Starting with ONETEP

Rebecca J. Clements

November 19, 2019

Obtaining a copy of ONETEP

First, you must create a BitBucket account with your University of Southampton email address. Once you have been added to the Contributors' group and have access to the official ONETEP repository (`https://bitbucket.org/onetep/onetep`), read the relevant sections of the contributing document, found at `https://bitbucket.org/onetep/onetep/src/master/CONTRIBUTING.markdown`. Use this to create your own private fork of the main ONETEP repository. Here, you will also find details of how to contribute any developments you make to the ONETEP code in the future.

You can now make a copy of your ONETEP repository on any machine, using *git clone*. Make sure you have Git on your computer. From your chosen directory, use one of the following commands, which can also be copied from the website, by clicking *clone* at the top of your repository webpage:

```
git clone https://rjc1g14@bitbucket.org/<username>/<repo-name>.git
git clone git@bitbucket.org:<username>/<repo-name>.git
```

and type in your password.

You need to make sure you have set up an environment with the correct runtime and compilation variables. See the configuration scripts for this, in $/<repo-name>/config/$. The configuration file of choice depends on the system you are running on, e.g. $conf.iridis5.intel18.omp.scalapack$ is the latest version for compiling on IRIDIS5, and $conf.RH7.intel18.omp.scalapack$ is preferable for compiling on the RHEL7 linux desktop computers in the chemistry office. Load the required modules for your compiler in the terminal and set the environment details specified. For example,

```
# Load your versions of the following modules for the compiler, Intel MPI
    and MKL
# module load ifort
# module load iimpi
# module load imkl
```

```
# e.g.

# for Iridis:
module load intel-compilers/18.0.3 >/dev/null 2>/dev/null
module load intel-mkl/2018.1.163 >/dev/null 2>/dev/null
module load intel-mpi/2018.1.163 >/dev/null 2>/dev/null

# for Desktop PC:
module load intel/2017u2 # Intel Parallel Studio XE 2017
# When using the Intel compilers on the desktop, you must first source
    compilervars.sh, see:
# https://software.intel.com/en-us/articles/setting-up-the-build-
    environment-for-using-intel-c-or-fortran-compilers
source /local/software/intel/2017u2/compilers_and_libraries_2017.2.174/linux
    /bin/compilervars.sh -arch intel64 -platform linux

# Set the environment details
ulimit -s unlimited
ulimit -c unlimited
export OMP_STACKSIZE=64M
# Set a sensible default number of OMP threads for running on the login
    nodes (optional)
export OMP_NUM_THREADS=4
```

Add the lines to ∼/*.bashrc* in order to automatically execute every time you open the terminal. Make sure to use the latest configuration file and compiler.

To update your ONETEP version, you must use *git fetch* and merge the update into your local repository's master branch. First you should check the URLs you have saved with the command:

```
git remote -v
```

You should have the URLs to the main ONETEP repository, named *bitbucket_master* or *upstream* or something else, and your private repository, likely to be listed under *origin* or another name. If you do not, you must save these URL using *git remote add*. Then you can download the main repository update and merge into your local branch:

```
git remote add <name> <repo URL> -f
git fetch <name of main repository given to the onetep/onetep url>
git merge <same name as above>/master
```

Compiling

To compile, use the following commands to execute the Makefile in the main directory of your repository. The ONETEP code is made up of all the modules

found in the */src/* directory, with the main program called *onetep.F*90. The first command below compiles all of these modules into object files, and then combines them all into one executable. You must be in the main directory for this. You must replace the architecture, here, with the name of the configuration file you are using. The second command compiles in debug mode, which enables extra information to be displayed in the output when running ONETEP. This makes it easier to identify the reason for any errors in the running of the code, in the case when the error messages are not intuitive. If this is so, these should be addressed. Debug mode shows when each subroutine and function is called in the calculation, for example. The final command allows you to clear the compiled files. This should not generally be necessary. You should only need to recompile when updating the ONETEP code version between projects.

```
make onetep ARCH=RH7.intel17.omp.scalapack
make debug ARCH=...
make clean ARCH=...
```

Testing for successful setup

You need to QC (quality control) test the code and make sure it runs correctly in parallel with MPI and OMP before running any of your own calculations. There are instructions about this with the code that you should read. See $/ < repo - name > /tests/$ for the tests directory. You must run all tests by following the *README.test* document. This uses the Python script from the */testcode/* directory. Make sure to specify your chosen configuration file in place of $\$ARCH$ in the $userconfig$ file. This runs the script on the login nodes. To submit the tests to the job queue on a cluster computer such as IRIDIS5, there are submission scripts available for this. See below for running on IRIDIS. If any QC tests fail, speak to the group.

Running ONETEP

Once you have compiled ONETEP and all QC tests pass, it is time to run your calculations. The executable can be found in the */bin/* directory, although it is recommended that you use *onetep_launcher* which runs this executable $< repo - name > /utils/onetep\_launcher$, as it usually sets up the correct environment.

Input files

Go to ONETEP's website, `onetep.org`. Here you will find the Tutorials section, which introduces running various levels of ONETEP calculations. Take a look at some of the input files at the bottom of the page. Input files in ONETEP have the *.dat* file ending. If you download any as *.txt* files, you will

need to change the ending.

Data input files contain keywords, instructing ONETEP on what calculations to run, and to set the parameters needed to run them. Check out the keywords listed in this file on the webpage `onetep.org/Main/Keywords` to see what they mean. If not specified, most of them have default settings, as listed on the webpage.

The keywords come in different types; logical, block, integer, real, text, and physical. Logical comprises of True or False. Block indicates more than one line specifying coordinates. Integer and real are numbers. Text is a string of characters. Physical refers to physical variables, which come with units such as Angstrom, bohr, Joule, Hartree, etc.

The important ones to get started are:

- Task; to choose what main calculation you would like ONETEP to perform, e.g. a single point energy or geometry optimisation. You can run a properties calculation this way, using output files generated from a single point energy calculation or using the task singlepoint and the separate keyword do_properties set to T.
- XC functional; to choose how to approximate the xc functional term in the Kohn Sham DFT equation
- Simulation box dimensions *%block lattice_cart*
- The atomic positions in Cartesian coordinates, *%block positions_abs*

As can be seen in the example input files, all block keywords must end with an *endblock*. All coordinates are in atomic units. They can be specified in Angstroms i.e.

```
%block positions_abs
ang
C 16.521413 15.320039 23.535776
O 16.498729 15.308934 24.717249
...
%endblock positions_abs
```

The *_format* keywords specify the output type. The *species* and *species_pot* blocks detail the parameters of the atoms. Non-orthogonal Generalised Wannier Functions (NGWFs) are used to model the atoms instead of conventional atomic orbitals. Under the keyword species, the name we give to each atom of the system is given first, followed by the element of the atom, the number of electrons, the number of NGWFs used to model the atom (typically 4 for any atom, and 1 for hydrogen) and the radius of each NGWF (typically between 8.0 and 10.0 for an accurate calculation).

```
e.g. C C 6 4 8.0
```

The *species_pot* specifies the path location of the pseudopotential used for each element of the system. *.recpot* files exclude core electrons, the standard for ONETEP. *.paw* files include core electrons. Some of these can be found in your repository's pseudo directory. For more, please ask Chris.

To continue a calculation if it has run out of computation time, use the keywords below. The original input must have the *write* keywords, but no *read* keywords because the files aren't available to read at this stage. Any continuing input files must include all four. If the input file name isn't changed upon continuation, the output file will overwrite with only the new energy iterations performed, so make sure to back up files before continuing.

```
write_denskern T
write_tightbox_ngwfs T
read_denskern T
read_tightbox_ngwfs F
```

On Desktop

Once the modules have been loaded and the code has been compiled, ONETEP can be run in parallel, across a few processes e.g. less than 5 for a few atoms, using the command below. Try varying the number of processes. The & sends the job into the background so that the terminal can still be used.

```
mpirun -np <number of processes> <path to your repo>/
    utils/onetep_launcher <input>.dat > <output>.out &
```

To check the jobs, type *jobs*.

On IRIDIS 5

Ask the group for the submission script for IRIDIS. There is one for running the QC tests and one for running your own calculations. A few variables will need changing in this script:

- Change the number of cores relevant to the size of the system. e.g.

  ```
  ... ntasks=40
      nodes=1
      ntasks-per-node 40
      cpus-per-task <any number>
  ```

  $ntasks = nodes \times ntasks - per - node$. There are 40 tasks, or processors, available per node. Less than 5 processors are needed for a system containing a few atoms, whereas 200 is need for 2000 atoms, and the above is

adequate for tens of atoms. When choosing the number of processes, use a multiple of 40 for efficiency or parallelism (MPI). The $cpus-per-task$ allows for OMP threading and may be preferable.

- Change the executable path to $< repo-name > /utils/onetep\_launcher$
- Change the input filename to your input

The submission script must load the latest compilers on IRIDIS. Use the the IRIDIS5 pages for help with submitting your jobs, `https://southampton.ac.uk/isolutions/staff/iridis.page`. To submit the submission script to the job queue, use:

```
sbatch <name of script>.slurm
```

The input file must be in the same directory as the submission script unless you have specified its path. It is recommended that you submit all jobs from your Scratch account, $/scratch/ < username >$, which has more storage, rather than your home directory, although your Scratch account is not backed up by the university.