load quantum-espresso

# Run using input in test_calc.in
srun pw.x -i test_cals.in
```

CHAPTER

TWENTYSIX

STAR-CCM+

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+. Currently we only support Power on Demand (PoD) licenses

For queries about other types of license options please 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.

You will need to setup passwordless SSH connections to Cirrus.

26.3.1 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:

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=STAR-CCM_test
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=14
```

(continues on next page)

(continued from previous page)

```
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load the default HPE MPI environment
module load mpt
module load starccm+

export SGI_MPI_HOME=$MPI_ROOT
export PATH=$STARCCM_EXE:$PATH
export LM_LICENSE_FILE=48001@192.168.191.10
export CDLMD_LICENSE_FILE=48001@192.168.191.10

scontrol show hostnames $SLURM_NODELIST > ./starccm.launcher.host.$SLURM_JOB_ID.txt
starccm+ -clientldlibpath /scratch/sw/libnsl/1.3.0/lib/ -ldlibpath /scratch/sw/libnsl/1.
↪3.0/lib/ -power -podkey <PODkey> -licpath 48001@192.168.191.10 -server -machinefile ./
↪starccm.launcher.host.$SLURM_JOB_ID.txt -np 504 -rsh ssh
```

You should replace "<PODkey>" with your PoD licence key.

26.3.2 Automatically load and start a Star-CCM+ simulation

You can use the "-batch" option to automatically load and start a Star-CCM+ simulation.

Your submission script will look like this (the only difference with the previous examples is the "starccm+" line)

```
#!/bin/bash

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=STAR-CCM_test
#SBATCH --time=0:20:0
#SBATCH --exclusive
#SBATCH --nodes=14
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1

# Replace [budget code] below with your budget code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load the default HPE MPI environment
module load mpt
module load starccm+
```

(continues on next page)

(continued from previous page)

```
export SGI_MPI_HOME=$MPI_ROOT
export PATH=$STARCCM_EXE:$PATH
export LM_LICENSE_FILE=48001@192.168.191.10
export CDLMD_LICENSE_FILE=48001@192.168.191.10

scontrol show hostnames $SLURM_NODELIST > ./starccm.launcher.host.$SLURM_JOB_ID.txt
starccm+ -clientldlibpath /scratch/sw/libnsl/1.3.0/lib/ -ldlibpath /scratch/sw/libnsl/1.
↪3.0/lib/ -power -podkey <PODkey> -licpath 48001@192.168.191.10 -batch simulation.java -
↪machinefile ./starccm.launcher.host.$SLURM_JOB_ID.txt -np 504 -rsh ssh
```

This script will load the file "simulation.java". You can find instructions on how to write a suitable "simulation.java" here

The file "simulation.java" must be in the same directory as your Slurm submission script (or you can provide a full path).

26.3.3 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.

CHAPTER

TWENTYSEVEN

VASP

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. VASP 5 and VASP 6 are separate packages on Cirrus and requests for access need to be made separately for the two versions via SAFE.

If you have a VASP 5 or VASP 6 licence and wish to have access to VASP on Cirrus please request access through SAFE:

- How to request access to package groups

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

# job options (name, compute nodes, job time)
#SBATCH --job-name=VASP_test
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --exclusive
#SBATCH --time=0:20:0

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]

# Load VASP version 5 module
module load vasp/5

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

# Run standard VASP executable
srun vasp_std
```

CHAPTER

TWENTYEIGHT

SPECFEM3D CARTESIAN

SPECFEM3D Cartesian, simulates acoustic (fluid), elastic (solid), coupled acoustic/elastic, poroelastic or seismic wave propagation in any type of conforming mesh of hexahedra (structured or not.) It can, for instance, model seismic waves propagating in sedimentary basins or any other regional geological model following earthquakes. It can also be used for non-destructive testing or for ocean acoustics.

28.1 Useful Links

- SPECFEM3D User Resources
- SPECFEM3D Wiki

28.2 Using SPECFEM3D Cartesian on Cirrus

SPECFEM3D Cartesian is freely available to all Cirrus users.

28.3 Running parallel SPECFEM3D Cartesian jobs

SPECFEM3D can exploit multiple nodes on Cirrus and will generally be run in exclusive mode over more than one node. Furthermore, it can be run on the GPU nodes.

For example, the following script will run a SPECFEM3D job using 4 nodes (144 cores) with pure MPI.

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

# Slurm job options (name, compute nodes, job time)
#SBATCH --job-name=SPECFEM3D_Example
#SBATCH --time=1:0:0
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1


# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
# Replace [partition name] below with your partition name (e.g. standard,gpu-skylake)
```

(continues on next page)

(continued from previous page)

```
#SBATCH --partition=[partition name]
# Replace [qos name] below with your qos name (e.g. standard,long,gpu)
#SBATCH --qos=[qos name]


# Load SPECFEM3D module
module load specfem3d

# Run using input in input.namd
srun xspecfem3D
```

CHAPTER

TWENTYNINE

INTEL MKL: BLAS, LAPACK, SCALAPACK

The Intel Maths Kernel Libraries (MKL) contain a variety of optimised numerical libraries including BLAS, LAPACK, and ScaLAPACK. In general, the exact commands required to build against MKL depend on the details of compiler, environment, requirements for parallelism, and so on. The Intel MKL link line advisor should be consulted.

See https://software.intel.com/content/www/us/en/develop/articles/intel-mkl-link-line-advisor.html

Some examples are given below. Note that loading the appropriate intel tools module will provide the environment variable *MKLROOT* which holds the location of the various MKL components.

29.1 Intel Compilers

29.1.1 BLAS and LAPACK

To use MKL libraries with the Intel compilers you just need to load the relevant Intel compiler module, and the Intel `cmkl` module, e.g.:

```
module load intel-20.4/fc
module load intel-20.4/cmkl
```

To include MKL you specify the `-mkl` option on your compile and link lines. For example, to compile a simple Fortran program with MKL you could use:

```
ifort -c -mkl -o lapack_prb.o lapack_prb.f90
ifort -mkl -o lapack_prb.x 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`.

29.1.2 ScaLAPACK

The distributed memory linear algebra routines in ScaLAPACK require MPI in addition to the compiler and MKL libraries. Here we use Intel MPI via:

```
module load intel-20.4/fc
module load intel-20.4/mpi
module load intel-20.4/cmkl
```

ScaLAPACK requires the Intel versions of BLACS at link time in addition to ScaLAPACK libraries; remember also to use the MPI versions of the compilers:

```
mpiifort -c -o linsolve.o linsolve.f90
mpiifort -o linsolve.x linsolve.o -L${MKLROOT}/lib/intel64 \
-lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
-lmkl_blacs_intelmpi_lp64 -lpthread -lm -ldl
```

29.2 GNU Compiler

29.2.1 BLAS and LAPACK

To use MKL libraries with the GNU compiler you first need to load the GNU compiler module and Intel MKL module, e.g.,:

```
module load gcc
module load intel-20.4/cmkl
```

To include MKL you need to link explicitly against the MKL libraries. For example, to compile a single source file Fortran program with MKL you could use:

```
gfortran -c -o lapack_prb.o lapack_prb.f90
gfortran -o lapack_prb.x lapack_prb.o -L$MKLROOT/lib/intel64 \
-lmkl_gf_lp64 -lmkl_core -lmkl_sequential
```

This will build against the serial version of MKL; to build against the threaded version use:

```
gfortran -c -o lapack_prb.o lapack_prb.f90
gfortran -fopenmp -o lapack_prb.x lapack_prb.o -L$MKLROOT/lib/intel64 \
-lmkl_gf_lp64 -lmkl_core -lmkl_gnu_thread
```

29.2.2 ScaLAPACK

The distributed memory linear algebra routines in ScaLAPACK require MPI in addition to the MKL libraries. On Cirrus, this is usually provided by SGI MPT.

```
module load gcc
module load mpt
module load intel-20.4/cmkl
```

Once you have the modules loaded you need to link against two additional libraries to include ScaLAPACK. Note we use here the relevant `mkl_blacs_sgimpt_lp64` version of the BLACS library. Remember to use the MPI versions of the compilers:

```
mpif90 -f90=gfortran -c -o linsolve.o linsolve.f90
mpif90 -f90=gfortran -o linsolve.x linsolve.o -L${MKLROOT}/lib/intel64 \
-lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
-lmkl_blacs_sgimpt_lp64 -lpthread -lm -ldl
```

29.2.3 ILP vs LP interface layer

Many applications will use 32-bit (4-byte) integers. This means the MKL 32-bit integer interface should be selected (which gives the `_lp64` extensions seen in the examples above).

For applications which require, e.g., very large array indices (greater than 2^31-1 elements), the 64-bit integer interface is required. This gives rise to `_ilp64` appended to library names. This may also require `-DMKL_ILP64` at the compilation stage. Check the Intel link line advisor for specific cases.

CHAPTER

THIRTY

HDF5

Serial and parallel versions of HDF5 are available on Cirrus.

Table 1: Parallel HDF5 modules on Cirrus

Module name	Library version	Compiler	MPI library
hdf5parallel/1.10.4-intel18-impi18	1.10.4	Intel 18	Intel MPI 18
hdf5parallel/1.10.6-intel18-mpt222	1.10.6	Intel 18	HPE MPT 2.22
hdf5parallel/1.10.6-intel19-mpt222	1.10.6	Intel 19	HPE MPT 2.22
hdf5parallel/1.10.6-gcc6-mpt222	1.10.6	GCC 6.3.0	HPE MPT 2.22

Instructions to install a local version of HDF5 can be found on this repository: https://github.com/hpc-uk/build-instructions/tree/main/utils/HDF5

CHAPTER

THIRTYONE

PROFILING USING SCALASCA

Scalasca is installed on Cirrus, which is an open source performance profiling tool. Two versions are provided, using GCC 8.2.0 and the Intel 19 compilers; both use the HPE MPT library to provide MPI and SHMEM. An important distinction is that the GCC+MPT installation cannot be used to profile Fortran code as MPT does not provide GCC Fortran module files. To profile Fortran code, please use the Intel+MPT installation.

Loading the one of the modules will autoload the correct compiler and MPI library:

```
module load scalasca/2.6-gcc8-mpt225
```

or

```
module load scalasca/2.6-intel19-mpt225
```

Once loaded, the profiler may be run with the `scalasca` or `scan` commands, but the code must first be compiled first with the Score-P instrumentation wrapper tool. This is done by prepending the compilation commands with `scorep`, e.g.:

```
scorep mpicc -c main.c -o main
scorep mpif90 -openmp main.f90 -o main
```

Advanced users may also wish to make use of the Score-P API. This allows you to manually define function and region entry and exit points.

You can then profile the execution during a Slurm job by prepending your `srun` commands with one of the equivalent commands `scalasca -analyze` or `scan -s`:

```
scalasca -analyze srun ./main
scan -s srun ./main
```

You will see some output from Scalasca to stdout during the run. Included in that output will be the name of an experiment archive directory, starting with *scorep_*, which will be created in the working directory. If you want, you can set the name of the directory by exporting the `SCOREP_EXPERIMENT_DIRECTORY` environment variable in your job script.

There is an associated GUI called Cube which can be used to process and examine the experiment results, allowing you to understand your code's performance. This has been made available via a Singularity container. To start it, run the command `cube` followed by the file in the experiment archive directory ending in *.cubex* (or alternatively the whole archive), as seen below:

```
cube scorep_exp_1/profile.cubex
```

The Scalasca quick reference guide found here provides a good overview of the toolset's use, from instrumentation and use of the API to analysis with Cube.

CHAPTER

THIRTYTWO

INTEL VTUNE

32.1 Profiling using VTune

Intel VTune allows profiling of compiled codes, and is particularly suited to analysing high performance applications involving threads (OpenMP), and MPI (or some combination thereof).

Using VTune is a two-stage process. First, an application is compiled using an appropriate Intel compiler and run in a "collection" phase. The results are stored to file, and may then be inspected interactively via the VTune GUI.

32.2 Collection

Compile the application in the normal way, and run a batch job in exclusive mode to ensure the node is not shared with other jobs. An example is given below.

Collection of performance data is based on a `collect` option, which defines which set of hardware counters are monitered in a given run. As not all counters are available at the same time, a number of different collections are available. A different one may be relevant if interested in different aspects of performance. Some standard options are:

`vtune -collect=performance-snapshot` may be used to product a text summary of performance (typically to standard output), which can be used as a basis for further investigation.

`vtune -collect=hotspots` produces a more detailed analysis which can be used to inspect time taken per function and per line of code.

`vtune -collect=hpc-performance` may be useful for HPC codes.

`vtune --collect=meory-access` will provide figures for memory-related measures including application memory bandwidth.

Use `vtune --help collect` for a full summary of collection options. Note that not all options are available (e.g., prefer NVIDIA profiling for GPU codes).

32.2.1 Example SLURM script

Here we give an example of profiling an application which has been compiled with Intel 20.4 and requests the `memory-access` collection. We assume the application involves OpenMP threads, but no MPI.

```
#!/bin/bash

#SBATCH --time=00:10:00
#SBATCH --nodes=1
```

(continues on next page)

(continued from previous page)

```
#SBATCH --exclusive

#SBATCH --partition=standard
#SBATCH --qos=standard

export OMP_NUM_THREADS=18

# Load relevant (cf. compile-time) Intel options
module load intel-20.4/compilers
module load intel-20.4/vtune

vtune -collect=memory-access -r results-memory ./my_application
```

Profiling will generate a certain amount of additional text information; this appears on standard output. Detailed profiling data will be stored in various files in a sub-directory, the name of which can be specified using the `-r` option.

Notes

- Older Intel compilers use `amplxe-cl` instead of `vtune` as the command for collection. Some existing features still reflect this older name. Older versions do not offer the "performance-snapshot" collection option.
- Extra time should be allowed in the wall clock time limit to allow for processing of the profiling data by `vtune` at the end of the run. In general, a short run of the application (a few minutes at most) should be tried first.
- A warning may be issued:

  > amplxe: Warning: Access to /proc/kallsyms file is limited. Consider changing /proc/sys/kernel/kptr_restrict to 0 to enable resolution of OS kernel and kernel modules symbols.

  This may be safely ignored.

- A warning may be issued:

  > amplxe: Warning: The specified data limit of 500 MB is reached. Data collection is stopped. amplxe: Collection detached.

  This can be safely ignored, as a working result will still be obtained. It is possible to increase the limit via the `-data-limit` option (500 MB is the default). However, larger data files can take an extremely long time to process in the report stage at the end of the run, and so the option is not recommended.

- For Intel 20.4, the `--collect=hostspots` option has been observed to be problematic. We suggest it is not used.

32.2.2 Profiling an MPI code

Intel VTune can also be used to profile MPI codes. It is recommended that the relavant Intel MPI module is used for compilation. The following example uses Intel 18 with the older `amplxe-cl` command:

```
#!/bin/bash

#SBATCH --time=00:10:00
#SBATCH --nodes=2
#SBATCH --exclusive

#SBATCH --partition=standard
```

(continues on next page)

(continued from previous page)

```
#SBATCH --qos=standard

export OMP_NUM_THREADS=18

module load intel-mpi-18
module load intel-compilers-18
module load intel-vtune-18

mpirun -np 4 -ppn 2 amplxe-cl -collect hotspots -r vtune-hotspots \
       ./my_application
```

Note that the Intel MPI launcher `mpirun` is used, and this precedes the VTune command. The example runs a total of 4 MPI tasks (`-np 4`) with two tasks per node (`-ppn 2`). Each task runs 18 OpenMP threads.

32.3 Viewing the results

We recommend that the latest version of the VTune GUI is used to view results; this can be run interactively with an appropriate X connection. The latest version is available via

```
$ module load oneapi
$ module load vtune/latest
$ vtune-gui
```

From the GUI, navigate to the appropriate results file to load the analysis. Note that the latest version of VTune will be able to read results generated with previous versions of the Intel compilers.