Jacek Dziedzic
J.Dziedzic@soton.ac.uk
2021.03.18, v1.6

Instructions for compiling and running ONETEP on Iridis5

1. Setting up the environment

You will need to load the correct set of modules. The provided config file is guaranteed to work with the Intel compiler v19.0.3 and Intel MPI 2018.3.222. To do this, add the following lines verbatim to your `~/.bashrc` file:

```
module load intel-compilers/19.0.3 >/dev/null 2>/dev/null
module load intel-mkl/2019.3.199   >/dev/null 2>/dev/null
module load intel-mpi/2018.3.222   >/dev/null 2>/dev/null
module load intel-tools/2019.3     >/dev/null 2>/dev/null
```

Then log out and log back in again for the changes to take effect. Check that you have the correct set of modules loaded by issuing the command `module list`. You should get something like:

```
Currently Loaded Modules:
  1) intel-compilers/19.0.3   3) intel-mpi/2018.3.222   5) intel-tools/2019.3
  2) intel-mkl/2019.3.199     4) gnuplot/5.2.2
```

which lists the above four modules in addition to any other module files you might have loaded (`gnuplot` in my instance). Make sure there are *no other compilers or MPI versions* in the list as they might interfere with the set-up.

2. Compilation

Use the config file `conf.iridis5.intel19.omp.scalapack`. Issue the command below while in the directory of your ONETEP installation:

```
make onetep ARCH=iridis5.intel19.omp.scalapack
```

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=iridis5.intel19.omp.scalapack -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 Iridis5, it would be prudent to test it by running a 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.iridis5.2021` (provided in the directory `hpc_resources/Iridis5` of your ONETEP installtion) to the `tests` directory of your ONETEP installation. Then submit it to the batch system by changing to the `tests` directory and issuing

```
sbatch qcsubmit.iridis5.2021
```

There is no need to edit this file. The tests run in the `scavenger` queue and should complete within two hours of starting. 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:

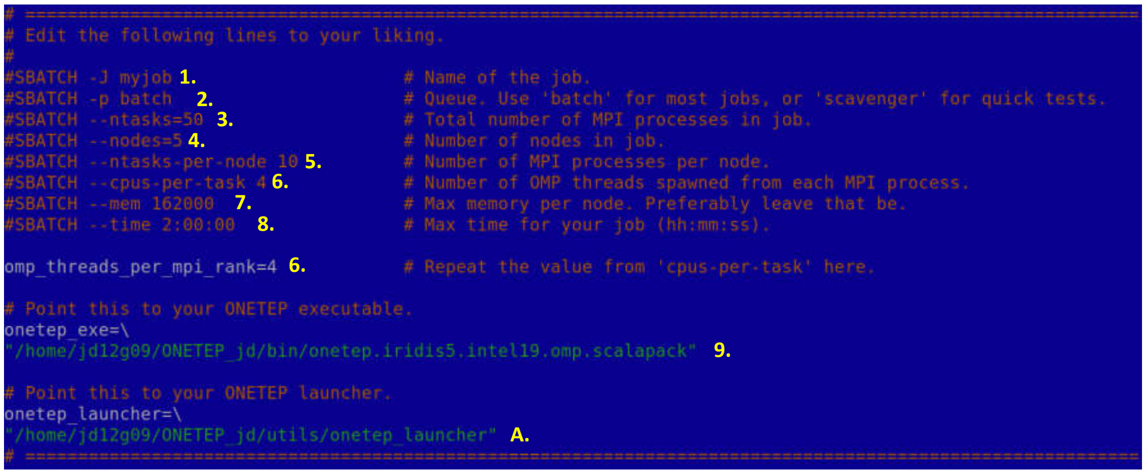
`All done. 102 out of 102 tests passed (12 warnings).`

If all tests passed – you're good to go.

4. Running

Use the provided submission script: `jobsubmit.iridis5.2021`. 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 next page). Submit it with: `sbatch  jobsubmit.iridis5.2021`. Details on how to monitor and control submitted and running jobs can be found on the Iridis5 support page.



```
# ================================================================================================
# Edit the following lines to your liking.
#
#SBATCH -J myjob 1.                    # Name of the job.
#SBATCH -p batch   2.                  # Queue. Use 'batch' for most jobs, or 'scavenger' for quick tests.
#SBATCH --ntasks=50  3.                # Total number of MPI processes in job.
#SBATCH --nodes=5 4.                   # Number of nodes in job.
#SBATCH --ntasks-per-node 10 5.        # Number of MPI processes per node.
#SBATCH --cpus-per-task 4 6.           # Number of OMP threads spawned from each MPI process.
#SBATCH --mem 162000   7.              # Max memory per node. Preferably leave that be.
#SBATCH --time 2:00:00   8.            # Max time for your job (hh:mm:ss).

omp_threads_per_mpi_rank=4  6.         # Repeat the value from 'cpus-per-task' here.

# Point this to your ONETEP executable.
onetep_exe=\
"/home/jd12g09/ONETEP_jd/bin/onetep.iridis5.intel19.omp.scalapack"  9.

# Point this to your ONETEP launcher.
onetep_launcher=\
"/home/jd12g09/ONETEP_jd/utils/onetep_launcher"  A.
# ================================================================================================
```

Things to adjust:

1. 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.
2. The name of the queue to which you're submitting your job. On Iridis5 you will generally want to use the `batch` queue, which is a general queue. Jobs larger than 16 nodes will be automatically moved to the `largejobs` queue, regardless of where you submit them. They then take substantially longer to start. Short, small test jobs (up to 2h, up to 5 nodes) can be

submitted to the `scavenger` queue. It typically permits them to start very quickly, although some users abuse this queue and you still might have to wait. Jobs ran in `scavenger` might be pre-empted (killed prematurely) should the batch system decide it's worth it. Generally – stick to `batch`.

3. The *total* number of **tasks** (MPI processes) you want to start, summed over all nodes. In typical usages you will want to run 10 MPI processes per node – then this would be ten times the number of nodes you want.
4. The *total* number of **nodes** (machines) you are asking for. Each Iridis5 node has **40 CPU cores**.
5. The number of MPI processes *per node* you want to run. I suggest using 10.
6. The number of OpenMP *threads* spawned from each MPI process. I suggest using 4. In this way each node is saturated – 10 MPI processes spawning 4 OpenMP threads each use all the 40 CPU cores. Note that this value needs to be repeated several lines later.
7. The total RAM in MB *per node* that you are asking for. I suggest leaving that value be. If you lower it, you won't be able to use all available memory. If you set it higher, the batch system might refuse to run your job at all. The total available memory on an Iridis5 node is somewhat larger, but a margin is needed for the OS and supporting software.
8. The walltime you are asking for in `hh:mm:ss`. The maximum for the batch queue is `60:00:00`. The maximum for scavenger is `02:00:00`. Don't exceed these values or your job will never start. Normally using lower values allows faster job turnaround – the scheduler is able to stick short jobs into the allocation more easily. This *does not seem to be the case on Iridis5* – there doesn't seem to be any benefit from carefully estimating your walltime.
9. Path to your ONETEP executable. Change it to the location of your binary.
10. 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 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.

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 Iridis5 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 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.

5. 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.