Jacek Dziedzic
J.Dziedzic@soton.ac.uk
2021.03.18, v1.0

Instructions for compiling and running ONETEP on Red Hat 7.9 desktop machines

These instructions should also work on Red Hat 7.7 desktop machines.

1. Setting up the environment

You will need to load the correct set of modules. We provide two config files – one for the Intel compiler v19.0.3 and one for the Intel compiler v18.0.5. Both are installed on newer machines, while the older machines might only have the latter. It is likely that the config file for v18 will also work with the Intel compiler v17.0.2, although this is not recommended.

You can issue the command

```
module avail
```

to see which one is installed on your machine. See if the list contains **`intel/2019`** and/or **`intel/2018`**.

To ensure you always have the correct module loaded, add the following line verbatim to the end of your `~/.bashrc` file:

```
module load intel/2019 >/dev/null 2>/dev/null
```

if you prefer Intel v19, or, if you'd rather use Intel v18:

```
module load intel/2018 >/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 Modulefiles:
  1) intel/2019           2) mathematica/11.0.1
```

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

In the interest of brevity, in the text that follows I will assume you are using the 2019 version, but the instructions are valid for the 2018 version too – just replace every `19` with `18`.

2. Compilation

Use the config file `conf.RH79.intel19.omp.scalapack`. Issue the command below while in the top-level directory of your ONETEP installation:

```
make onetep ARCH=RH79.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=RH79.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 your machine, 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 script `run_all_tests.RH79.2021` (provided in the directory `hpc_resources/RH79_desktops` of your ONETEP installtion) to the `tests` directory of your ONETEP installation. Then simply run it by changing to the `tests` directory and issuing

```
./run_all_tests.RH79.2021
```

There is no need to edit this file, unless you want to manually adjust the number of OMP threads the tests run on (the default is 4, which should work OK). If the script aborts immediately, something is seriously wrong and you should read the error messages that appear on the screen. If everything goes well, after a few minutes you should start seeing 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 message like

```
--- Finished running QC tests at Wed Mar 17 10:12:18 GMT 2021.
    Examine the output above to see if they passed.
```

This does not necessarily mean that all tests completed successfully. You should examine the earlier output 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

These are desktop machines, so there is no batch (queueing) system. To run a job, simply change to the directory where your input (`.dat`) file is and issue

```
mpirun -np n path_to_onetep_dir/utils/onetep_launcher -t m myinput.dat
```

In the above replace all the symbols in *italics* with actual values, and keep everything else verbatim. So, replace:

- *n* with the actual number of MPI processes on which you'd like to run your job. For example 4 or 8 are usually good ideas, unless you have only several atoms, in which case you might want to use 1 or 2.
- *path_to_onetep_dir* with the actual path to the top-level directory of your ONETEP installation. For instance `~/programs/ONETEP` or wherever your ONETEP install is.
- *m* with the actual number of OMP threads which you'd like spawned from every MPI process. In general, aim for *n***m* to coincide with the total number of CPU cores on your machine (likely 8 or 16).
- *myinput.dat* with the actual name of your input file.

There is no need for you to set the stack size (`ulimit -s`), `OMP_STACKSIZE`, or `OMP_NUM_THREADS`. Also, **do not** add any `threads_` keywords to your input file. This is all handled by the `onetep_launcher` script, really.

The above form sends all output to the screen and blocks your terminal until the job is completed. For anything serious you might want to instead send the output to a file and to run the job in the background. To do this, issue (as one line):

```
mpirun -np n path_to_onetep_dir/utils/onetep_launcher -t m
myinput.dat >myinput.out 2>myinput.err </dev/null &
```

Of course replace *myinput* with the actual name of your input, like before. Keep `</dev/null &` verbatim.

This will send the ONETEP output to *myinput.out* and error messages to *myinput.err*. Your job will run in the background, freeing your terminal. If you log out, the job will likely be killed after a while. To prevent this from happening, issue:

```
disown -ah
```

This will let you log out. You do not need to do this step if you do not plan on logging out while the job runs.

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, `mpirun`, MPI or the Fortran RTL.
- **myinput.error_message**: if ONETEP fails gracefully, this will contain an error message.