Jacek Dziedzic
University of Southampton
2022.06.04, v2.60

Instructions for compiling and running ONETEP on ARCHER2

These instructions are for the full 23-cabinet ARCHER2 system introduced on 2021.11.22. The 4-cabinet system is no longer supported.

As of 2022.06.04 on ARCHER2 you can either use a centrally-installed binary (which is recommended for new users), or you can compile your own. In either case, there are three supported compiler + library combinations available:

- **Cray Fortran** + libsci – recommended,
- **GNU Fortran** + libsci – alternative,
- **GNU Fortran** + MKL – alternative.

Using a centrally-installed binary is the easiest option, but it will only let you use a particular ONETEP version that has been compiled for you. As of 2022.06.04 this is ONETEP v6.1.9.0. When you compile your own, you can use the newest available version.

The following table lists the details of the compiler and libraries in each of the three scenarios and the earliest ONETEP version that supports these combinations. It also shows approximate times to run the QC-test suite, which can serve as a rough guide to performance.

	Compiler	**MPI**	**OMP**	**FFTW**	**BLAS and LAPACK**	**ScaLAPACK**	**Time to run QC tests (8 threads)**	**Time to run QC tests (16 threads)**
Cray+libsci (v6.1.7.4 or newer)	Cray Fortran 12.0.3	yes, mpich 8.1.4	yes	Cray 3.3.8.11 x86_rome	libsci 21.04.1.1 CRAY/9.0	libsci 21.04.1.1 CRAY/9.0	6747 s	5244 s
GNU+libsci (v6.1.7.10 or newer)	GNU Fortran 9.3.0	yes, mpich 8.1.4	yes	Cray 3.3.8.11 x86_rome	libsci 21.04.1.1 GNU/9.1	libsci 21.04.1.1 GNU/9.1	7643 s	6033 s
GNU+MKL (v6.1.7.4 or newer)	GNU Fortran 9.3.0	yes, mpich 8.1.4	yes	MKL 21.2-2883	MKL 21.2-2883	MKL 21.2-2883	7542 s	5755 s

All of these use ScaLAPACK and are OpenMP-capable. All of these use mpich 8.1.4 for MPI.

The **Cray Fortran** version is faster and it scales better to higher numbers of threads. We recommend users use this version.

The **GNU Fortran** versions are not as fast as the Cray Fortran version, but the GNU compiler is less finicky and the result may be somewhat more stable. The MKL version is slightly faster than the libsci version (but not as fast as Cray Fortran), and it scales better to higher numbers of threads.

Outstanding known issues on ARCHER2

- Worked around:
  - `LD_LIBRARY_PATH` is not set correctly and requires manual adjustment. This is taken care of by the config files and the submission scripts. Fore more info, see: https://docs.archer2.ac.uk/known-issues/#default-fftw-library-points-to-intel-haswell-version-rather-than-amd-rome-at-runtime-added-2022-02-23. **No actions needed from your side**.
  - ARCHER2's libfabric leaks memory, which, for instance, causes MKL's `PDSYGVX()` to leak when ScaLAPACK is used together with OpenMP (as is the default). Two workarounds are in place since ONETEP v6.1.7.3 – we turn off MKL's threading for this call, and to stop the memory leak in other contexts, we also sett the environment variable `FI_MR_CACHE_MAX_COUNT` to 0 in all submission scripts. **No actions needed from your side, but be aware that this hurts performance** whenever diagonalisation is performed (EDFT, mostly) and MKL is used. Reported to ARCHER2 helpdesk as Q1839965. Closed, as this is meant to be fixed in a new ARCHER2 PE.
  - ARCHER2's libsci's implementation gives noisy results for `PDSYGVX()` when ScaLAPACK is used together with OpenMP (as is the default). A workaround is in place since ONETEP v6.1.6.0 that turns off libsci's threading for this call. Reported to ARCHER2 helpdesk as Q1807973 and reproduced by HPE. It is said to be fixed in CPE 21.12's cray-libsci/21.08.1.2. This CPE is not available on ARCHER2 as of 2022.04.29. **No actions needed from your side, but be aware that this hurts performance** whenever diagonalisation is performed (EDFT, mostly) and libsci is used.
  - ARCHER2's libsci's implementation of `dgesv()` performs poorly with GNU Fortran, particularly for larger numbers of threads. This LAPACK call is only used in DMA and HFx, and will be used in the future in DFTB. Reported to ARCHER2 helpdesk as Q1767358. Reproduced by HPE, but no resolution offered except manually reducing the number of OMP threads before the call. This is what we do since v6.1.7.10. **No actions needed from your side**.
  - With Cray in debug mode multiple QC tests fail in `ngwf_gradient_mod` due to the OMP thread stack overflowing. This is a known issue (#1891). The arrays in question are actually allocated inside an OMP region (marked as PRIVATE) and are then passed to `ngwf_gradient_local()` as `intent(out)` arguments, so they should not be winding up on the thread stack – blame the compiler. **Release mode works fine.** The problem can be worked around by increasing OMP_STACKSIZE to 128M in debug mode by adding `-o 128M` to `onetep_launcher`'s command line in the submission script.
  - The latest MPI version on ARCHER2 – mpich 8.1.9 – is broken in its implementation of `MPI_CANCEL()`. This breaks all DMA and HFx functionality and the corresponding QC tests. Worked around – all our config files and submission scripts use mpich 8.1.4, so **no actions needed from your side**. Reported to ARCHER2 helpdesk as Q1788013. HPE acknowledges this is known as internal bug PE-37224 - segmentation fault in MPI_Cancel() on a receive request that uses MPI_ANY_SOURCE with single node runs and it is said to be fixed in CPE 21.10 (mpich 8.1.10). This CPE is not available on ARCHER2 as of 2022.04.29.
- Known bugs in ONETEP:
  - QC test #81 often fails and spectral function unfolding does not work correctly. This is a known issue (#1866) with ONETEP and is not limited to ARCHER2.
- Under investigation:
  - All or almost all QC tests fail with gfortran in debug mode by SIGFPE'ing in `dgemm_kernel_loop_mnk_a1b1_naples()`.
  - QC test #69 (conduction) fails with MKL in debug mode by SIGFPE'ing in `pdstein()`.
  - Linking with non-OMP-capable libsci (which is not recommended, see the last page of this document) causes additional QC test fails:
    - 07-conduction,
    - 68.

    These both seem like segfaults in libsci.

Instructions for the centrally-installed binaries (recommended for new users)

1. Basics

There is no need to set up the environment. There is no need to compile anything, the binaries have been compiled for you. You only need to load the correct module to make ONETEP accessible. The ONETEP modules are only available to ARCHER2 users who have a valid licence and applied via the SAFE. See more at : https://docs.archer2.ac.uk/research-software/onetep/onetep/

To load the correct module issue **one** of the following commands on ARCHER2:

```
module load onetep/6.1.9.0-CCE-LibSci
```
or
```
module load onetep/6.1.9.0-GCC-LibSci
```
or
```
module load onetep/6.1.9.0-GCC-MKL
```

which correspond to the three rows in the table on p. 1. The first of these commands is already contained in the submission scripts provided with ONETEP in the `hpc_resources/ARCHER2` directory (`qcsubmit.ARCHER2.2022_central`, `jobsubmit.ARCHER2.2022_central`), so if you want the default option, you do not need to do anything. If you want one of the other two, you will have to replace this line with one of the other two alternatives.

2. Testing

If you are new to ARCHER2, it would be prudent to test ONETEP by running the suite of quality-checks ("QCs") bundled with it. These test a number of important functionalities by running short jobs and comparing results against known good values. To run the QC-test suite, copy the QC-test submission script `qcsubmit.ARCHER2.2022_central` (provided in the directory `hpc_resources/ARCHER2` of your ONETEP installtion) to the `tests` directory of your ONETEP installation. Then, edit this file to provide your *budget code*. The default budget code is `e89-soto`, which is suitable for Southampton at the time this document is written. If you need to change this, replace `e89-soto` (at line 36) with the correct budget code.

Then submit it to the batch system by changing to the `tests` directory and issuing

```
sbatch qcsubmit.ARCHER2.2022_central
```

Once the testing job starts, you should see, in the same directory, a file called `slurm-`*`nnnnn`*`.out`, where *`nnnnn`* is the ID (number) of your job. This contains the log of how your job is running – diagnostic messages from the batch system and the submission script, and any output from `testcode` – the python script that actually runs the tests. If everything goes well, you should see subsequent test numbers (there are about 85 of them), followed by "Passed" or, occasionally "WARNING". The warnings can usually be ignored. If there are any errors, the test number will be followed by "FAILED". If this happens, go to the directory of this particular test and examine the `test.out.*`, `test.err.*` and (potentially) `*.error_message` files. They should give you an idea about what went wrong. Once the tests complete, you will see a file called **`%DONE`** in your `tests` directory. This does not necessarily mean that all tests completed successfully. You should examine the `slurm-`*`nnnnn`*`.out` file for a line like this:

```
All done. 108 out of 108 tests passed (12 warnings).
```

If all tests passed – you're good to go. If only QC test #81 fails – you're good to go too – this is a known issue at the time this document is written.

3. Running

Use the provided submission script: `jobsubmit.ARCHER2.2022_central` Place it in a directory where the input files for your run are. Make sure you only run one calculation in a directory. Do not run multiple calculations (multiple `.dat` files) in a single directory – this will lead to a mess and the provided script disallows it.

Adjust the script to your liking by editing its first lines (see below). Submit it with: `sbatch jobsubmit.ARCHER2.2022_central`. Details on how to monitor and control submitted and running jobs can be found on the ARCHER2 support page.

```
18  # ================================================================================================================
19  # Edit the following lines to your liking.
20  #
21  #SBATCH --job-name=mine                # Name of the job.
22  #SBATCH --ntasks=32                    # Total number of MPI processes in job.
23  #SBATCH --nodes=2                      # Number of nodes in job.
24  #SBATCH --ntasks-per-node=16           # Number of MPI processes per node.
25  #SBATCH --cpus-per-task=8              # Number of OMP threads spawned from each MPI process.
26  #SBATCH --time=5:00:00                 # Max time for your job (hh:mm:ss).
27  #SBATCH --partition=standard           # Queue. standard CPU nodes with AMD EPYC 7742 64-core processor
28  #SBATCH --account=e89-soto             # Replace 'e89-soton' with your budget code.
29  #SBATCH --qos=standard                 # Requested Quality of Service (QoS), See ARCHER2 documentation
30
31  export OMP_NUM_THREADS=8               # Repeat the value from 'cpus-per-task' here.
32
33  # Set up the job environment, loading the ONETEP module.
34  # The module automatically sets OMP_PLACES, OMP_PROC_BIND and FI_MR_CACHE_MAX_COUNT.
35  # To use a different binary, replace this line with either (drop the leading '#')
36  # module load onetep/6.1.9.0-GCC-LibSci
37  # to use the GCC-libsci binary, or with
38  # module load onetep/6.1.9.0-GCC-MKL
39  # to use the GCC-MKL binary.
40
41  module load onetep/6.1.9.0-CCE-LibSci
42
43  # ================================================================================================================
44  # !!! You should not need to modify anything below this line.
45  # ================================================================================================================
```

Things to adjust by line number:

21 The name of your job. It has no effect on how it's run, but it will help you distinguish it from any other jobs you might have.

22 The *total* number of **tasks** (MPI processes) you want to start, summed over all nodes. In typical usages you will want to run 16 MPI processes per node – then this would be 16 times the number of nodes you want.

23 The *total* number of **nodes** (machines) you are asking for. Each ARCHER2 node has **128 CPU cores**.

24 The number of MPI processes (tasks) *per node* you want to run. I suggest using 16.

25 The number of OpenMP *threads* spawned from each MPI process. I suggest using 8 (when using 16 tasks per node). In this way each node is saturated, using all the 128 CPU cores. Note that this value needs to be repeated several lines later.

26 The walltime you are asking for in `hh:mm:ss`. The maximum for the standard quality of service is `24:00:00`. Don't exceed this value or your job will never start.

27 Choose the type of queue or partition, default to `standard`.

28 Your account code for which you have budget available.

29 The quality of service, which defines the maximum number of nodes, walltime and number of concurrent jobs in queue.

31 Repeat the value from line 25 here.

41 The `module load` command, as explained in "1. Basics" if you want a non-default compiler+library combination.

Once you adjusted the above, you are ready to submit your job. The submission script, together with `onetep_launcher`, will take care of everything. There is no need for you to set the stack size, `OMP_STACKSIZE`, or `OMP_NUM_THREADS`. Also, **do not** add any `threads_` keywords to your input file. It's all handled by the script, really.

Now read sections 5 and 6 from the GNU Fortran version – these are common to all the installs.

Instructions for the Cray Fortran version (recommended)

1. Setting up the environment

You will need to load the correct set of modules. The provided config file (`conf.archer2_2022_cray`) is guaranteed to work with Cray Fortran v12.0.3, MPICH v8.1.4 (default), and Cray FFTW 3.3.8.11. To do this, execute the following lines before compiling ONETEP:

```
module load PrgEnv-cray/8.1.0
module load cce/12.0.3
module load cray-fftw/3.3.8.11
module load cray-mpich/8.1.4
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

The last line takes of a known issue "*Default FFTW library points to Intel Haswell version rather than AMD Rome at runtime*" described at https://docs.archer2.ac.uk/known-issues/

Ignore any warning messages. You might wish to add these commands to the end of your `~/.bash_profile` to make this happen automatically:

```
module load PrgEnv-cray/8.1.0  >/dev/null 2>/dev/null
module load cce/12.0.3         >/dev/null 2>/dev/null
module load cray-fftw/3.3.8.11 >/dev/null 2>/dev/null
module load cray-mpich/8.1.4   >/dev/null 2>/dev/null
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

2. Compilation

Use the config file `conf.archer2_2022_cray`. Issue the command below while in the directory of your ONETEP installation:

```
make onetep ARCH=archer2_2022_cray
```

This should compile and link a working binary, which will be placed in the `bin` subdirectory. If you tend to do this more often you might like the following command more:

```
make onetep ARCH=archer2_2022_cray -j | utils/colorise
```

It compiles many files in parallel (so it's faster) and colorises output to make it more readable.

3. Testing

If you have just compiled a new binary for ARCHER2, it would be prudent to test it by running the suite of quality-checks ("QCs") bundled with ONETEP. These test a number of important functionalities by running short jobs and comparing results against known good values. To run the QC-test suite, copy the QC-test submission script `qcsubmit.ARCHER2.2022_cray` (provided in the directory `hpc_resources/ARCHER2` of your ONETEP installtion) to the `tests` directory of your ONETEP installation. Then, edit this file to provide your *budget code*. The default budget code is `e89-soto`, which is suitable for Southampton at the time this document is written. If you need to change this, replace `e89-soto` (at line 36) with the correct budget code.

Then submit it to the batch system by changing to the `tests` directory and issuing

```
sbatch qcsubmit.ARCHER2.2022_cray
```

Once the testing job starts, you should see, in the same directory, a file called `slurm-nnnnn.out`, where *nnnnn* is the ID (number) of your job. This contains the log of how your job is running – diagnostic messages from the batch system and the submission script, and any output from `testcode` – the python script that actually runs the tests. If everything goes well, you should see subsequent test numbers (there are about 85 of them), followed by "Passed" or, occasionally "WARNING". The warnings can usually be ignored. If there are any errors, the test number will be followed by "FAILED". If this happens, go to the directory of this particular test and examine the `test.out.*`, `test.err.*` and (potentially) `*.error_message` files. They should give you an idea about what went wrong. Once the tests complete, you will see a file called **`%DONE`** in your `tests` directory. This does not necessarily mean that all tests completed successfully. You should examine the `slurm-nnnnn.out` file for a line like this:

```
All done. 108 out of 108 tests passed (12 warnings).
```

If all tests passed – you're good to go. If only QC test #81 fails – you're good to go too – this is a known issue at the time this document is written.

4. Running

Use the provided submission script: `jobsubmit.ARCHER2.2022_cray`. Place it in a directory where the input files for your run are. Make sure you only run one calculation in a directory. Do not run multiple calculations (multiple `.dat` files) in a single directory – this will lead to a mess and the provided script disallows it.

Adjust the script to your liking by editing its first lines (see below). Submit it with: `sbatch jobsubmit.ARCHER2.2022_cray`. Details on how to monitor and control submitted and running jobs can be found on the ARCHER2 support page.

```
22  # ================================================================================================================
23  # Edit the following lines to your liking.
24  #
25  #SBATCH --job-name=mine               # Name of the job.
26  #SBATCH --ntasks=32                   # Total number of MPI processes in job.
27  #SBATCH --nodes=2                     # Number of nodes in job.
28  #SBATCH --ntasks-per-node=16          # Number of MPI processes per node.
29  #SBATCH --cpus-per-task=8             # Number of OMP threads spawned from each MPI process.
30  #SBATCH --time=5:00:00                # Max time for your job (hh:mm:ss).
31  #SBATCH --partition=standard          # Queue. standard CPU nodes with AMD EPYC 7742 64-core processor
32  #SBATCH --account=e89-soto            # Replace 'e89-soton' with your budget code.
33  #SBATCH --qos=standard                # Requested Quality of Service (QoS), See ARCHER2 documentation
34
35  export OMP_NUM_THREADS=8              # Repeat the value from 'cpus-per-task' here.
36
37  # Point this to your ONETEP executable.
38  onetep_exe=\
39  "/work/e89/e89/jacek/onetep_cray/bin/onetep.archer2_2022_cray"
40
41  # Point this to your ONETEP launcher.
42  onetep_launcher=\
43  "/work/e89/e89/jacek/onetep_cray/utils/onetep_launcher"
44
45  # ================================================================================================================
```

Things to adjust by line number:

42 The name of your job. It has no effect on how it's run, but it will help you distinguish it from any other jobs you might have.
43 The *total* number of **tasks** (MPI processes) you want to start, summed over all nodes. In typical usages you will want to run 16 MPI processes per node – then this would be 16 times the number of nodes you want.
44 The *total* number of **nodes** (machines) you are asking for. Each ARCHER2 node has **128 CPU cores**.
45 The number of MPI processes (tasks) *per node* you want to run. I suggest using 16.
46 The number of OpenMP *threads* spawned from each MPI process. I suggest using 8 (when using 16 tasks per node). In this way each node is saturated, using all the 128 CPU cores. Note that this value needs to be repeated several lines later.
47 The walltime you are asking for in `hh:mm:ss`. The maximum for the standard quality of service is `24:00:00`. Don't exceed this value or your job will never start.
48 Choose the type of queue or partition, default to `standard`.
49 Your account code for which you have budget available.
50 The quality of service, which defines the maximum number of nodes, walltime and number of concurrent jobs in queue.
51 Repeat the value from line 29 here.
39 Path to your ONETEP executable. Change it to the location of your binary.
43 Path to the `onetep_launcher` script. Change it to the location of `onetep_launcher`. It is located in the `utils` subdirectory of the ONETEP installation.

Once you adjusted the above, you are ready to submit your job. The submission script, together with `onetep_launcher`, will take care of everything. There is no need for you to set the stack size, `OMP_STACKSIZE`, or `OMP_NUM_THREADS`. Also, **do not** add any `threads_` keywords to your input file. It's all handled by the script, really.

Now read sections 5 and 6 from the GNU Fortran version – these are common to all the installs.

Instructions for the GNU Fortran version (also works, but will likely be slower)

1. Setting up the environment

You will need to load the correct set of modules. The provided config file (`conf.archer2_2022_gfortran`) is guaranteed to work with GNU Fortran v9.3.0, MPICH v8.1.4 (default) and Cray FFTW. To do this, execute the following lines before compiling ONETEP:

```
module load PrgEnv-gnu
module load gcc/9.3.0
module load cray-fftw
module load cray-mpich/8.1.4
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

The last line takes of a known issue "*Default FFTW library points to Intel Haswell version rather than AMD Rome at runtime*" described at https://docs.archer2.ac.uk/known-issues/

Ignore any warning messages. You might wish to add these commands to the end of your `~/.bash_profile` to make this happen automatically:

```
module load PrgEnv-gnu        >/dev/null 2>/dev/null
module load gcc/9.3.0         >/dev/null 2>/dev/null
module load cray-fftw         >/dev/null 2>/dev/null
module load cray-mpich/8.1.4 >/dev/null 2>/dev/null
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

2. Compilation

Use the config file `conf.archer2_2022_gfortran`. Issue the command below while in the directory of your ONETEP installation:

```
make onetep ARCH=archer2_2022_gfortran
```

This should compile and link a working binary, which will be placed in the `bin` subdirectory. If you tend to do this more often you might like the following command more:

```
make onetep ARCH=archer2_2022_gfortran -j | utils/colorise
```

It compiles many files in parallel (so it's faster) and colorises output to make it more readable.

3. Testing

If you have just compiled a new binary for ARCHER2, it would be prudent to test it by running the suite of quality-checks ("QCs") bundled with ONETEP. These test a number of important functionalities by running short jobs and comparing results against known good values. To run the QC-test suite, copy the QC-test submission script `qcsubmit.ARCHER2.2022_gfortran` (provided in the directory `hpc_resources/ARCHER2` of your ONETEP installtion) to the `tests` directory of your ONETEP installation. Then, edit this file to provide your *budget code*. The default budget code is `e89-soto`, which is suitable for Southampton at the time this document is written. If you need to change this, replace `e89-soto` (at line 35) with the correct budget code.

Then submit it to the batch system by changing to the `tests` directory and issuing

```
sbatch qcsubmit.ARCHER2.2022_gfortran
```

Once the testing job starts, you should see, in the same directory, a file called `slurm-`*`nnnnn`*`.out`, where *`nnnnn`* is the ID (number) of your job. This contains the log of how your job is running – diagnostic messages from the batch system and the submission script, and any output from `testcode` – the python script that actually runs the tests. If everything goes well, you should see subsequent test numbers (there are about 85 of them), followed by "Passed" or, occasionally "WARNING". The warnings can usually be ignored. If there are any errors, the test number will be followed by "FAILED". If this happens, go to the directory of this particular test and examine the `test.out.*`, `test.err.*` and (potentially) `*.error_message` files. They should give you an idea about what went wrong. Once the tests complete, you will see a file called **`%DONE`** in your `tests` directory. This does not necessarily mean that all tests completed successfully. You should examine the `slurm-`*`nnnnn`*`.out` file for a line like this:

```
All done. 108 out of 108 tests passed (12 warnings).
```

If all tests passed – you're good to go. If only test 81 failed, you're good to go too – this is a known issue at the time this document is written.

4. Running

Use the provided submission script: `jobsubmit.ARCHER2.2022_gfortran`. Place it in a directory where the input files for your run are. Make sure you only run one calculation in a directory. Do not run multiple calculations (multiple `.dat` files) in a single directory – this will lead to a mess and the provided script disallows it.

Adjust the script to your liking by editing its first lines (see below). Submit it with: `sbatch jobsubmit.ARCHER2.2022_gfortran`. Details on how to monitor and control submitted and running jobs can be found on the ARCHER2 support page.

```
22  # ================================================================================================
23  # Edit the following lines to your liking.
24  #
25  #SBATCH --job-name=mine              # Name of the job.
26  #SBATCH --ntasks=16                  # Total number of MPI processes in job.
27  #SBATCH --nodes=2                    # Number of nodes in job.
28  #SBATCH --ntasks-per-node=16         # Number of MPI processes per node.
29  #SBATCH --cpus-per-task=8            # Number of OMP threads spawned from each MPI process.
30  #SBATCH --time=5:00:00               # Max time for your job (hh:mm:ss).
31  #SBATCH --partition=standard         # Queue. standard CPU nodes with AMD EPYC 7742 64-core processor
32  #SBATCH --account=e89-soto           # Replace 'e89-soton' with your budget code.
33  #SBATCH --qos=standard               # Requested Quality of Service (QoS), See ARCHER2 documentation
34
35  export OMP_NUM_THREADS=8             # Repeat the value from 'cpus-per-task' here.
36
37  # Point this to your ONETEP executable.
38  onetep_exe=\
39  "/work/e89/e89/jacek/onetep_gfortran/bin/onetep.archer2_2022_gfortran"
40
41  # Point this to your ONETEP launcher.
42  onetep_launcher=\
43  "/work/e89/e89/jacek/onetep_gfortran/utils/onetep_launcher"
44
45  # ================================================================================================
```

Things to adjust by line number:

25 The name of your job. It has no effect on how it's run, but it will help you distinguish it from any other jobs you might have.

26 The *total* number of **tasks** (MPI processes) you want to start, summed over all nodes. In typical usages you will want to run 8 MPI processes per node – then this would be eight times the number of nodes you want.

27 The *total* number of **nodes** (machines) you are asking for. Each ARCHER2 node has **128 CPU cores**.

28 The number of MPI processes (tasks) *per node* you want to run. I suggest using 16.

29 The number of OpenMP *threads* spawned from each MPI process. I suggest using 8 (when using 16 tasks per node). In this way each node is saturated, using all the 128 CPU cores. Note that this value needs to be repeated several lines later.

30 The walltime you are asking for in `hh:mm:ss`. The maximum for the standard quality of service is `24:00:00`. Don't exceed this value or your job will never start.

31 Choose the type of queue or partition, default to `standard`.

32 Your account code for which you have budget available.

33 The quality of service, which defines the maximum number of nodes, walltime and number of concurrent jobs in queue.

34 Repeat the value from line 29 here.

39 Path to your ONETEP executable. Change it to the location of your binary.

43 Path to the `onetep_launcher` script. Change it to the location of `onetep_launcher`. It is located in the `utils` subdirectory of the ONETEP installation.

Once you adjusted the above, you are ready to submit your job. The submission script, together with `onetep_launcher`, will take care of everything. There is no need for you to set the stack size, `OMP_STACKSIZE`, or `OMP_NUM_THREADS`. Also, **do not** add any `threads_` keywords to your input file. It's all handled by the script, really.

5. Files produced by the submission script

The submission script will try to communicate any error or success conditions by creating files with filenames beginning with `%` (so they are easy to spot in the file listing) in the directory of your run. These include:

`%NO_DAT_FILE` – there are no `.dat` files in your job directory. Job did not start.

`%MORE_THAN_ONE_DAT_FILE` – there are multiple `.dat` files in your job directory. Job did not start. This script only works correctly in a one-dat-file-per-run set-up.

`%ONETEP_EXE_MISSING` – the path you provided does not point to a valid ONETEP executable. It either does not exist or is not executable. Job did not start.

`%ONETEP_LAUNCHER_MISSING` – the path you provided does not point to a valid `onetep_launcher`. It either does not exist or is not executable. Job did not start.

`%SRUN_ERROR` – `srun`, the MPI wrapper script on ARCHER2 reported an error. You job did not start or failed. Examine the standard error file for any error messages.

`%ONETEP_ERROR_DETECTED` – ONETEP failed gracefully, producing an error message. Job ran, but failed. Examine the `.error_message` file to see what happened.

`%ONETEP_DID_NOT_COMPLETE` – Job likely started, but it doesn't look like it's completed, even though there's no error message. It's likely your job deadlocked (got stuck), crashed non-gracefully (due to a bug or running out of memory) or ran out of walltime. Examine the standard error file for any error messages and the ONETEP output for any hints on what might have happened.

`%DONE` – your calculation ran to completion with no apparent errors.

6. Crucial files

Assuming your ONETEP input is called **`myinput.dat`**, these are some important files in your run.

- **`myinput.out`**: your ONETEP output.
- **`myinput.err`**: the standard error output of your job. It will be an empty file if everything goes well. Otherwise, it will contain error messages from the OS, the batch system, `srun`, MPI or the Fortran RTL.
- **`myinput.error_message`**: if ONETEP fails gracefully, this will contain an error message.
- **`slurm-`*`nnnnn`*`.out`**, where ***`nnnnn`*** is the ID (number) of your job. This contains the log of how your job ran – diagnostic messages from the batch system and the submission script. On successful completion it will also contain usage statistics for the nodes you used.
- **`$modules_loaded`**: The list of modules actually loaded when the job is run is echoed here. You might want to examine it if you have any doubts if they match your expectations.
- **`$modules`**: Any output from the `module load` commands issued from the submission script. You might want to examine it if you have any doubts if they match your expectations.
- **`$ldd`**: The output of the `ldd` command on your ONETEP executable. You might want to examine it if you have any doubts whether the correct libraries are used when running ONETEP.

Instructions for the GNU Fortran version with MKL

1. Setting up the environment

You will need to load the correct set of modules. The provided config file (`conf.archer2_2022_gfortran_mkl`) is guaranteed to work with GNU Fortran v9.3.0, MPICH v8.1.4 (default) and MKL's FFTW. To do this, execute the following lines before compiling ONETEP:

```
module load PrgEnv-gnu
module load gcc/9.3.0
module load mkl
module load cray-mpich/8.1.4
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

The last line takes of a known issue "*Default FFTW library points to Intel Haswell version rather than AMD Rome at runtime*" described at https://docs.archer2.ac.uk/known-issues/

Ignore any warning messages. You might wish to add these commands to the end of your `~/.bash_profile` to make this happen automatically:

```
module load PrgEnv-gnu       >/dev/null 2>/dev/null
module load gcc/9.3.0        >/dev/null 2>/dev/null
module load mkl              >/dev/null 2>/dev/null
module load cray-mpich/8.1.4 >/dev/null 2>/dev/null
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
```

2. Compilation

The MKL install on Archer2 does not provide an `mkl_service.mod` that would be ABI-compatible with GNU Fortran v9.3.0, and so we need to compile our own (this only needs to be done once). Go to your ONETEP directory and issue the following commands:

```
mkdir lib 2>/dev/null
mkdir lib/archer2_2022_gfortran_mkl
cd lib/archer2_2022_gfortran_mkl
module load mkl
cp $MKLROOT/include/mkl_service.f90 .
ftn -c mkl_service.f90
stat mkl_service.mod >/dev/null && echo "SUCCESS" || echo "It did not work."
cd -
```

These create a suitable directory in your installation's `lib` directory, and put `mkl_service.mod` there. If everything goes OK, you should see **`SUCCESS`** printed out. If something goes wrong, the printout will say **`It did not work`**.

Now that you have this out of the way, use the config file `conf.archer2_2022_gfortran_mkl`. Issue the command below while in the directory of your ONETEP installation:

```
make onetep ARCH=archer2_2022_gfortran_mkl
```

This should compile and link a working binary, which will be placed in the `bin` subdirectory. If you tend to do this more often you might like the following command more:

```
make onetep ARCH=archer2_2022_gfortran_mkl -j | utils/colorise
```

It compiles many files in parallel (so it's faster) and colorises output to make it more readable.

3. Testing

If you have just compiled a new binary for ARCHER2, it would be prudent to test it by running the suite of quality-checks ("QCs") bundled with ONETEP. These test a number of important functionalities by running short jobs and comparing results against known good values. To run the QC-test suite, copy the QC-test submission script `qcsubmit.ARCHER2.2022_gfortran_mkl` (provided in the directory `hpc_resources/ARCHER2` of your ONETEP installtion) to the `tests` directory of your ONETEP installation. Then, edit this file to provide your *budget code*. The default budget code is `e89-soto`, which is suitable for Southampton at the time this document is written. If you need to change this, replace `e89-soto` (at line 35) with the correct budget code.

Then submit it to the batch system by changing to the `tests` directory and issuing

```
sbatch qcsubmit.ARCHER2.2022_gfortran_mkl
```

Once the testing job starts, you should see, in the same directory, a file called `slurm-`*`nnnnn`*`.out`, where *`nnnnn`* is the ID (number) of your job. This contains the log of how your job is running – diagnostic messages from the batch system and the submission script, and any output from `testcode` – the python script that actually runs the tests. If everything goes well, you should see subsequent test numbers (there are about 85 of them), followed by "Passed" or, occasionally "WARNING". The warnings can usually be ignored. If there are any errors, the test number will be followed by "FAILED". If this happens, go to the directory of this particular test and examine the `test.out.*`, `test.err.*` and (potentially) `*.error_message` files. They should give you an idea about what went wrong. Once the tests complete, you will see a file called **`%DONE`** in your `tests` directory. This does not necessarily mean that all tests completed successfully. You should examine the `slurm-`*`nnnnn`*`.out` file for a line like this:

```
All done. 108 out of 108 tests passed (12 warnings).
```

If all tests passed – you're good to go. If only test 81 failed, you're good to go too – this is a known issue at the time this document is written.

4. Running

Use the provided submission script: `jobsubmit.ARCHER2.2022_gfortran_mkl`. Place it in a directory where the input files for your run are. Make sure you only run one calculation in a directory. Do not run multiple calculations (multiple `.dat` files) in a single directory – this will lead to a mess and the provided script disallows it.

Adjust the script to your liking by editing its first lines (see below). Submit it with: `sbatch jobsubmit.ARCHER2.2022_gfortran_mkl`. Details on how to monitor and control submitted and running jobs can be found on the ARCHER2 support page.

```
18  # ================================================================================================================
19  # Edit the following lines to your liking.
20  #
21  #SBATCH --job-name=mine              # Name of the job.
22  #SBATCH --ntasks=32                  # Total number of MPI processes in job.
23  #SBATCH --nodes=2                    # Number of nodes in job.
24  #SBATCH --ntasks-per-node=16         # Number of MPI processes per node.
25  #SBATCH --cpus-per-task=8            # Number of OMP threads spawned from each MPI process.
26  #SBATCH --time=1:00:00               # Max time for your job (hh:mm:ss).
27  #SBATCH --partition=standard         # Queue. standard CPU nodes with AMD EPYC 7742 64-core processor
28  #SBATCH --account=e89-soto           # Replace 'e89-soton' with your budget code.
29  #SBATCH --qos=standard               # Requested Quality of Service (QoS), See ARCHER2 documentation
30
31  export OMP_NUM_THREADS=8             # Repeat the value from 'cpus-per-task' here.
32
33  # Point this to your ONETEP executable.
34  onetep_exe=\
35  "/work/e89/e89/jacek/onetep_gfortran_mkl/bin/onetep.archer2_2022_gfortran_mkl"
36
37  # Point this to your ONETEP launcher.
38  onetep_launcher=\
39  "/work/e89/e89/jacek/onetep_gfortran_mkl/utils/onetep_launcher"
40
41  # ================================================================================================================
```

Things to adjust by line number:

21 The name of your job. It has no effect on how it's run, but it will help you distinguish it from any other jobs you might have.

22 The *total* number of **tasks** (MPI processes) you want to start, summed over all nodes. In typical usages you will want to run 8 MPI processes per node – then this would be eight times the number of nodes you want.

23 The *total* number of **nodes** (machines) you are asking for. Each ARCHER2 node has **128 CPU cores**.

24 The number of MPI processes (tasks) *per node* you want to run. I suggest using 16.

25 The number of OpenMP *threads* spawned from each MPI process. I suggest using 8 (when using 16 tasks per node). In this way each node is saturated, using all the 128 CPU cores. Note that this value needs to be repeated several lines later.

26 The walltime you are asking for in `hh:mm:ss`. The maximum for the standard quality of service is `24:00:00`. Don't exceed this value or your job will never start.

27 Choose the type of queue or partition, default to `standard`.

28 Your account code for which you have budget available.

29 The quality of service, which defines the maximum number of nodes, walltime and number of concurrent jobs in queue.

31 Repeat the value from line 25 here.

35 Path to your ONETEP executable. Change it to the location of your binary.

39 Path to the `onetep_launcher` script. Change it to the location of `onetep_launcher`. It is located in the `utils` subdirectory of the ONETEP installation.

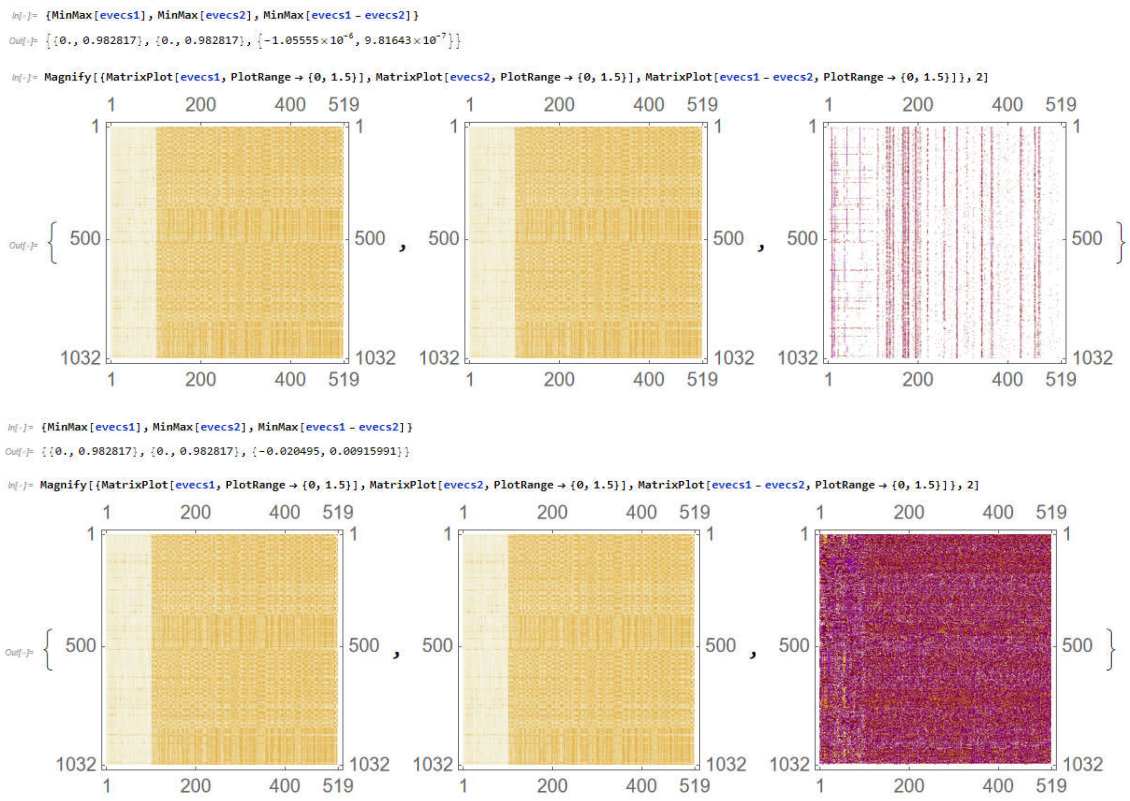
Once you adjusted the above, you are ready to submit your job. The submission script, together with `onetep_launcher`, will take care of everything. There is no need for you to set the stack size, `OMP_STACKSIZE`, or `OMP_NUM_THREADS`. Also, **do not** add any `threads_` keywords to your input file. It's all handled by the script, really.

Now read sections 5 and 6 from the GNU Fortran version – these are common to all the installs.

Linking with non-OMP versions of libsci

TL;DR: **Don't do this.** Long story: the libsci library on Archer2, broadly speaking, replaces MKL, providing the FFTW, BLAS, LAPACK and ScaLAPACK APIs. This library automatically uses threads internally if called from outside an OpenMP block, and refrains from threading if called from inside an OpenMP block. My initial tests seem to indicate that this does not work too well in the ScaLAPACK's `PDSYGVX()` call when it is used in an OMP-capable binary. In ONETEP this call is used in `dense_eigensolve()`. The eigenvectors returned from the call appear to be very noisy, much noisier than normally expected from ScaLAPACK.

Below are plots of two sets of eigenvectors obtained for identical inputs, and their difference, – first without OpenMP (top), and then with OpenMP.



The magnitude of the noise is ~1E-6 in the first instance, and ~2E-2 in the second instance. Currently we work around this issue by turning off threads before the call and turning threads back on after the call.

If you wish to build a copy of ONETEP without OMP in libsci, add this line verbatim to your LIBS line in the config file:

`-L${CRAY_LIBSCI_DIR}/CRAY/9.0/x86_64/lib -l sci_cray_mpi -l sci_cray`

for the Cray compiler, and

`-L${CRAY_LIBSCI_DIR}/GNU/9.0/x86_64/lib -l sci_gnu_mpi -l sci_gnu` for gfortran

Be warned, however, that this has not been well-tested yet and at least two QC tests fail with this set-up (see the first page of this document).