ARC Software Guide

Release 0.1

The ARC Team

Jan 26, 2023

CONTENTS

This documentation provides information on the software applications installed on ARC clusters and also provides template submission scripts for some of the most popular applications.

CHAPTER

ONE

ACCESSING INSTALLED SOFTWARE APPLICATIONS

1.1 Environment Modules

The Linux operating system makes extensive use of the "working environment", which is a collection of individual environment variables. An environment variable is a named object in the Linux shell that contains information used by one or more applications; two of the most used such variables are **$HOME**, which defines a user's home directory name, and **$PATH**, which represents a list paths to different executables. A large number of environment variables are already defined when a Linux shell is open but the environment can be customised, either by defining new environment variables relevant to certain applications (e.g. software license variables) or by modifying existing ones (e.g. adding a new path to **$PATH**).

`module` is a Linux utility, which is used to manage of working environment in preparation for running the applications installed on the ARC systems. By loading the module for a certain installed application, the environment variables that are relevant for that application are automatically defined or modified.

The ARC/HTC software environment comprises a mixture of commercial applications, software built using the Easy-Build framework and software built using our own local build recipes. You should use the environment modules system (via the `module` command) to load applications into your environment on ARC/HTC.

Because the EasyBuild framework adds many new module components into the module list - the best way to search for an application you require on the system is by using the `module spider` command.

If you wish to browse a current list ARC modules online, you can find it here: ARC Module List

---

Note: The ARC cluster login nodes reference a **different** set of modules than the main cluster, so to be sure that the module (or module version) you require is available for use on ARC you should **always** search for modules in an interactive session on a compute node. e.g:

```
[software@arc-login01 ~]$ srun -p interactive --pty /bin/bash
srun: CPU resource required, checking settings/requirements...

[software@arc-c304 ~]$ module spider <name>
```

So, for example, to search for the GROMACS application:

```
[software@arc-login01 ~]$ srun -p interactive --pty /bin/bash
srun: CPU resource required, checking settings/requirements...

[software@arc-c304 ~]$ module spider gromacs

-----------------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-------------------------------------
  GROMACS:
-----------------------------------------------------------------------------------------------
↪-------------------------------------
  Description:
    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. This is a CPU only build, containing␣
↪both MPI and threadMPI builds.

   Versions:
      GROMACS/2020-fosscuda-2019b
      GROMACS/2020.4-foss-2020a-PLUMED-2.6.2
      GROMACS/2020.4-foss-2020a
```

Note: `module spider` is NOT case-sensitive for searching, so:

```
module spider GROMACS
module spider gromacs
module spider Gromacs
```

…are all equivalent. However, when loading module using module load you must use the correct case, for example:

```
[software@arc-c304 ~]$ module load GROMACS/2020.4-foss-2020a
```

If the software name you are using in "module spider" returns too many options you can use

```
module -r spider '^name'
```

for example:

```
module -r '^Python'
module -r ^R$'
```

You can also build your own software in your home or data directories using one of the compilers provided (which are also available through the environment modules system). Typically the compiler toolchains, including maths libraries and MPI can be loaded using the modules named foss (e.g. foss/2020a) for free open-source software (i.e. GCC) or intel (e.g. intel/2020a) for the Intel compiler suite.

If no version is specified, the default version of the software is loaded (usually the latest version):

```
module load GROMACS
module list GROMACS

Currently Loaded Modules Matching: GROMACS
  1) GROMACS/2020.4-foss-2020a
```

Specific versions, other than the default can be loaded by specifying the version:

```
module load GROMACS/2020.4-foss-2020a-PLUMED-2.6.2
module list GROMACS
```

(continues on next page)

(continued from previous page)

```
Currently Loaded Modules Matching: GROMACS
  1) GROMACS/2020.4-foss-2020a-PLUMED-2.6.2
```

A module can be "unloaded" with the unload option, for example:

```
module unload MATLAB/2020b
```

1.2 Building software against installed modules

If you need to compile your own software but would like to use an ARC built module for its libraries/headers. You need to make use of the `EBROOT` environment variable which is defined when you load the module. For example: if you need to build against the `Boost` libraries you first need to load the module:

```
module load Boost/1.79.0-GCC-11.3.0
```

Loading the above will define `EBROOTBOOST` - the variable name is always `EBROOT` followed by the main module name - this environment variable will contain the path to the Boost software for this specific module:

```
echo $EBROOTBOOST
/apps/system/easybuild/software/Boost/1.79.0-GCC-11.3.0

ls $EBROOTBOOST
easybuild  include  lib  lib64
```

So if you need to specify the Boost location to your build, you can supply for example: `$EBROOTBOOST/include` for the header files and `$EBROOTBOOST/lib64` for the library files.

CHAPTER

TWO

INSTALLING NEW SOFTWARE APPLICATIONS

2.1 Licensed Software (Commercial or Restricted Use)

There are a number of licensed commercial or restricted use applications already installed on ARC. See our Restricted Licence Software Applications Form for a list.

We can help you install new applications of this type by hosting them centrally and restricting access by project, or you can install them yourself in your `$DATA` area following the installation guides provided by the software vendor.

> **Warning:** Do not attempt to run package managers such as `apt` or `yum` on the ARC systems these will **not** work as they will (by default) attempt to install code into protected system directories which you do not have permission to write to.
>
> If you have such requirements, contact the ARC team via support@arc.ox.ac.uk

2.2 Open Source Software

ARC uses the EasyBuild framework to manage central application software installations on the clusters. We already have a large number of applications installed on ARC and you can find a list of installed software here.

If the package you require is in this list we should be able to install this for you reasonably quickly. Use the ARC Software Installation Request Form to request this.

You can also install software in your own `$DATA` area using the Easybuild toolchains (compilers and libraries) to satisfy any dependencies that the application may have.

If you have any problems contact a member of the ARC team via support@arc.ox.ac.uk

CHAPTER

THREE

EFFICIENTLY RUNNING SERIAL SOFTWARE

3.1 Introduction

Processing workloads often involve serial applications (i.e. single-threaded, single processes), which run on a single core. Jobs like this should preferably be run on HTC (not ARC); the scheduler on HTC is optimised for small jobs, as is the hardware; ARC is optimised to prefer large parallel applications. HTC is the better cluster to use for serial applications.

This guide gives tips on how to design a job script for a serial application. The assumption is the workload involves a large number of similar but independent runs of the same application with different input parameters.

3.2 Using job arrays

In the scenario of a workload with a large number of independent runs, packing several runs inside a job works well with job arrays. Using job arrays, a number of jobs can be started with a single submission command and a single submission script. The job array index can be used within the submission script to identify the parameters each process works with (input or output files, command line parameters, etc.).

To give an example, assume `serialapp` has to run a parameter sweep, with values ranging from `1.01` to `1.96` with a step of `0.01` All the processing can be carried out by submitting one job with the following job script:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --job-name=test
#SBATCH --time=00:30:00

RUN_PARAM=$(printf "1.%02d\n" $SLURM_ARRAY_TASK_ID)

serialapp $RUN_PARAM
```

The job can be submitted using the command:

```
sbatch --array=1-96
```

The variable SLURM_ARRAY_TASK_ID is the job array index (set by SLURM when an array job is submitted) and runs from 1 to 96, in accordance with the submission command above. It could also be used to, for example, run a specific line out of a file, or identify an input file.

There are other ways to control the input parameters per task; very common is to list them in a file and read one line per task out of that file (corresponding to the task ID).

For example; say serialapp requires input parameters more like:

```
serialapp -a X -b Y -c Z
```

and you need to run 5 copies of it with different values of X, Y and Z. You could generate a file called "job_parameters" listing all the parameters for each run one one line:

```
-a 1 -b 2 -c 3
-a 2 -b 4 -c 6
-a 3 -b 6 -c 9
-a 4 -b 8 -c 12
-a 5 -b 10 -c 15
```

and then submit a 5 task array, with each task reading the line corresponding to it's task ID from the parameter file:

```
#SBATCH --nodes=1
#SBATCH --job-name=test
#SBATCH --time=00:30:00

RUN_PARAM=$(sed -n ${SLURM_ARRAY_TASK_ID}p job_parameters)

serialapp $RUN_PARAM
```

This job could be submitted using the command:

```
sbatch --array=1-5
```

CHAPTER

FOUR

COMPILING AND RUNNING MPI SOFTWARE

4.1 Introduction

This guide is intended to give an overview of what is needed to compile and run MPI software on the ARC cluster systems.

The guide shows how to:

- Compile a MPI application,
- Prepare a job submission script and
- Submit the job.

4.2 About MPI

MPI stands for Message Passing Interface, an interface standard that defines a number of library routines aimed at the programming of message-passing (distributed-processing) applications. The interface specifications were designed by a group of researchers from both academia and industry and cover bindings for C, C++ and Fortran.

Being standardised, MPI programming leads to highly portable code. Nevertheless, the MPI standard has many implementations in libraries (both commercial and open source software), and the quality and performance of MPI libraries can differ significantly.

Any MPI library implementation has a number of tools that help programmers build and run MPI applications. The main tools are:

compiler utilities and an application run agent.

Compiler utilities (`mpicc`, `mpiicc`, `mpicxx`, `mpif77`, `mpif90`, `mpiifort`) are used to compile and link MPI programs. These are not compilers as such but wrappers around back-end compilers (e.g. the GNU or Intel compilers) and are designed to make compiling and linking against the MPI library easy.

The run agent launches and manages the execution of a MPI executable on distributed computer systems. This agent is called mpirun or mpiexec, with mpirun being the most frequently used one.

4.3 MPI on the ARC systems

The ARC clusters have two main MPI implementations installed, however this guide is intended to be independent of any particular flavour of MPI. The MPI libraries available per cluster system are presented below.

The MPI implementations OpenMPI and Intel-MPI are installed on the clusters ARC and HTC, are optimised and configured to use the InfiniBand interconnect. Each MPI implementation has several versions installed, and may be used with different compilers. All installations are managed through the environment module system.

4.4 Preparing and Running An Example

Preparation

Log in to one of the ARC clusters and ensure you are running on an interactive node (this is important!), create a directory in which to do some work and go to it. The sequence of commands is:

```
srun -p interactive --pty /bin/bash
cd $DATA
mkdir examples
cd examples
```

Then, copy the ARC MPI example files to your newly created directory:

```
cp /apps/common/examples/mpi/* .
```

Run the command `ls` to list the copied files. Simple C `cluster_myprog.c` and Fortran `cluster_myprog.f` MPI example codes are provided. Also, there is a submission script `slurm.sh` You can edit and adapt the submission script for the cluster on which you are running the example.

4.5 Compiling the application

The compilation and linking of an MPI program is managed by the compiler wrappers `mpicc` and `mpif77` for GCC and `mpiicc` and `mpiifort` for Intel - and performed by the back-end compiler. The MPI wrapper scripts ensure the correct options for MPI operation are supplied to the compiler.

4.6 Toolchains

The ARC and HTC systems have a number of compiler, MPI and maths library combinations grouped into toolchains which are versioned every six months (a and b versions). These are based upon the EasyBuild standard toolchain definitions to ensure reproducability. For Intel compilers these are named intel and for GCC they are named foss (free open-source software).

For example the `intel/2020a` toolchain contains the following components:

```
module load intel/2020a
module list

Currently Loaded Modules:
  1) GCCcore/9.3.0               3) binutils/2.34-GCCcore-9.3.0   5) impi/2019.7.217-
↪iccifort-2020.1.217   7) imkl/2020.1.217-iimpi-2020a
```

(continues on next page)

(continued from previous page)

```
  2) zlib/1.2.11-GCCcore-9.3.0   4) iccifort/2020.1.217           6) iimpi/2020a        ␣
↪                  8) intel/2020a
```

The `foss/2020a` toolchain contains:

```
module load foss/2020a
module list

Currently Loaded Modules:
  1) GCCcore/9.3.0                 4) GCC/9.3.0                      7) libxml2/2.9.10-
↪GCCcore-9.3.0     10) OpenMPI/4.0.3-GCC-9.3.0   13) FFTW/3.3.8-gompi-2020a
  2) zlib/1.2.11-GCCcore-9.3.0     5) numactl/2.0.13-GCCcore-9.3.0   8) libpciaccess/0.
↪16-GCCcore-9.3.0  11) OpenBLAS/0.3.9-GCC-9.3.0  14) ScaLAPACK/2.1.0-gompi-2020a
  3) binutils/2.34-GCCcore-9.3.0   6) XZ/5.2.5-GCCcore-9.3.0         9) hwloc/2.2.0-
↪GCCcore-9.3.0        12) gompi/2020a               15) foss/2020a
```

Important Note for Intel toolchain users: When using the intel toolchain, the MPI build wrappers `mpicc`, `mpicxx` and `mpifc` point to the GCC compilers. To use the Intel compilers you should use the wrappers: `mpiicc`, `mpiicpc` and `mpiifort` respectively. If you are using a third-party build which cannot be easily modified, you can override the behaviour of the `mpicc`, `mpicxx` and `mpifc` wrappers to use Intel compilers by setting the following environment variables:

```
export MPICH_CC=icc

export MPICH_FC=ifort
export MPICH_F90=ifort
export MPICH_F77=ifort

export MPICH_CPP="icc -E"

export MPICH_CXX=icpc
export MPICH_CCC=icpc
```

Other toolchains/versions can be made available, a list of EasyBuild supported versions can be found here. Please note that the ARC systems only support `foss/2018b` and newer, and `intel/2020a` and newer - due to operating system compatibility.

4.7 Compilation

After loading your chosen toolchain module, compile one of the source files:

For the `foss` toolchain use:

```
mpicc cluster_myprog.c -o cluster_myprog
```

Or (for the Fortran code):

```
mpif77 cluster_myprog.f -o cluster_myprog
```

For the `intel` toolchain use:

```
mpiicc cluster_myprog.c -o cluster_myprog
```

Or (for the Fortran code):

```
mpiifort cluster_myprog.f -o cluster_myprog
```

Run the `ls` command to verify the executable cluster_myprog was created.

4.8 Preparing the submission script

Edit the submission script provided `slurm.sh` to input the details of the job. The key lines to pay attention to in the script are:

- the request for resources (number of nodes and walltime)
- the chosen toolchain and
- the mpirun command.

The submission script should look like this for a foss toolchain build:

```
#!/bin/bash

#SBATCH --job-name=myprog
#SBATCH --time=00:10:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=my.name@email.com

module load foss/2020a

mpirun ./cluster_myprog
```

or for an `intel` toolchain build:

```
#!/bin/bash

#SBATCH --job-name=myprog
#SBATCH --time=00:10:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=my.name@email.com

module load intel/2020a

mpirun ./cluster_myprog
```

In this example, SLURM is instructed to allocate 2 nodes `--nodes=2` for 10 minutes `--time=00:10:00` Also, the run is scheduled for 8 MPI processes per node; this maps each MPI process to a physical core, leading to a (generally) optimal run configuration.

N.B. In ARC there are 48 cores per node but in this example we are only using 8 cores per node.

The command line `mpirun ./cluster_myprog` runs the executable `cluster_myprog` built with the approprate toolchain MPI library.

4.9 Running the application

After having prepared the submission script, submit the job with:

```
sbatch slurm.sh
```

This will print a job number and return control to the Linux prompt at once. Monitor its execution using the SLURM `squeue` command.

4.10 Checking the results

After the job is run, you should have two email notifications (one for the start of the job, one for its end) and a couple of extra files in your directory. The SLURM scheduler will create a single output file, slurm-XXXX.out. [where XXXX is the JobId number]

The output file slurm-XXXX.out should contain the output from the execution, which can be seen by doing for example:

```
cat slurm-XXXX.out
```

The output should look like this (the exact execution of processes is out of order due to the parallelisation):

```
Process  2  received  from process  1
Process  9  received  from process  4
Process  1  received  from process  0
Process  15 received  from process  14
Process  11 received  from process  10
Process  13 received  from process  12
Process  4  received  from process  3
Process  6  received  from process  5
Process  12 received  from process  11
Process  10 received  from process  9
Process  7  received  from process  6
Process  8  received  from process  7
Process  0  received  from process  16
Process  2  received  from process  1
Process  3  received  from process  2
Process  5  received  from process  4
Process  14 received  from process  13
```

4.11 MPI Core Allocation (and OpenMP)

In the above examples we have used the SLURM `--ntasks-per-node` option to allocate a single CPU core to each MPI process. There may be occasions where we want to run fewer MPI processes per node, and use insead OpenMP for the remaining allocated cores. We can do this using the `--cpus-per-task` option.

Below is an example submission script (for OpenMPI) which requests two nodes with 1 MPI process each, where each MPI process can use 8 cores (for OpenMP) - so a total allocation of 16 cores:

```
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --time=00:10:00
#SBATCH --partition=devel

module load mpitest/1.0

mpirun --map-by numa:pe=${SLURM_CPUS_PER_TASK} mpisize
```

The command from the `mpitest module`, named mpisize outputs the following information:

```
Hello from host "arc-c303". This is MPI task 1, the total MPI Size is 2, and there are 8␣
↪CPU core(s) allocated to *this* MPI task, these being { 0 1 2 3 4 5 6 7 }
Hello from host "arc-c302". This is MPI task 0, the total MPI Size is 2, and there are 8␣
↪CPU core(s) allocated to *this* MPI task, these being { 0 1 2 3 4 5 6 7 }
```

From the results above we can see that as expected, two MPI processes ran, one on node `arc-c302` and the other on `arc-303` and each of these processes were allocaed 8 CPUs.

Note: The mpirun option `--map-by numa:pe=${SLURM_CPUS_PER_TASK}` is not required if running with Intel MPI.

CHAPTER

FIVE

LEGACY APPLICATION SOFTWARE

The ARC cluster environment has a number of compute nodes dedicated to running older "legacy" software applications. These are applications which are unable to run on the newer operating system installed on the main cluster. These nodes are kept within a "legacy" partition on the cluster.

Currently the main cluster is running CentOS 8.1 and the legacy nodes are running CentOS 7.7

The following legacy applications are supported:

- ANSYS products v18.2, v19.5, v20.2

You can access the legacy software modules by using the command:

```
module use /apps/common/legacy/modules
```

You may then use the module avail command to find the application you require.

5.1 Example Submission Scripts

Fluent Example:

```
#!/bin/bash

#SBATCH --partition=legacy
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=FluentTest

module use /apps/common/legacy/modules
module load fluent/18.2

srun hostname > hostfile.txt

fluent 2d -g -ssh -t${SLURM_NTASKS} -pinfiniband -mpi=intel -cnf=hostfile.txt -i cav.inp
```

For Fluent 19.5 change the module load in the above example to:

```
module load fluent/2019R3
```

For Fluent 20.2 change the module load to:

```
module load fluent/2020R2
```

CHAPTER

SIX

USING PYTHON ON ARC

The ARC team maintain centrally Python Anaconda installations for Anaconda 2 and Anaconda 3. These installations typically do not have any extra packages installed, they exist such that users of the ARC service can use them to install and maintaiin their own virtual environments.

6.1 Creating your own virtual environment

Note: You must create your conda environments from a SLURM interactive session. So, ensure you have an active interactive session by running:

```
srun -p interactive --pty /bin/bash
```

Running conda installations interactively on the login nodes will result in memory errors and other compatibility issues.

You should decide which version of Python you wish to use, 2 or 3. There are Anaconda modules available for both versions, the current Anaconda versions can be found by typing:

```
module spider anaconda
```

To load the version of Anaconda you want, in this example we are using the latest version, use one of the following commands:

Python 2:

```
module load Anaconda2
```

Python 3:

```
module load Anaconda3
```

or one of the specific Anaconda versions shown by `module spider`.

Once the module is loaded you can use the `conda` commands to create a virtual environment in your **$DATA** area. For example to create an environment named `myenv` in **$DATA** we can use the following commands:

```
export CONPREFIX=$DATA/myenv
```

Python 2:

```
conda create --prefix $CONPREFIX --copy python=2.7
```

Python 3:

```
conda create --prefix $CONPREFIX
```

Note: Please be aware of messages from `conda` which instruct you to run `conda init` - this command will add lines to your `~/.bashrc` file which can in **certain** circumstances cause undesirable behaviour in SLURM batch files. We recommend activating with `source activate` if issues occur in batch files.

You can now use (activate) the environment by running one of the following commands:

```
source activate $CONPREFIX
```

or:

```
conda activate $CONPREFIX
```

You can then use the `conda install` or `pip` commands to install packages. We recommend the use of `conda install` where possible to maintain package version consistency in the virtual environment. For example:

```
conda install numpy
```

or:

```
pip install numpy
```

Warning: In the above examples we use the `--prefix` option to `conda create` This is to ensure that the conda virtual environment is placed in `$DATA`. If you ommit this there is a risk that your environment will be placed in the default location which is `$HOME/.conda/envs` this will very likely over time cause you to go over quota in your `$HOME` area which will cause problems running jobs.

6.2 Conda Build Scripts

For ease of use, rather than running the conda commands from the command line, we recommend creating a small script to create the environment, such that you can re- build the environment in future, if required. For example, you could create a file named `build_env.sh` with the following contents

```
!# /bin/bash

# Load the version of Anaconda you need
module load Anaconda3

# Create an environment in $DATA and give it an appropriate name
export CONPREFIX=$DATA/envname
conda create --prefix $CONPREFIX

# Activate your environment
source activate $CONPREFIX

# Install packages...
```

(continues on next page)

(continued from previous page)

```
conda install <packagename>
..
..
```

You could then run this script with

```
sh ./build_env.sh
```

6.3 Conda Package Cache

By default Anaconda will *cache* all packages installed using `conda install` into a directory in your `$HOME` area named `~/.conda/pkgs` before installing them into your virtual environment. Over time this has the potential to put you over quota in `$HOME`

If you find yourself over quota in `$HOME` check how much space is being used in `~/.conda/pkgs`

```
cd ~/.conda
du -sh pkgs
```

The `du` command above may take some time to run. When complete, the command will show how much space is in use in `pkgs` - for example

```
12G     pkgs
```

In this case 12GB of space is being used by downloaded packages. To tidy up, run the following commands

```
module load Anaconda3
conda clean --packages --tarballs
```

You can repeat the `du` command above to check that the space has been freed.

6.4 Using Anaconda from within a submission script

In order to use your installed virtual environment from a batch script, you will need to load the appropriate Anconda module and activate your environment. Using values from the above example (and assuming Python version 3, Anaconda 2020/11):

```
# After SBATCH section of script

module load Anaconda3/2020.11
source activate $DATA/myenv

# Your Python commands here...
```

6.5 Important Anaconda Information

When using Anaconda on the ARC systems, please take note of the following:

- Do not load Anaconda virtual environments automatically on log in from your .bashrc or .bash_profile scripts. These will cause issues to SLURM submitted jobs.
- Ensure you have deactivated the virtual environment BEFORE submitting a SLURM job using sbatch, otherwise you will have issues with packages from your virtual environment not being found.
- You should load all you require from the submission script - as in the submission script example above.

6.6 Using Bioconda

Use the instructions above to create a basic Python Anaconda 2 or 3 virtual environment, then use the following commands to ensure the bioconda repostories are enabled:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Bioconda packages may then be installed by using the `conda install` command, for example to install `bwa`:

```
conda install bwa
```

6.7 Using MPI with Anaconda

The `mpi4py` package bundled with, or installed by Anaconda may not work correctly on the ARC cluster. The `mpi4py` code should be linked to an ARC compiled MPI library on the system - in order to do this in your virtual environment run the following commands:

From the login node, start an interactive session on a compute node - this is important for testing later:

```
srun --nodes=1 --ntasks-per-node=4 --partition=interactive --pty /bin/bash
```

First load the appropriate modules and create your virtual environment:

```
module purge
module load foss/2020a
module load Anaconda3/2020.11
```

In this example we are using Anaconda3 2020.11 and using OpenMPI 4.0.3 which comes as part of the `foss/2020a` toolchain.

Note: At present we do not recommend using the `foss/2021b` toolchain due to an incompatibility between OpenMPI 4.1.x and `mpi4py`.

Next we build our virtual environment, and activate it (if you already have your own environment you could use this instead, and simply activate it):

```
conda create --prefix $DATA/mpienv --copy python=3.8
source activate $DATA/mpienv
```

Now that we have an activated Anaconda virtual environment we can download and install `mpi4py`, in this case version 3.1.3:

```
wget https://bitbucket.org/mpi4py/mpi4py/downloads/mpi4py-3.1.3.tar.gz
tar -zxf mpi4py-3.1.3.tar.gz

cd mpi4py-3.1.3
python setup.py build --mpicc=`which mpicc`
python setup.py install
python setup.py clean
```

Assuming the build above completes successfully we can now test the `mpi4py` installation.

First create a file called mpihello.py containing the following Python code:

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
name=MPI.Get_processor_name()
print("hello world")
print(("name:",name,"my rank is",comm.rank))
```

As we are connected to an interactive node, and have four cores allocated (as per the original `srun` command) we can test this code as follows:

```
mpirun python mpihello.py
```

Note: we must run Python with the `mpirun` wrapper in order to execute MPI code. The output from the above command should be:

```
hello world
('name:', 'arc-c304', 'my rank is', 0)
hello world
('name:', 'arc-c304', 'my rank is', 1)
hello world
('name:', 'arc-c304', 'my rank is', 2)
hello world
('name:', 'arc-c304', 'my rank is', 3)
```

In the above each allocated core has run the code and outputted the string "hello world" and given its hostname and MPI rank.

For a more real world example, we will take the same hello world code and run it in a batch script on multiple nodes in the development partition, the example submission script is as follows, it should be located in the same directory as `mpihello.py` and named `runmpi.sh`

```
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --partition=devel
#SBATCH --job-name=PyHello
#SBATCH --time=00:10:00

module purge
module load Anaconda3/2020.11
```

(continues on next page)

(continued from previous page)

```
module load foss/2020a

source activate $DATA/mpienv

mpirun python ./mpihello.py
```

Note: There is no need to specify the number of tasks to the `mpirun` command. All ARC built MPI libraries are SLURM aware and are able to determine the task count.

Here you can see we have asked for two development nodes, with four MPI tasks on each node (a total of eight MPI tasks). Note we have also loaded BOTH the Anaconda3 and foss/2020a modules as used in the mpi4py build.

To submit the job, use the SLURM sbatch command:

```
sbatch runmpi.sh
```

Once the job completes, the SLURM output file should contain the following information from the job:

```
hello world
('name:', 'arc-c303', 'my rank is', 4)
hello world
('name:', 'arc-c303', 'my rank is', 5)
hello world
('name:', 'arc-c302', 'my rank is', 0)
hello world
('name:', 'arc-c302', 'my rank is', 1)
hello world
('name:', 'arc-c302', 'my rank is', 2)
hello world
('name:', 'arc-c302', 'my rank is', 3)
hello world
('name:', 'arc-c303', 'my rank is', 6)
hello world
('name:', 'arc-c303', 'my rank is', 7)
```

It can be seen from the above that the MPI processes are running on two hosts: `arc-c302` and `arc-c303` and there are four MPI ranks per host, as specified in the submission script.

6.8 Using Jupyter Notebooks on ARC

The easiest way to run Jupyter notebooks is by using the ARC Graphical Nodes

If you need to run on a local browser, you can follow the instructions below:

Interactive Notebook example

The following example shows how to connect to a Jupyter notebook session running on ARC. The example shows the best practice method of running the Jupyter session on an interactive node.

Note: We make the assumption that you are already connected to the university network or VPN. We also strongly recommend using MobaXterm as the SSH client if using Windows for this activity.

---

1) First we need to log in to an ARC login node, for this example we are using `arc-login.arc.ox.ac.uk` and start an interactive session using the standard `srun` command:

```
srun --nodes=1 --ntasks-per-node=4 --partition=interactive --pty /bin/bash
```

Once the `srun` allocation is started on the interactive node (for example: `arc-c304`) you will get a command prompt and can load the Anaconda module and start Jupyter with the options shown below:

```
module load Anaconda3/2021.11
jupyter notebook --no-browser --ip=*
```

Note: It is essential to use Anaconda3 version 2021.11 or newer for notebooks to work on ARC

The response from Jupyter will be of the following form:

```
[W 16:01:05.472 NotebookApp] WARNING: The notebook server is listening on all IP␣
↪addresses and not using encryption. This is not recommended.
[W 2022-03-24 16:01:05.689 LabApp] 'ip' has moved from NotebookApp to ServerApp. This␣
↪config will be passed to ServerApp. Be sure to update your config before our next␣
↪release.
[W 2022-03-24 16:01:05.689 LabApp] 'ip' has moved from NotebookApp to ServerApp. This␣
↪config will be passed to ServerApp. Be sure to update your config before our next␣
↪release.
[W 2022-03-24 16:01:05.689 LabApp] 'ip' has moved from NotebookApp to ServerApp. This␣
↪config will be passed to ServerApp. Be sure to update your config before our next␣
↪release.
[I 2022-03-24 16:01:05.695 LabApp] JupyterLab extension loaded from /apps/system/
↪easybuild/software/Anaconda3/2021.11/lib/python3.9/site-packages/jupyterlab
[I 2022-03-24 16:01:05.695 LabApp] JupyterLab application directory is /apps/system/
↪easybuild/software/Anaconda3/2021.11/share/jupyter/lab
[I 16:01:05.699 NotebookApp] Serving notebooks from local directory: /home/ouit0554
[I 16:01:05.699 NotebookApp] Jupyter Notebook 6.4.5 is running at:
[I 16:01:05.699 NotebookApp] http://arc-c305:8888/?
↪token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf
[I 16:01:05.699 NotebookApp]  or http://127.0.0.1:8888/?
↪token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf
[I 16:01:05.699 NotebookApp] Use Control-C to stop this server and shut down all kernels␣
↪(twice to skip confirmation).
[C 16:01:05.704 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/ouit0554/.local/share/jupyter/runtime/nbserver-2598622-open.html
    Or copy and paste one of these URLs:
       http://arc-c305:8888/?token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf
    or http://127.0.0.1:8888/?token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf
```

Note: In the above output, the last few lines contain key information. You will use the information from the last two lines to make a connection to the Jupyter server.

In our example above we can see (from the penultimate line) that the server is running on node arc-c305 and the port is 8888:

`http://arc-c305:8888/?token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf`

We are also given a URL for accessing the notebook in your local browser (on the last line). For example:

`http://127.0.0.1:8888/?token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf`

These values **will** be different for each run and you must make a note of them.

You can now minimise (but do not close) your SSH window to the interactive session above.

We now need to tunnel the Jupyter port we have just created to your local desktop.

Open new terminal window on your local machine…

The format of the command we need to run to make the tunnel is as follows, but you will need to make changes:

```
ssh -L 8888:arc-c305:8888 ouit0554@arc-login.arc.ox.ac.uk
```

In the above you will need to substitute the server node and port number from the above information and also use your username before the @ character in the connection to `arc-login.arc.ox.ac.uk`

For clarity here is an explanation of the fields in the above command:

```
ssh -L [local port]:[remote host]:[remote port] [ARC username]@arc-login.arc.ox.ac.uk
```

You will be asked to enter your ARC password, and once authenticated the tunnel will be set up and you should be able to use the URL from the original Jupyter output on your local browser to connect to the server running on ARC.

Note: It is possible that when making the connection above you may receive a "port in use" error on your local machine. In which case you should change the first port number in the SSH command, and also change this in the web URL.

Closing down the Jupyter server

Open the window with the orginal connection to ARC (where you started Jupyter) and hit CTRL-C, when asked if you would like to `Shutdown this notebook server?` Answer `y-` as in the following example:

```
^C[I 16:33:40.585 NotebookApp] interrupted
Serving notebooks from local directory: /home/ouit0554
0 active kernels
Jupyter Notebook 6.4.5 is running at:
http://arc-c305:8888/?token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf
 or http://127.0.0.1:8888/?token=1f4df891d02d913a7b926203a978c0b3060113c1607527bf
Shutdown this notebook server (y/[n])? y
[C 16:33:42.014 NotebookApp] Shutdown confirmed
[I 16:33:42.016 NotebookApp] Shutting down 0 kernels
[I 16:33:42.016 NotebookApp] Shutting down 0 terminals
[ouit0554@arc-c305 ~]$ exit
srun: error: arc-c305: task 0: Exited with exit code 130
```

CHAPTER

SEVEN

USING R ON ARC

R is a powerful free software environment for statistical computing. It provides access to a wide variety of statistical methods (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc.) as well as to graphical techniques. R is highly extensible and a large number of additional libraries are available. A number of libraries are already installed on the ARC systems and more can be installed on request.

R has fairly frequent releases and we try to keep up to date. Check the versions available using `module spider R` To avoid incompatibilities, each new R installation uses its own installation of the supported libraries.

7.1 Loading an R Module

To use R on the ARC systems you simply need to load the latest R environment module, in this example version 4.1.2 - it can be loaded using the following command:

```
module load R/4.1.2-foss-2021b-ARC
```

To see all versions of R we have available, use the command

```
module spider R
```

The base install has many popular R packages installed. Additionally there are modules available with the `-ARC` suffix e.g. `R/4.1.2-foss-2021b-ARC` these modules load many more R libraries than the base R installation and contains those libraries most requested by ARC users. It is possible that you will need access to libraries which are not installed in the base or ARC R modules. You can install R libraries in an R library repository within your storage area (e.g. $HOME or $DATA) please see below.

Please note: Some R libraries depend on the existence of non-R applications or other shared binaries. Attempting to install an R library with binary dependencies may fail. In this case please contact the ARC team and we will install the dependencies for you centrally.

7.1.1 Typical R submission script

The following is an example of a typical R submission script for ARC:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --partition=short
#SBATCH --time=12:00:00
```

(continues on next page)

(continued from previous page)

```
#SBATCH --job-name=Rtest

module purge
module load  R/4.1.2-foss-2021b-ARC

Rscript --no-restore --no-save r_script.R
```

The script above requests a single node and 8 cores. You will need to ensure your R script is written using R parallel libraries in order to make use of these CPU resources.

7.2 Installing packages into your own R library

As this is an interactive process which may involve building software, it needs to be performed on an interactive node, so first start an interactive session:

```
srun -p interactive --pty /bin/bash
```

In order to use your own R library repository, you need to define an environment variable named `R_LIBS` containing the path to your local packages (this will need to be available each time you intend to use your local library, so you may wish to place it in your `$HOME/.bash_profile` file)

```
export R_LIBS=~/local/rlibs
```

Please note: If you do not place the above line in your `$HOME/.bash_profile` file, you will need to ensure you include it in your submission scripts in order for R to find your locally installed libraries.

You can then create this folder (Note: this only needs to be done once):

```
mkdir -p ~/local/rlibs
```

Once this is done you can start R and run the `install.packages` command to install into this local library repository - or indeed follow the instructions given for a particular package. As an example to install the latest `devtools` package:

```
[user@arc-c001]$ R

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

(continues on next page)

(continued from previous page)

```
> install.packages("devtools", lib="~/local/rlibs")
```

For packages such as `BiocManager` you will need to add the lib location to both the `BiocManager` installation and subsequent `BiocManager::install` commands for example:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager", lib="~/local/rlibs")

BiocManager::install("dada2", version = "3.11", lib="~/local/rlibs")
```

You may find you need to use the `http` URL protocol rather than `https` for some repositories.

Important note: As some R libraries require compilation in order to be installed, it is worth noting that the interactive nodes share the same CPU architecture as the majority of ARC/HTC compute nodes but not the login nodes. This allows users to optimise their compiled code to make use of the most recent CPU features. However, if you attempt to load a library built on the interactive nodes from the login nodes or older compute nodes you may see execution errors such as "Illegal instruction" or "Illegal Operand" - these errors are simply warning you that the CPU cannot understand the more recent instructions the compiler has generated.

To mitigate the above issue:

- If you want to test the library interactively, please ensure you use an interactive node - not a login node for this purpose.
- If you submit a batch job, ensure that you specify:

  ```
  #SBATCH --constraint='cpu_gen:Cascade_Lake'
  ```

The above will ensure your job runs on a node with the same architecture as the interactive node you used to build the library.

7.3 Using Rmpi on ARC

Example Rmpi script for SLURM

The following is a brief example of how to run an Rmpi script on SLURM. This example uses the latest R version R/4.1.2-foss-2021b

Create the following script and save it as `Rmpi-test.R`:

```
#
# Rmpi test
#
library("Rmpi")

# Spawn (size-1) slaves, one being resevered for master.
#
size <- mpi.universe.size()
mpi.spawn.Rslaves(nslaves = size - 1)

# Identify each slave process and display a message
#
mpi.bcast.cmd(rank <- mpi.comm.rank())
```

(continues on next page)

(continued from previous page)

```
mpi.bcast.cmd(size <- mpi.comm.size())
mpi.bcast.cmd(hostname <- mpi.get.processor.name())
mpi.remote.exec(paste("This is rank", rank, "of", size, "running on node: ", hostname))

# Close all slaves and finish
#
mpi.close.Rslaves(dellog = FALSE)
mpi.quit()
```

Now create a submission script called `submit.sh` containing the following lines:

```
#! /bin/bash

#SBATCH --job-name=mpi-test
#SBATCH --time=00:10:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --output=rmpi.out
#SBATCH --partition=devel

module load R/4.1.2-foss-2021b
mpirun -np 1 R --vanilla -f Rmpi-test.R
rm *.log
```

The script above requests at total of 16 processes, where 8 are run on each of 2 nodes. The job is sent to the `devel` partition (hence maximum 10 minute run time) and the job output is sent to a file named `rmpi.out`

To submit this job, use the command:

```
sbatch submit.sh
```

After a few minutes the job should complete and a file named `rmpi.out` should be created in the submission directory, this should contain the following output:

```
$ cat rmpi.out

R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

   Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

(continues on next page)

(continued from previous page)

```
> #
> # Rmpi test
> #
> library("Rmpi")
>
> # Spawn (size-1) slaves, one being resevered for master.
> #
> size <- mpi.universe.size()
> mpi.spawn.Rslaves(nslaves = size - 1)
        15 slaves are spawned successfully. 0 failed.
master  (rank 0 , comm 1) of size 16 is running on: arc-c266
slave1  (rank 1 , comm 1) of size 16 is running on: arc-c266
slave2  (rank 2 , comm 1) of size 16 is running on: arc-c266
slave3  (rank 3 , comm 1) of size 16 is running on: arc-c266
... ... ...
slave14 (rank 14, comm 1) of size 16 is running on: arc-c267
slave15 (rank 15, comm 1) of size 16 is running on: arc-c267
>
> # Identify each slave process and display a message
> #
> mpi.bcast.cmd(rank <- mpi.comm.rank())
> mpi.bcast.cmd(size <- mpi.comm.size())
> mpi.bcast.cmd(hostname <- mpi.get.processor.name())
> mpi.remote.exec(paste("This is rank", rank, "of", size, "running on node: ", hostname))
$slave1
[1] "This is rank 1 of 16 running on node:  arc-c266"
$slave2
[1] "This is rank 2 of 16 running on node:  arc-c266"
$slave3
[1] "This is rank 3 of 16 running on node:  arc-c266"
$slave4
[1] "This is rank 4 of 16 running on node:  arc-c266"
$slave5
[1] "This is rank 5 of 16 running on node:  arc-c266"
$slave6
[1] "This is rank 6 of 16 running on node:  arc-c266"
$slave7
[1] "This is rank 7 of 16 running on node:  arc-c266"
$slave8
[1] "This is rank 8 of 16 running on node:  arc-c267"
$slave9
[1] "This is rank 9 of 16 running on node:  arc-c267"
$slave10
[1] "This is rank 10 of 16 running on node:  arc-c267"
$slave11
[1] "This is rank 11 of 16 running on node:  arc-c267"
$slave12
[1] "This is rank 12 of 16 running on node:  arc-c267"
$slave13
[1] "This is rank 13 of 16 running on node:  arc-c267"
$slave14
```

(continues on next page)

(continued from previous page)

```
[1] "This is rank 14 of 16 running on node:  arc-c267"
$slave15
[1] "This is rank 15 of 16 running on node:  arc-c267"
>
> # Close all slaves and finish
> #
> mpi.close.Rslaves(dellog = FALSE)
[1] 1
> mpi.quit()
```

From the above output you can see the slave processes ran correctly with a total of 16 processes across two ARC compute nodes.

7.4 Using R-Studio on ARC

You can use the R-Studio IDE on ARC via an interactive `X11` session. This means you will need to connect to ARC via a client running an `X11` server such as MobaXterm for Windows, or XQuartz for Mac.

It is necessary to make your SSH connection to the ARC login node with the -X option in order to forward `X11` graphics back to your desktop machine - for example:

```
ssh -X username@arc-login.arc.ox.ac.uk
```

Note: Mac users may need to use -Y instead of -X if they get security warnings with -X.

See the **Connecting to ARC Clusters** section of the ARC User Guide for more information.

Once connected to a login node, you will need to connect to an interactive session on a compute node:

```
srun -p interactive --x11 --pty /bin/bash
```

The above line will create an interactive session and ensure that the `X11` forwarding is preserved.

To run R-Studio you will need to load the appropriate module:

```
module spider RStudio
```

The above command returns the available versions of R-Studio, for example:

```
-----------------------------------------------------------------------------------------------
 ↪-----------------------------------------------------
RStudio: RStudio/2022.02-R-4.2.1-ARC
-----------------------------------------------------------------------------------------------
 ↪-----------------------------------------------------
```

In the above case, loading the module `RStudio/2022.02-R.4.2.1-ARC` will load version 2022.2 of RStudio along with R version 4.2.1 with added ARC libraries.

So to begin using the application you will need to type:

```
module load RStudio/2022.02-R.4.2.1-ARC
rstudio
```

After a short delay, the R-Studio window should appear on your desktop display.

Note: Mac users may experience problems using the XQuartz X11 server with R-Studio, where the application starts then fails with OpenGL errors. In this case follow the guidance below…

Forcing R-Studio to use software rendering

Log in to arc-login and run the following commands:

```
mkdir -p $HOME/.config/RStudio
touch $HOME/.config/RStudio/desktop.ini
```

Edit the file $HOME/.config/RStudio/desktop.ini to contain the following text:

```
[General]
cookies=@Invalid()
desktop.renderingEngine=software
general.disableGpuDriverBugWorkarounds=true
general.ignoreGpuExclusionList=true
```

The above settings ensure that RStudio only uses software graphics rendering rather than OpenGL which seems to give problems with XQuartz and some other X11 servers.

CHAPTER

EIGHT

APPLICATION SOFTWARE GUIDES

The guides in this section provide information on running specific software applications in the ARC environment.

8.1 Abaqus

Introduction

Abaqus is a popular commercial finite element, multi-physics simulation from Simulia. Abaqus is a parallel application and can run in both MPI and in shared-memory threaded mode. The parallel execution is triggered and controlled by command line options to the abaqus driver, along with all the other run options.

The steps to run an Abaqus job are:

- load the Abaqus module;
- create a submission script;
- submit the job.

Note: The only supported version on ARC is the latest version **Abaqus 2022** This is due to the Intel MPI and compiler support provided by the vendor.

Module Information:

```
module spider Abaqus

----------------------------------------------------------------------------
Abaqus: Abaqus/2022
----------------------------------------------------------------------------
  Description:
    Finite Element Analysis software for modeling, visualization and best-in-class␣
↪implicit and explicit dynamics FEA.
```

Example Submission Script

The example submission scripts below are suitable for running on the ARC cluster:

```
#!/bin/bash

#SBATCH --clusters=arc
#SBATCH --partition=devel
#SBATCH --nodes=2
```

(continues on next page)

(continued from previous page)

```
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=AbaqusStandard

module purge
module load Abaqus/2022

. abaqus.sh

abaqus fetch job=s4b.inp
abaqus job=s4b  \
       input=s4b \
       cpus=${SLURM_NTASKS} \
       interactive
```

The above is an Abaqus/Standard job running in hybrid MPI mode. The following example runs an Abaqus/Explicit simulation:

```
#!/bin/bash

#SBATCH --clusters=arc
#SBATCH --partition=devel
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=AbaqusExplicit

module purge
module load Abaqus/2022

. abaqus.sh

abaqus fetch job=knee_bolster
abaqus fetch job=knee_bolster_ef1
abaqus fetch job=knee_bolster_ef2
abaqus fetch job=knee_bolster_ef3

abaqus job=knee_bolster  \
       input=knee_bolster \
       cpus=${SLURM_NTASKS} \
       interactive
```

Note: The line `. abaqus.sh` in the above scripts is **important**, it ensures that Abaqus is configured correctly for the ARC environment by creating a file named `abaqus_v6.env` in the job directory.

It also creates a scratch directory for temporary Abaqus files, and ensures the Intel MPI library is used.

If you wish to make use of user defined Fortran functions, you should add the line:

```
module load iccifort/2020.1.217
```

To your submission script after the Abaqus module load. The `abaqus.sh` sccript also sets up the environment/options

for the ifort compiler.

8.2 CASTEP

Introduction

CASTEP is an electronic structure materials modelling code based on density functional theory (DFT), with functionality including geometry optimization molecular dynamics, phonons, NMR chemical shifts and much more.

Warning: While the use of CASTEP is free for University researchers, you still require a licence to run it. Licensing information can be found here: http://www.castep.org/CASTEP/GettingCASTEP

Once you have a valid licence, you need to request access via this form: https://www.arc.ox.ac.uk/restricted-licence-software-applications to request access on ARC.

The guide shows how to:

- prepare a CASTEP job submission script and
- submit and run a CASTEP job.

Module information:

```
module use /apps/common/private/modules
module spider castep

    Versions:
       CASTEP/16.11-info
       CASTEP/16.11-intel-2020a
       CASTEP/21.1.1-foss-2019b
       CASTEP/21.1.1-intel-2021b
       CASTEP/21.11-info
```

Note: The modules with the suffix `-info` should not be loaded.

Running a CASTEP job

Here is an example of how to run the Crambin example (part of the CASTEP benchmarks http://www.castep.org/CASTEP/Crambin) on a cluster node.

An example submission script would look as follows - create a file named `run-castep.sh` containing:

```
#!/bin/bash

#SBATCH --clusters=arc
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=CASTEP
#SBATCH --partition=devel
```

(continues on next page)

(continued from previous page)

```
module purge

module use /apps/common/private/modules
module load CASTEP/21.1.1-intel-2021b


mpirun castep.mpi crambin
```

The script requires two nodes (48 cores per node) and launches CASTEP taking the input from the file `crambin.cell`

To launch into execution, issue the command:

```
sbatch run-castep.sh
```

8.3 ANSYS CFX

Introduction

ANSYS CFX's advanced physics modeling capabilities help engineers solve the most complex challenges in turbomachinery applications. This powerful software has been extensively validated and is renowned for its robustness and accuracy.

Module Information:

```
module spider CFX

----------------------------------------------------------------------------
  cfx: cfx/2021R2
----------------------------------------------------------------------------

    This module can be loaded directly: module load cfx/2021R2

   Help:
        Adds Ansys CFX 2021R2 to your PATH environment variable
```

If you need to use a version of CFX earlier that R2021R2 then you will have to use the "legacy" software partition. Infomation on legacy applications can be found here.

Example Submission Script

The example submission script below is suitable for running on the ARC cluster:

```
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=CFXExample
#SBATCH --partition=devel

module load cfx/2021R2
module load OpenMPI/4.0.5-GCC-10.2.0
```

(continues on next page)

(continued from previous page)

```
export CFX5_OPENMPI_DIR=$EBROOTOPENMPI

MPI_HOSTS=$(scontrol show hostnames $SLURM_JOB_NODELIST | tr "\n" "," | sed 's/.$//')

cfx5solve -def perf_Airfoil_50M_R16.def \
         -double -part-large \
          -part $SLURM_NTASKS -batch -parallel -par-host-list $MPI_HOSTS \
          -start-method "Open MPI Distributed Parallel"
```

Note: This example is customised for the standard ANSYS CFX benchmark Airfoil 50M and uses the OpenMPI libraries built for ARC.

8.4 ANSYS FLUENT

Introduction

ANSYS FLUENT software contains the broad physical modeling capabilities needed to model flow, turbulence, heat transfer, and reactions for industrial applications ranging from air flow over an aircraft wing to combustion in a furnace, from bubble columns to oil platforms, from blood flow to semiconductor manufacturing, and from clean room design to wastewater treatment plants.

Module Information:

```
module spider FLUENT

----------------------------------------------------------------------------
  fluent:
----------------------------------------------------------------------------
     Versions:
        fluent/2021R2
```

If you need to use a version of FLUENT earlier that R2021R2 then you will have to use the "legacy" software partition. Infomation on legacy applications can be found here.

Example Submission Script

The example submission script below is suitable for running on the ARC cluster. This example is customised to ensure FLUENT uses the ARC supplied OpenMPI libraries instead of those supplied by ANSYS:

```
#!/bin/bash

#SBATCH --partition=devel
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=FluentTest

module purge
module load fluent/2021R2
module load OpenMPI/4.0.5-GCC-10.2.0
export OPENMPI_ROOT=$EBROOTOPENMPI
```

(continues on next page)

(continued from previous page)

```
NODELIST="${SLURM_SUBMIT_DIR}/hostlist.${SLURM_JOB_ID}.txt"
scontrol show hostname ${SLURM_NODELIST} >${NODELIST}

fluent 2d -g -slurm -t${SLURM_NPROCS} -pinfiniband -mpi=openmpi -cnf=${NODELIST} -i␣
↪journal.inp

rm ${NODELIST}
```

8.5 Gaussian

Introduction

Gaussian is a commercial computational chemistry software for modelling electronic structures. Gaussian provides state-of-the-art capabilities for electronic structure modeling. Gaussian 16 is licensed for a wide variety of computer systems. All versions of Gaussian 16 contain every scientific/modeling feature, and none imposes any artificial limitations on calculations other than your computing resources and patience.

The guide shows how to:

- prepare a Gaussian job submission script and
- submit and run a Gaussian job.

Module information:

```
module spider gaussian

    Versions:
       Gaussian/03.E.01-ARCUS-B
       Gaussian/09.D.01-ARCUS-B
       Gaussian/16.A.03-ARCUS-B

       Gaussian/16.C.01
```

Note: The modules with the suffix `-ARCUS-B` are the built from the legacy ARCUS-B system, using PGI 16.5 compiler and Atlas. The other modules are built natively on ARC.

While the use of Gaussian is free for University researchers, the ARC team has to enable users to run Gaussian on an individual basis; if you are planning to run Gaussian, you need to request access via this form: https://www.arc.ox.ac.uk/restricted-licence-software-applications otherwise you will see the message permission denied when attempting to run Gaussian commands.

Gaussian on the ARC systems

Gaussian 03 , Gaussian 09 and Gaussian 16 are available on all the ARC systems and were built from source. They can be loaded using one of the following commands:

```
module load Gaussian/16.A.03-ARCUS-B
module load Gaussian/09.D.01-ARCUS_B
module load Gaussian/03.E.01-ARCUS-B
module load Gaussian/16.C.01
```

Gaussian is a multi-threaded application and users are advised to take advantage of this feature as it leads to a faster execution. Running Gaussian multi-threaded is controlled from the header of the Gaussian com input files.

Running a Gaussian job

Here is an example of how to run the valinomycin test (part of the Gaussian distribution) on a cluster node.

First, load the Gaussian 09 module:

```
module load Gaussian/16.A.03-ARCUS-B
```

Then, edit the file test397.com to set the number of threads. This should be set to match the availability of cores per compute node, which in the case of ARC is 48.

%NProcShared=48

An example submission script would look as follows - create a file named `run-g16.sh` containing:

```
#! /bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --time=01:00:00
#SBATCH --job-name=testGaussian
#SBATCH --partition=short

module load Gaussian/16.A.03-ARCUS-B

g16 < test397.com > test397.log
```

The script requires one node, launches Gaussian 16 taking the input from the file `test397.com` and redirects the standard output to the file `test397.log`.

To launch into execution, issue the command:

```
sbatch run-g16.sh
```

Gaussview

The Gaussview software is also installed on the ARC clusters for convenience. You should use one the interactive nodes to run Gaussview. However, we recommend that the desktop version of Gaussview is used to prepare input for Gaussian, as running graphical applications across the network is less efficient. The Department of Chemistry has a site licence for Gaussview and can be contacted for details at the following email address: help@itsupport.chem.ox.ac.uk

8.6 GROMACS

Introduction

GROMACS is a versatile package for molecular dynamics simulations, which solves the Newtonian equations of motion for systems with hundreds to millions of particles. GROMACS has been primarily designed for biochemical molecules (such as proteins, lipids and nucleic acids) but has also been used for research on non-biological systems, e.g. polymers. This is a CPU only build, containing both MPI and threadMPI builds.

CPU Version:

```
GROMACS/2020.4-foss-2020a
```

GPU versions:

```
GROMACS/2020-fosscuda-2019b
GROMACS/2020.4-foss-2020a-PLUMED-2.6.2
```

The guide shows how to

- prepare a GROMACS job submission script and
- submit the GROMACS job.

GROMACS on ARC clusters

Several versions of GROMACS are installed on the ARC clusters, mostly with only single precision support. GROMACS is managed through the `module` utility; use `module spider gromacs` to see the modules available, and also use `module` to load the appropriate GROMACS version.

Running a GROMACS job on multi node cluster

Here is an example of a submission script GROMACS job on ARC:

```
#!/bin/bash

#SBATCH --nodes=4
#SBATCH --ntasks-per-node=48
#SBATCH --time=08:00:00
#SBATCH --partition=short
#SBATCH --clusters=arc

module purge
module load GROMACS/2020.4-foss-2020a

mpirun gmx_mpi mdrun -s nsteps800.tpr -deffnm nc2-cubic-md -ntomp 1 -dlb yes -noconfout -
↪npme 64
```

Also, here is an example of using GROMACS with GPU support on the HTC cluster:

```
#!/bin/bash

#SBATCH --cluster=htc
#SBATCH --partition=medium
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=5
#SBATCH --mem=377000
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00

module purge
module load GROMACS/2020.4-fosscuda-2019b

mpirun gmx_mpi mdrun -s nsteps800.tpr -deffnm nc2-cubic-md -ntomp 1 -dlb yes -noconfout -
↪npme 64
```

8.7 HemeLB

Introduction

HemeLB is a high performance lattice-Boltzmann solver optimized for simulating blood flow through sparse geometries, such as those found in the human vasculature. It is routinely deployed on powerful supercomputers, scaling to hundreds of thousands of cores even for complex geometries . HemeLB has traditionally been used to model cerebral bloodflow and vascular remodelling in retinas , but is now being applied to simulating the fully coupled human arterial and venous trees.

The guide shows how to

- run HemeLB using a pre-compiled module;
- build a version of HemeLB;
- submit a test HemeLB job.

To use HemeLB you can either use the pre-installed modules or build your own verion in your project `$DATA` area.

Using the pre-installed versions of HemeLB:

```
module spider HemeLB

     HemeLB/0.8
```

And load the appropriate version. For example:

```
module load HemeLB/0.8
```

The above will load HemeLB into your environment, and the package will be available to run via `mpirun`.

Building your own version of HemeLB from source

First ensure you are using an interactive session on a compute node:

```
srun -p interactive --time=02:00:00 --pty /bin/bash
```

We have a recipe which you can use to build HemeLB (this has been tested with version 0.8):

```
cd $DATA

export HEME_HOME=$DATA/HemeLB/0.8

module load foss/2020a
module load CMake/3.16.4-GCCcore-9.3.0
module load ParMETIS/4.0.3-gompi-2020a
module load Boost/1.72.0-gompi-2020a
module load Python/2.7.18-GCCcore-9.3.0

mkdir -p $HEME_HOME/bin
mkdir -p $HEME_HOME/lib

wget http://hemelb.org.s3-website.eu-west-2.amazonaws.com/files/hemelb-v0.8.tar.gz

tar xvf hemelb-v0.8.tar.gz
cd hemelb-v0.8
```

(continues on next page)

(continued from previous page)

```
rm -rf CMakeCache.txt

mkdir bootstrap_build && cd bootstrap_build
export CXXFLAGS="-I/usr/include/tirpc"
export LDFLAGS="-ltirpc"

cmake -DCMAKE_INSTALL_PREFIX=$HEME_HOME ..
make -j8

cd ../dependencies/lib
cp -r * $HEME_HOME/lib/
```

To use the executable you will need to set the following environment variables:

```
export HEME_HOME=$DATA/HemeLB/0.8
export PATH=$HEME_HOME/bin:$PATH
export LD_LIBRARY_PATH=$HEME_HOME:$LD_LIBRARY_PATH
```

Example submission script

This example uses the input data found in HemeLB's `examples/bifurcation/bifurcation_hires` directory, this is a version using the ARC module:

```
#! /bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --partition=devel
#SBATCH --time=00:10:00

module load HemeLB/0.8

mpirun hemelb -in  input.xml -out output
```

The following is a version for a locally built version:

```
#! /bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --partition=devel
#SBATCH --time=00:10:00

module load foss/2020a
module load ParMETIS/4.0.3-gompi-2020a
module load Boost/1.72.0-gompi-2020a
module load Python/2.7.18-GCCcore-9.3.0

export HEME_HOME=$DATA/HemeLB/0.8
export PATH=$HEME_HOME/bin:$PATH
export LD_LIBRARY_PATH=$HEME_HOME:$LD_LIBRARY_PATH

mpirun hemelb -in  input.xml -out output
```

Note: You need to explicitly include the modules that HemeLB requires in this script (the ARC build module loads these automatically)

8.8 Java

Introduction

Java is one of the most popular programming languages, particularly for client-server web applications, often preferred over other options for faster code development, higher code reliability and easy portability. Although not often used in scientific programming, Java is nevertheless the language in which some applications are written (e.g. Beast, a Bayesian MCMC analysis application for molecular sequences) or in which researchers choose to develop their own codes.

Module information:

```
module spider java

----------------------------------------------------------------------------
Java:
----------------------------------------------------------------------------
    Description:
      Java Platform, Standard Edition (Java SE) lets you develop and deploy
      Java applications on desktops and servers.

     Versions:
        Java/1.7.0_60
        Java/1.8.0_131
        Java/1.8.0_241
        Java/11.0.2
```

Multi-threaded Java applications

Java applications can benefit from faster execution on modern multi-core machines through using multiple execution threads, and Java provides built-in support for multithreaded programming. A multithreaded program contains two or more threads (each defining a separate path of execution) that can run concurrently. Multi-threaded programming in Java is beyond the scope of this guide and users are invited to start with one of the numerous web tutorials available on the matter, e.g. http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

Example submission script:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=threadedTest

module purge

# Use latest Java version
module load Java
```

(continues on next page)

(continued from previous page)

```
myApplication -threads ${SLURM_NTASKS} -input myAppInput.txt -output myAppOutput.txt
```

In the above example, `myApplication` `myAppinput` and `myAppOutput` should be substituted for the appropriate names of your Java application and data files.

8.9 Julia

Introduction

Julia is a high-level, high-performance dynamic programming language for numerical computing

Module information:

```
module spider julia

-----------------------------------------------------------------------------
Julia:
-----------------------------------------------------------------------------
    Description:
    Julia is a high-level, high-performance dynamic programming language for numerical␣
↪computing

   Versions:
      Julia/1.5.1-linux-x86_64
      Julia/1.5.3-linux-x86_64
      Julia/1.6.2-linux-x86_64
```

Multi-threaded Julia applications

Julia applications can benefit from faster execution on modern multi-core machines through using multiple execution threads, and Julia provides built-in support for multithreaded programming. A multithreaded program contains two or more threads (each defining a separate path of execution) that can run concurrently.

Example submission script:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --time=12:00:00
#SBATCH --job-name=JuliaTest
#SBATCH --partition=short

module purge
module load Julia/1.6.2-linux-x86_64
julia -p ${SLURM_TASKS_PER_NODE} -t ${SLURM_CPUS_PER_TASK} myInput.jl
```

In the above example, `myInput.jl` should be substituted for the appropriate names of your Java application and data files. The values of `-p` (number of worker processes) and `-t` (number of threads), are derived from the the values of `ntasks-per-node` and `cpus-per-task` in the submission script.

8.10 LAMMPS

Introduction

LAMMPS is a classical molecular dynamics code, and an acronymfor 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 atthe atomic, meso, or continuum scale. LAMMPS runs on single processors or in parallel using message-passing techniques and a spatial-decomposition of thesimulation domain. The code is designed to be easy to modify or extend with new functionality.

The guide shows how to

- prepare a LAMMPS with a Python virtual environment and
- submit a test LAMMPS job.

To use LAMMPS you can either use the pre-installed modules or build your own Python virtual environment.

Using the pre-installed versions of LAMMPS:

```
module spider LAMMPS

     LAMMPS/3Mar2020-foss-2020a-Python-3.8.2-kokkos-QUIP
     LAMMPS/3Mar2020-foss-2020a-Python-3.8.2-kokkos
     LAMMPS/23Jun2022-foss-2021b-kokkos-CUDA-11.4.1
     LAMMPS/23Jun2022-foss-2021b-kokkos
```

And load the appropriate version. For example:

```
module load LAMMPS/3Mar2020-foss-2020a-Python-3.8.2-kokkos
```

The above will load LAMMPS into your environment, and the package will be available from within Python.

Setting up your own virtual environment (Python 3)

First ensure you are using an interactive session on a compute node:

```
srun -p interactive --time=02:00:00 --pty /bin/bash
```

We have a set of build scripts which you can use to build LAMMPS with most of the packages, and its own Anaconda 3 environment. To obtain a copy of these scripts:

```
cd $DATA
mkdir scripts ; cd scripts
cp -r /apps/common/examples/LAMMPS/2022.06/* .
```

By default our build script will create the following directories:

```
$DATA/LAMMPS-env    (Python environment and installation location)
$DATA/LAMMPS-build  (Build files)
```

If you would like to use different directories, you will need to change the `$DATA/scripts/LAMMPS-config.sh` file. Specifically the following two lines:

```
export LAMMPS_INSTALL=$DATA/LAMMPS-env
export LAMMPS_BUILD=$DATA/LAMMPS-build
```

To run the build of LAMMPS:

```
cd $DATA/scripts
sh LAMMPS-build.sh
```

Once the build completes you can test the LAMMPS build by submitting the test job in the `scripts` folder as follows:

```
cd $DATA/scripts
sbatch LAMMPS-test.sh
```

The output in the generated SLURM file `slurm-nnnnnn.out` should include the following information:

```
Testing Python interface...(quick)
LAMMPS (23 Jun 2022)
  using 1 OpenMP thread(s) per MPI task
Total wall time: 0:00:00
```

Followed by:

```
Testing dipole example run...
LAMMPS (23 Jun 2022)
   using 1 OpenMP thread(s) per MPI task
WARNING: Atom style hybrid defines both, per-type and per-atom masses; both must be set,␣
↪but only per-atom masses will be used (src/atom_vec_hybrid.cpp:133)

...
...
...

Lattice spacing in x,y,z = 1.6903085 1.6903085 1.6903085
Created orthogonal box = (0 0 -0.84515425) to (16.903085 16.903085 0.84515425)
6 by 8 by 1 MPI processor grid

Per MPI rank memory allocation (min/avg/max) = 5.055 | 5.055 | 5.055 Mbytes
   Step          Temp          E_pair         c_erot         TotEng         Press
         0   0             -2.1909822      0             -2.1909822     -2.5750971
...
...
...

Performance: 23142333.912 tau/day, 53570.217 timesteps/s
99.3% CPU use with 48 MPI tasks x 1 OpenMP threads

...
...
...

Total # of neighbors = 1650
Ave neighs/atom = 8.25
Neighbor list builds = 684
Dangerous builds = 0
Total wall time: 0:00:00
```

Note: If you need to change the packages installed you can edit the build script `$DATA/scripts/LAMMPS-build.sh`

and re-run it.

8.11 Mathematica

Introduction

Mathematica is a computational software program used in many scientific, engineering, mathematical and computing fields.

Mathematica is installed on the ARC clusters and is covered by the Oxford University Concurrent license.

The purpose of this example-based tutorial is to guide the user through a few aspects of running efficient Mathematica jobs on the ARC resources. These aspects are:

- using the Mathematica module,
- running Mathematica scripts non-interactively from batch jobs

Module Information:

```
module spider mathematica

----------------------------------------------------------------------------------------
 Mathematica:
----------------------------------------------------------------------------------------
    Versions:
       Mathematica/11.3.0
       Mathematica/12.2.0
       Mathematica/13.0.0
```

Mathematica Batch Submission

The following is an example submission script which runs a Mathematica input file named `test.m`:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --time=01:00:00
#SBATCH --job-name=MathTest
#SBATCH --partition=short

module purge
module load Mathematica/13.0.0

math -noprompt -run '<<test.m'
```

Assuming this script is named `run_math.sh` submit this to the scheduler:

```
sbatch run_math.sh
```

8.12 MATLAB

Introduction

MATLAB is a numerical computing and programming environment with a broad range of functionality (matrix manipulation, numerical linear algebra, general-purpose graphics, etc.). Additionally, specialised application areas (e.g. bioinformatics or financial derivatives) are served by a large number of optional toolboxes

Matlab is installed on the ARC clusters along with all the toolboxes covered by the Oxford University Concurrent license.

The purpose of this example-based tutorial is to guide the user through a few aspects of running efficient Matlab jobs on the ARC resources. These aspects are

- using the Matlab module,
- running Matlab scripts non-interactively from batch jobs and
- running Matlab jobs on parallel hardware:
- overview of parallel computing in Matlab,
- using the Matlab Parallel Computing toolbox,
- exploiting trivial task parallelism and
- multi-threaded Mex programming.

Module Information:

```
module spider matlab

----------------------------------------------------------------------------------------------------
↪------
 MATLAB:
----------------------------------------------------------------------------------------------------
↪------
    Versions:
       MATLAB/R2019b
       MATLAB/R2020a
       MATLAB/R2020b
       MATLAB/R2021b
       MATLAB/R2022a
```

Non-interactive Matlab Sessions

During code development on standard machines Matlab is usually run in interactive mode, in this way making full use of its integrated environment. By contrast, given the "batch processing" nature of supercomputing resources, the preferred mode of operation for Matlab on our systems is non-interactive.

One way to run Matlab non-interactively is through

- re-directing the standard input and output when invoking Matlab and
- invoking Matlab from a submission script, submitted to the queue via the slurm scheduler.

Input and output re-direction is arguably the easiest way of running Matlab non-interactivelly. It is achieved using the Linux operators < and >, with Matlab taking a code file as an input and writing the output to a file, e.g. `matlab < myScript.m > myOutput.txt` The main function/program e.g. `myScript.m` should have the `exit` command at the end in order to force Matlab to quit after finishing the execution of the code.

A simple example illustrating non-interactive Matlab use is found in the /apps/common/examples/matlab/seq directory which you can copy to your own area as follows:

```
cp -r /apps/common/examples/matlab $DATA/
cd $DATA/matlab/seq
```

In this example, the MATLAB program main.m sets a linear system with the right-hand side read from a file provided, solves it and saves the result to another file. A Matlab job is sent to the queue and executed on a backend node using the job scheduler.

Submission scripts should contain the following line to run the Matlab script:

```
matlab -nodisplay -nosplash < main.m > run.log
```

The flag -nodisplay instructs Matlab to run without the GUI, while -nosplash prevents the display of the Matlab logo. The < redirection operator ensures that Matlab runs your Matlab script, in this case main.m while the > operator re-directs the standard output (normally to the terminal) to run.log file.

For example for the seq example above, the contents of the SLURM submission script run_slurm.sh is:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --time=00:05:00
#SBATCH --job-name=matlab_test
#SBATCH --partition=devel

module purge
module load MATLAB

matlab -nodisplay -nosplash < main.m > run.log
```

To submit this to the scheduler:

```
sbatch run_slurm.sh
```

Overview of parallel computing in Matlab

Matlab was developed for a long time as a product for single-processor computing, partly because distributed parallelism was incompatible with Matlab's original design principles and partly because the potential market was perceived as too small to justify major development efforts. However, the advent of multi-core CPUs and the changing nature of Matlab, from the original educational "matrix laboratory" to a complex technical computing environment, prompted a revision of this situation. Matlab now benefits from running on modern parallel hardware in at least two ways.

The first is a built-in feature of Matlab, which "naturally" exploits multi-core processing via the underlying multi-threaded libraries Intel MKL and FFTW. Thus, linear algebra operations (such as the solution to a linear system Ab or matrix products A*B) and FFT operations (using the function fft) are implicitly multi-threaded and make use of all the cores available on a multi-core system without user intervention or special extra programming. Some of the vectorised operations in Matlab are also multi-threaded. However, this type of operations are only a part of Matlab programming and the vast proportion of the Matlab functionality are scripts or functions that can only use a single core.

Second, users can exploit parallel processing through a series of explicit programming techniques. The following techniques are briefly discussed below, with examples given:

- using the Matlab toolbox Parallel Computing Toolbox;
- trivial parallelism exploited through independent Matlab processes;
- multi-threaded MEX programming.

- Matlab has two toolboxes that enable explicit parallel programming: the **Parallel Computing Toolbox** and the **Parallel Server**. The **Parallel Computing Toolbox** is designed for programming multi-core architectures, while the **Parallel Server** extends the Matlab's functionality to large resources, such as clusters.

The functionality of the Parallel Computing Toolbox is extended from single cluster node processing to distributed processing across multiple nodes by the Parallel Server.

The University has licenses for the Parallel Computing Toolbox and users are encouraged to use it in their jobs run on the ARC hardware.

Using the Parallel Computing Toolbox

The Parallel Computing Toolbox offers the programmer a range of high-level parallelism contructs such as parfor (parallel for loops) and distributed arrays, which can be used to parallelise processing. Matlab scripts enhanced using these contructs can be run on a single multi-core system (such as a node of the ARC clusters), utilising all the cores available for parallel processing. While this offers scope for faster execution, the programming is not without catches and requires both programming experience and the understanding of the underlying algorithms. The MathWorks pages are the best introduction to the product.

A simple example of multi-core parallelism via the Parallel Computing Toolbox is provided in the `/apps/common/examples/matlab/par` directory which you can copy to your own area as follows:

```
cp -r /apps/common/examples/matlab $DATA/
cd $DATA/matlab/par
```

The program main.m evaluates an expensive function within a for loop and stores the results in an array. The for loop is parallelised using the parfor construct; a parfor loop behaves like an ordinary for loop on a single-core execution but shares the computational load between several workers (normally, each run on a separate core) in parallel execution. To make workers available for parallel execution, the command matlabpool is used in main.m; the example illustrates the behaviour of parfor both before and after the workers are initiated.

The example is run in batch mode with the command `sbatch run_slurm.sh` The submission file is:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --time=00:05:00
#SBATCH --job-name=matlab_test
#SBATCH --partition=devel


module purge
module load MATLAB

matlab -nodisplay -nosplash < main.m > run.log
```

Notice once again how MATLAB is instructed to not load the interactive window. The `ntasks-per-node` SLURM resource value is set to 16 to request 16 cores for this job.

Note: do not turn java off when lauching MATLAB (i.e. do not invoke `matlab -nojvm`); matlabpool uses the Java Virtual Machine.

After the job finishes, the CPU times spent executed the loops in `main.m` can be found in `timings.dat` showing a clear speed-up of the execution in parallel.

Exploiting trivial parallelism

An easy way to exploit multi-core systems is to split the workflow into parts that can be processed completely independently. The typical example in this category is a parameter sweep, where the same Matlab script is run a large number of times using different inputs; these runs are indepent from each other and can be carried out concurrently. Thus, the entire workflow can be scheduled in jobs that group 8 independent runs to match the 8 cores available per compute node. This strategy is best coupled with the use of the Matlab mcc compiler in order to avoid an excessive use of licenses.

A simple example is found in the the `/apps/common/examples/matlab/mcc` directory which you can copy to your own area as follows:

```
cp -r /apps/common/examples/matlab $DATA/
cd $DATA/matlab/mcc
```

The file `oscillator.m` is a Matlab script that computes the solution of a damped oscillator of unit mass (using the Matlab ode45 solver) and outputs the maximum oscillation in that solution. The script is prepared for use as a standalone deployed executable using the Matlab function isdeployed.

To deploy the script as a standalone application, start an interactive compute session, load the modules for Matlab and for the Intel-compilers, e.g.:

```
srun -p interactive --pty /bin/bash
module load MATLAB/R2021b intel/2020a
```

Then, compile the script using mcc and the command:

```
cd $DATA/matlab/mcc
mcc -v \
  -R -nojvm \
  -R -singleCompThread \
  -f ./mbuildopts.sh \
  -m oscillator.m
```

This command makes use of the options in the file `mbuildopts.sh` provided alongside the Matlab script and customised for the Intel compilers. If no option file is passed through the option -f, mcc uses the default options file, which uses the Gnu compilers gcc and g++; in principle, using the Intel compilers can lead to a faster executable.

The deployed executable is compiled to run using a single thread via the option -singleCompThread. This is important as a number of process are to run concurrently on the same multi-core system.

The mcc compilation creates an executable called oscillator. In addition to this, the process generates the files mccExcludedFiles.log and readme.txt, which can be safely discarded. Also, the wrapper script `run_oscillator.sh` is generated; this can be used to launch the executable oscillator into execution as it ensures the correct environment (paths to shared libraries and other environment variables) is set before execution. The ARC Matlab module updates all the necessary variables, and the executable oscillator can be launched directly, so using `run_oscillator.sh` is unecessary.

The submission script `run_slurm.sh` gives an example of how the deployed executable can be used to launch concurrent processes within the same job. On the clusters, the script requests a single compute node `#SBATCH --nodes=1` `#SBATCH --ntasks-per-node=8` so that 8 cores are available for processing. 8 separate processes are started with different parameters, such that the 8 processes compute a parameter sweep. The contents of `run_slurm.sh` is as follows:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
```

(continues on next page)

(continued from previous page)

```
#SBATCH --partition=devel
#SBATCH --time=00:10:00
#SBATCH --job-name=oscillator

module purge
module load MATLAB/R2021b intel/2020a


# start 8 processes in the background
./oscillator 0.01 0.3 > result1 &
./oscillator 0.02 0.3 > result2 &
./oscillator 0.03 0.3 > result3 &
./oscillator 0.04 0.3 > result4 &
./oscillator 0.05 0.3 > result5 &
./oscillator 0.06 0.3 > result6 &
./oscillator 0.07 0.3 > result7 &
./oscillator 0.08 0.3 > result8 &

# wait for all processes to finish (this is important!)
wait
```

With the run parameters (representing the damping coefficient and the stiffness) passed on as command line arguments. The processes are started in the background (using the symbol &), such that the second process can start before the first finishes, and so on. At the end of the script, a synchronisation point is necessary, which is implemented using a "wait" loop which "listens" for any processes called oscillator; without this synchronisation, the job launches the `oscillator` processes into background execution and finishes, without waiting for the processes to complete.

Each process prints the result (maximum oscillation) to the standard output; there is now way to "return" a numeric result from a standalone executable. It is easy to preserve the results after the job runs by redirecting the output to the files result*.

Finally, the use of `mcc` can be avoided altogether and Matlab can be run directly. For example, the first processing line in the script could be:

```
matlab -nojvm -singleCompThread -r "oscillator(0.01, 0.3); exit" > result1 &
```

However, deployed executables do not require Matlab licenses to run, which can make an important economy, especially in the case of a large number of concurrent processes (such as a parameter sweep).

8.13 MATLAB Parallel Server

MATLAB Parallel Server (MPS) allows you to create and use parallel pools that can scale to many CPUs more than in a single node. Currently MPS jobs must be submitted from a GUI or command line MATLAB session running on one of the ARC Graphical (NX) nodes or a session on an interactive node. You **must not** run MPS from a cluster login node.

Starting MATLAB from an NX node

If you are using the ARC Graphical (NX) nodes to run MATLAB. You need to ensure you select which cluster you wish to use, HTC or ARC.

Open the `Konsole` and load both the MATLAB and the cluster definition module:

```
module load MATLAB/R2022b cluster/arc

or

module load MATLAB/R2022b cluster/htc
```

You can then type `matlab` at the command line to start MATLAB.

Importing cluster profiles

We have generated some basic cluster profile definition files for each cluster ARC and HTC, these act as a starting point for your own profiles which can support custom resource requirements.

You need to import the ARC or HTC cluster profile into your MATLAB environment and set it as the default before you can submit MPS jobs.

This only needs doing once. The imported profile will be saved in your MATLAB settings directory.

Importing the profile can be done either by calling MATLAB functions or via the graphical interface. The profiles are stored here (for R2022b):

```
/apps/common/commercial/MATLAB/mps_profiles/R2022b
```

The profiles are configured for specific partitions or resources, as follows:

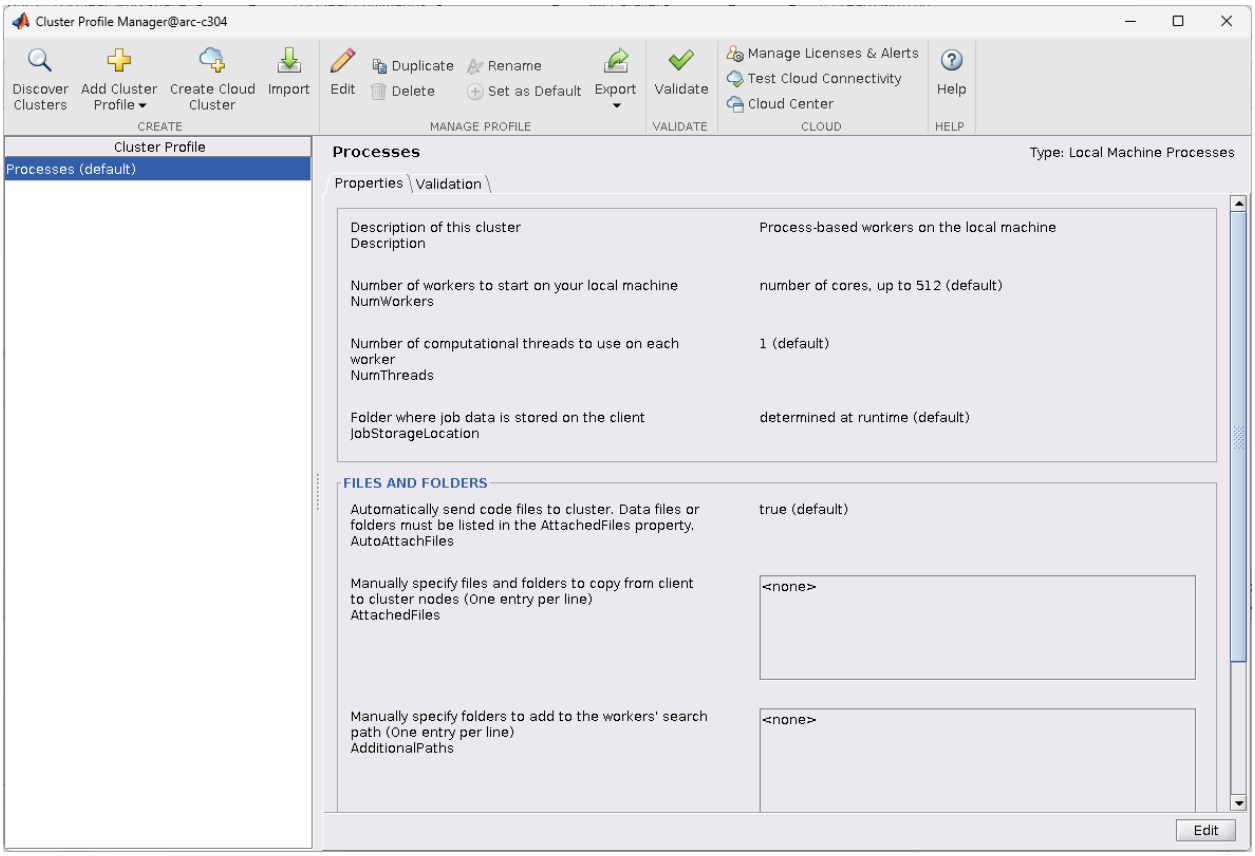
Partition/Resource	ARC	HTC
devel	arc_devel.mlsettings	htc_devel.mlsettings
short	arc_short.mlsettings	htc_short.mlsettings
medium	arc_medium.mlsettings	htc_medium.mlsettings

Importing cluster profiles directly from the MATLAB GUI

For these examples we will use the `arc_devel` profile.

To import using the graphical interface:

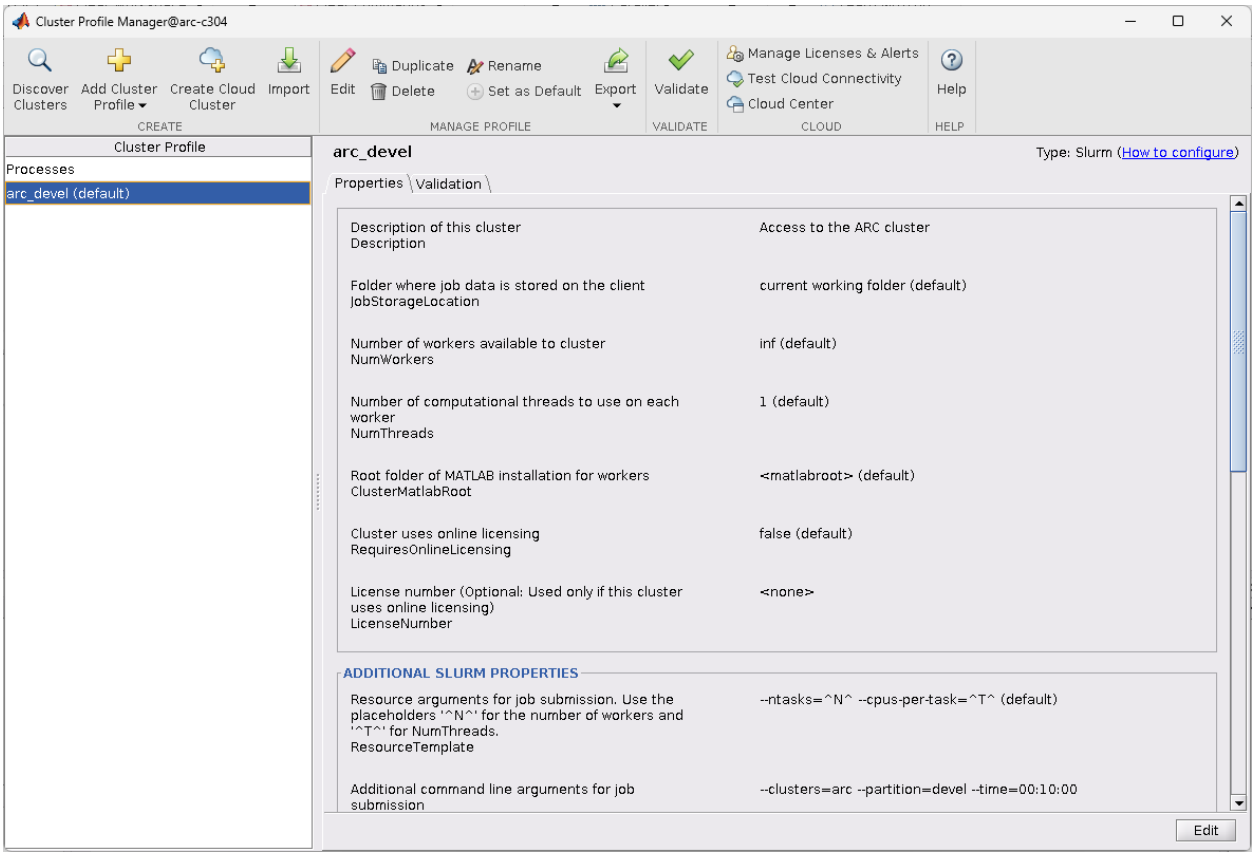
From the MATLAB Home tab select the `Parallel` menu and click `Create and Manage Clusters...` The **Cluster Profile Manager** window will open:



Select `Import` and from within the `Import Profiles from file` window navigate to the `arc_devel.mlsettings` file described above and select `Open`

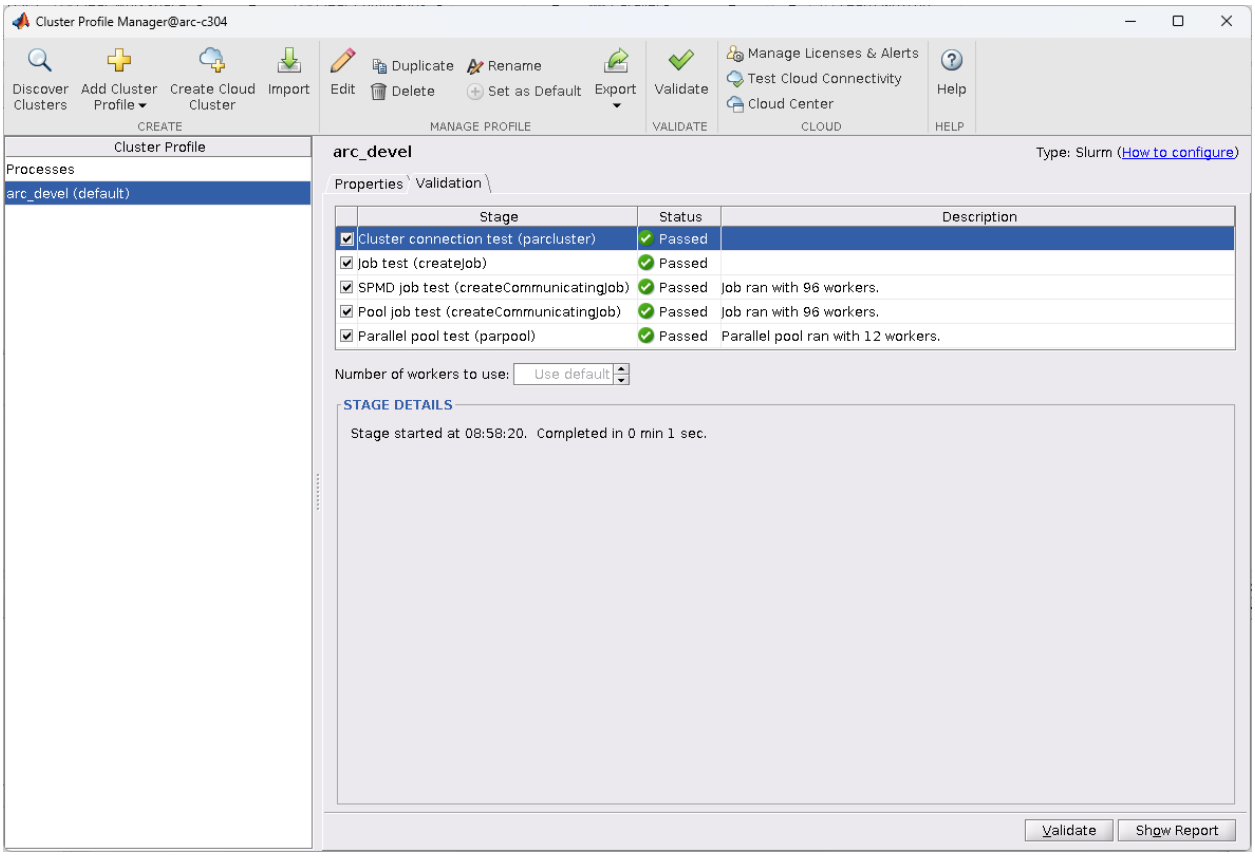
Select the resulting `arc_devel` profile and click `Set as Default`.

The Cluster Profile Manager window should now look like this:



Note: After you exit MATLAB, your default cluster profile is saved for future use.

To test the loaded profile, click the `Validate` button and the validation process will begin. Once the process completes the screen should look as follows:



Import using MATLAB functions

Instead of selecting and activating the cluster profile from the GUI you can run these functions from a command line MATLAB session:

```
arc_profile = parallel.importProfile ('/apps/common/commercial/MATLAB/mps_profiles/
↪R2022b/arc_devel.mlsettings');
parallel.defaultClusterProfile ('arc_devel');
```

The above could be used as part of a MATLAB script. See next example…

Full MPS Example

In this example we are going to use the `arc_short` partition. So, follow the instructions above to load the file named `arc_short.mlsettings`

Once this is loaded. From the MATLAB GUI click `New Script` and paste the following script into the editor:

```
%
% ARC Parallel MATLAB example
%
% Either have arc-short cluster profile set as default in GUI or uncomment the
% following two lines:
%
%parprof = parallel.importProfile("/apps/common/commercial/MATLAB/mps_profiles/R2022b/
↪arc_short.mlsettings")
%parallel.defaultProfile(parprof)
```

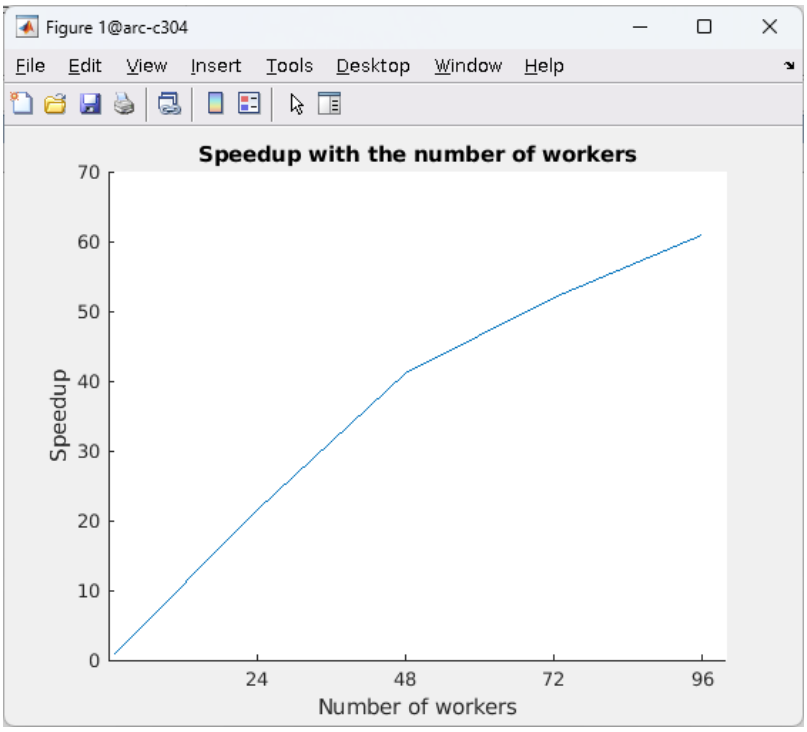
(continues on next page)

(continued from previous page)

```
primeNumbers = primes(uint64(2^20));
compositeNumbers = primeNumbers.*primeNumbers(randperm(numel(primeNumbers)));
factors = zeros(numel(primeNumbers),2);

% Create parallel pool
%
poolObj=parpool('arc_short',96);

numWorkers = [1 24 48 72 96];
tCluster = zeros(size(numWorkers));

for w = 1:numel(numWorkers)
    tic;
    parfor (idx = 1:numel(compositeNumbers), numWorkers(w))
        factors(idx,:) = factor(compositeNumbers(idx));
    end
    tCluster(w) = toc;
end

% Shutdown parallel pool.
delete(poolObj);

f = figure;
figure(f);
hold on
speedup = tCluster(1)./tCluster;
plot(numWorkers, speedup);
title('Speedup with the number of workers');
xlabel('Number of workers');
xticks(numWorkers(2:end));
ylabel('Speedup');
```

Once you have done this, click Run from the Editor. The script will start a job in the ARC short queue, and use this as a MATLAB worker pool.

If the job is able to run immediately the run time is around 5 minutes and you should be presented with the following plot when it completes:



Note: When you create the parallel pool with `parpool()` a batch job is submitted to SLURM. Your MATLAB script will only continue once this job is scheduled by SLURM and is in the RUNNING state. This is why we recommend running MATLAB from the ARC Graphical (NX) nodes where you can disconnect your session without terminating your client MATLAB session.

8.14 ORCA

Introduction

ORCA is a flexible, efficient and easy-to-use general purpose tool for quantum chemistry with specific emphasis on spectroscopic properties of open-shell molecules. It features a wide variety of standard quantum chemical methods ranging from semiempirical methods to DFT to single- and multireference correlated ab initio methods. It can also treat environmental and relativistic effects.

Module information:

```
module spider orca

----------------------------------------------------------------------------
ORCA:
----------------------------------------------------------------------------
    Versions:
      ORCA/4.2.1-gompi-2019b
      ORCA/5.0.0-gompi-2019b
      ORCA/5.0.1-gompi-2019b-dba
      ORCA/5.0.2-gompi-2019b-dba
      ORCA/5.0.3-gompi-2021b
```

Example submission script:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --time=12:00:00
#SBATCH --partition=short
#SBATCH --job-name=ORCA-Test

module load ORCA/5.0.3-gompi-2021b

# ORCA must be called with full pathname, so we use $EBROOTORCA environment variable to␣
↪help...

$EBROOTORCA/orca input.inp &> output.out
```

Note: In the above example script `input.inp` should be substituted for the appropriate name of your input file, and the number of nodes and tasks per node should be set accordingly.

8.15 Quantum ESPRESSO

Introduction

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.

The guide shows how to:

- prepare a QE job submission script and
- submit and run a QE job.

Module information:

```
module spider quantum

     Versions:
       QuantumESPRESSO/6.5-intel-2020a
       QuantumESPRESSO/6.6-foss-2020b
       QuantumESPRESSO/6.6-intel-2020a
       QuantumESPRESSO/6.7-foss-2020b
       QuantumESPRESSO/6.7-intel-2020a
       QuantumESPRESSO/7.1-foss-2022a
       QuantumESPRESSO/7.1-intel-2022a
```

Running a Quantum ESPRESSO job

This example assumes you have the QE input file `scf.in` in the same directory as the submission script.

An example submission script would look as follows - create a file named `run-qe.sh` containing:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --mem-per-cpu=8000
#SBATCH --partition=short
#SBATCH --time=12:00:00
#SBATCH --job-name=QE-Test

module purge
module load QuantumESPRESSO/6.7-foss-2020b

mpirun pw.x < scf.in > scf.out
```

To launch into execution, issue the command:

```
sbatch run-qe.sh
```

8.16 Qiskit

Introduction

Qiskit [quiss-kit] is an open-source SDK for working with quantum computers at the level of pulses, circuits, and application modules.

Module Information:

```
module spider Qiskit

----------------------------------------------------------------------------------------------------
↪----------------------------------------------------------------------------------------------------
↪-----------------------
 Qiskit: Qiskit/0.23.1-foss-2020a-Python-3.8.2
----------------------------------------------------------------------------------------------------
↪----------------------------------------------------------------------------------------------------
↪-----------------------
   Description:
     Qiskit is an open-source framework for working with noisy quantum computers at the␣
↪level of pulses, circuits, and algorithms.
```

If you need to use a newer version of Qiskit than that installed centrally on the cluster via modules, you may build a Python virtual environment and install it locally in your $DATA area.

Example Virtual Environment Build

We provide below an example virtual environment build for version 0.37.0 of Qiskit with an appropriate submission script. If you require a different version, specify the version in the variable `QUISKIT_VER You can find available version numbers here: https://pypi.org/project/qiskit/#history

Virtual Environment Build Steps:

```
srun -p interactive --pty /bin/bash
```

(continues on next page)

(continued from previous page)

```
module purge
module load Anaconda3/2022.05

export QISKIT_VER=0.37.0
conda create -y --prefix $DATA/qiskit-$QISKIT_VER-env --copy python=3.9
source activate $DATA/qiskit-$QISKIT_VER-env

pip install qiskit==$QISKIT_VER
```

Example Submission Script

The example submission script below is suitable for running on the ARC cluster

```
#!/bin/bash

#SBATCH --partition=devel
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=QiskitTest

module purge
module load Anaconda3/2022.05

export QISKIT_VER=0.37.0
source activate $DATA/qiskit-$QISKIT_VER-env

python (your python script here)
```

Warning: If you activate the virtual environment in the submission script, ensure that when you submit your job that no virtual environment is active. i.e. ensure you run `conda deactivate` before running `sbatch`

8.17 Stata

Introduction

Stata is a commercial statistical package, which provides a complete solution for data analysis, data management, and graphics. Stata version 14 is available in the multi-threaded package variant Stata/MP.

Stata/MP is capable of taking advantage of the multiple cores available on the cluster nodes and the product is licensed by ARC to run on a maximum of 8 cores per job. The product chooses the number of cores to run on automatically, but this depends on the algorithm used (not all methods can run on more than one core) and has an impact on scalability.

The guide shows how to

- load the Stata module;
- prepare a Stata job submission script and
- submit the Stata job.

Running a Stata job

First, you need to load the module for the Stata package, making the executables available in the path:

```
module load Stata/14
```

Then, you need to prepare a submission script for the Stata job, which should look something like this:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=01:00:00
#SBATCH --job-name=testStata

module purge
module load Stata/14

stata-mp -s do test.do
```

The script requests a single cluster node (nodes=1) and it is up to Stata/MP to make use of up to the 8 cores it is licensed to run on. Assuming you have named the above SLURM script `run-stata.sh` the jobs can be sent to the queue with the command:

```
sbatch run-stata.sh
```

The processing is in "batch mode" and all the Stata commands are input from the file `test.do` Below the contents of the example file `test.do` is given, which contains commands that make use of the multi-core capability of Stata, and can be used for testing purposes:

```
clear*
set rmsg on
set obs 10000000
forval n = 1/5 {
g i`n' = runiform()
}
g dv = rbinomial(1,.3)
memory

qui logit dv i*

qui xtmixed dv i*

*with bootstrap:
qui bs, reps(2000): logit dv i*
```

8.18 TensorFlow

Introduction

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.

TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

TensorFlow is capable of taking advantage of the GPU nodes in the ARC HTC cluster

The guide shows how to

- prepare a TensorFlow Anaconda virtual environment and
- submit a test TensorFlow job.

To use tensorflow you can either use the pre-installed modules or build your own Python virtual environment.

Using the pre-installed versions of Tensorflow:

```
module spider Tensorflow

     TensorFlow/1.15.2-fosscuda-2019b-Python-3.7.4
     TensorFlow/2.1.0-foss-2019b-Python-3.7.4
     TensorFlow/2.3.1-foss-2020a-Python-3.8.2
     TensorFlow/2.4.1-fosscuda-2020b
     TensorFlow/2.5.0-fosscuda-2020b
     TensorFlow/2.5.3-foss-2021a-CUDA-11.3.1
     TensorFlow/2.6.0-foss-2021a-CUDA-11.3.1
     TensorFlow/2.7.1-foss-2021b-CUDA-11.4.1
```

And load the appropriate version. For example:

```
module load TensorFlow/2.4.1-fosscuda-2020b
```

The above will load Tensorflow 2.4.1 into your environment, and the package will be available from within Python.

Setting up your own virtual environment (Python 3)

Tensorflow is best run on the HTC systems which have GPU nodes. The following commands show how you can set up an Anaconda virtual environment for TensorFlow. Note: this method will only work on HTC nodes, and the environment build process should be run from an interactive session...:

```
srun -p interactive --pty /bin/bash

module load Anaconda/2020.11
module load CUDA/11.1.1-GCC-10.2.0
module load cuDNN/8.0.4.30-CUDA-11.1.1

export CONPREFIX=$DATA/tensor-env
mkdir $CONPREFIX
conda create --prefix $CONPREFIX --copy python=3.8

source activate $CONPREFIX
conda install tensorflow-gpu
```

A bash script for submission to GPU nodes will be something like this:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:10:00
#SBATCH --job-name=tensor
#SBATCH --cluster=htc
#SBATCH --gres=gpu:1
```

(continues on next page)

(continued from previous page)

```
# Useful job diagnostics
#
echo "CUDA Devices(s) allocated: $CUDA_VISIBLE_DEVICES"
nvidia-smi
#
source activate $DATA/tensor-env

python -c "import tensorflow as tf; tf.enable_eager_execution(); print(tf.reduce_s um(tf.
↪random_normal([1000, 1000])))"
```

Note: The job diagnostic information is useful if you need to contact ARC support if you have problems running your job - so please include it.

Warning: You cannot run TensorFlow in GPU mode directly from the login nodes. This will result in errors - as these systems have no GPUs available, and they are restricted in memory.

8.19 VASP

Introduction

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.

Warning: VASP is licensed software and you require a licence to run it. Licensing information can be found here: https://www.vasp.at/

Once you have a valid licence, you need to request access via this form: https://www.arc.ox.ac.uk/restricted-licence-software-applications to request access on ARC.

The guide shows how to:

- prepare a VASP job submission script and
- submit and run a VASP job.

Module information:

```
module use /apps/common/private/modules
module spider vasp

    Versions:
       VASP/1.0-info
       VASP/5.4.4-intel2020b
       VASP/6.2.1-foss2020a
       VASP/6.3.2-foss2020a
```

Note: The modules with the suffix `-info` should not be loaded.

Running a VASP job

This example assumes you have the VASP input files in the same directory as the submission script.

An example submission script would look as follows - create a file named `run-vasp.sh` containing:

```
#!/bin/bash

#SBATCH --clusters=arc
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=00:10:00
#SBATCH --job-name=VASP
#SBATCH --partition=devel

module purge

module use /apps/common/private/modules
module load VASP/6.3.2-foss2020a

mpirun vasp_std
```

The script requires two nodes (48 cores per node) and launches VASP taking the input from the file INCAR

Note: VASP parallelisation is dependent on the simulation you are running see: https://www.vasp.at/wiki/index.php/Category:Parallelization for more information.

To launch into execution, issue the command:

```
sbatch run-vasp.sh
```