Lab 2
Supplemental Information

Edward Linscott and Miki Bonacci
Paul Scherrer Institute
edward.linscott@psi.ch, miki.bonacci@psi.ch

March 21, 2025

Contents

1 Quantum ESPRESSO **2**
1.1 Getting started . . . 2
1.1.1 The input file . . . 2
1.1.2 Visualizing crystals . . . 6
1.1.3 Running a calculation and obtaining the total energy . . . 6
1.2 Running PWscf efficiently . . . 7
1.2.1 Using bash scripts . . . 7
1.2.2 Using the virtual machine with multiple processors . . . 8

2 Convergence **9**
2.1 Robustly defining convergence . . . 9
2.2 Plane-wave expansion . . . 10
2.3 **k**-point sampling of the Brillouin Zone . . . 10

3 FAQs **11**

4 Using supercomputers **12**
4.1 Connecting to `helvetios` . . . 12
4.2 The home and scratch directories . . . 13
4.3 Copying files onto the supercomputer . . . 13
4.4 Scheduling on supercomputers with SLURM . . . 13
4.5 Submitting a job . . . 14
4.6 Fetching files from the supercomputer . . . 15

5 Miscellaneous **15**
5.1 Optional QE installation on personal workstation (not recommended) . . . 15
5.2 Other electronic structure codes . . . 16

1 Quantum ESPRESSO

In this lab we will use the QUANTUM ESPRESSO package [1], which is one of the most popular open-source softwares in the condensed matter community. QUANTUM ESPRESSO is an integrated suite of computer codes for electronic structure calculations and materials modeling at the nanoscale. ESPRESSO stands for *opEn Source Package for Research in Electronic Structure, Simulation, and Optimization*. It is based on density functional theory (DFT), plane waves, and pseudopotentials. It allows one to model structural and electronic properties of materials, vibrational properties (phonons), spectroscopic properties (absorption spectra, electron energy loss spectra, etc.), perform molecular dynamics (Born-Oppenheimer, Car-Parrinello, Langevin) and do many other things.

The PWscf program (**pw.x**) is one of the *core* components of QUANTUM ESPRESSO and it is based on DFT for calculations of the *ground state properties* of materials, *e.g.* total energy, forces, stress tensor, etc. The rest of this handout will demonstrate how to compute the total energy of a bulk calcium oxide(CaO) using the PWscf program.

There are many tutorials about QUANTUM ESPRESSO online.

1.1 Getting started

The environment related to Lab 2 will be the same as for Lab 1. We will be using the same QUANTUM MOBILE Virtual Machine, on which all the programs/tools required have already been installed. Remember to use the same shared folder settings as in for Lab 1 and, especially if you are using a workstation from the lab, save your work at the end of a session.

1.1.1 The input file

Among the files provided for this lab, `CaO_primitive.scf.in` contains the input information needed to compute the total energy per unit cell of calcium oxide using PWscf. If you view the file, it will look like this:

```
&control
    calculation = 'scf'
    restart_mode = 'from_scratch'
    prefix = 'CaO'
    tstress = .true.
    tprnfor = .true.
    pseudo_dir = './pseudopotentials/'
    outdir = './tmp/'
/
&system
    ibrav = 2
    celldm(1) = 9.0
    nat = 2
    ntyp = 2
    ecutwfc = 20.0
```

```
/
&electrons
    diagonalization = 'david'
    mixing_mode = 'plain'
    mixing_beta = 0.7
    conv_thr = 1.0d-8
/
ATOMIC_SPECIES
    Ca  1.0    Ca_pseudo_dojo_v0.4.1.upf
    O   1.0    O_pseudo_dojo_v0.4.1.upf
ATOMIC_POSITIONS alat
    Ca 0.00 0.00 0.00
    O  0.50 0.00 0.00
K_POINTS automatic
    4 4 4 0 0 0
```

All the input parameters for `pw.x` are described here, but the most relevant lines for the calculations related to this lab are the following:

- The line

  ```
      calculation = 'scf'
  ```

  indicates that this is a self-consistent calculation (SCF) to find the total energy of the system. Other options can be found using the link above.

- The line

  ```
      prefix = 'CaO'
  ```

  is a tag the code uses to identify this calculation.

- The line

  ```
      tstress = .true.
  ```

  indicates that the stress tensor will be computed.

- The line

  ```
      tprnfor = .true.
  ```

  indicates that the stress tensor will be computed. indicates that forces will be computed.

- The line

  ```
      pseudo_dir = './pseudopotentials/'
  ```

indicates the path to the place, where the pseudopotential file is located. We have provided the necessary pseudopotentials (`Ca_pseudo_dojo_v0.4.1.upf` and `O_pseudo_dojo_v0.4.1.upf`) within the `pseudopotentials` subdirectory. Make sure that the path to them is correct in your input file.

- The line

```
outdir = './tmp/'
```

indicates the path to the directory, where all temporary files of the code (such as wavefunctions, eigenvalues, etc.) will be written during the execution.

Note: If you give the same path for `outdir` for two different calculations with the same `prefix`, they will overwrite the temporary files and the calculation will crash. When you run several calculations at the same time, remember to give them different `prefix`, or you can use different temporary directories.

- The line

```
ibrav = 2
```

indicates the Bravais-lattice index. In this case `ibrav = 2` means that we are dealing with the face-centered cubic (fcc) primitive unit cell. For a list of all of the conventions, see here.

- The line

```
celldm(1) = 9.0
```

is the value of the lattice parameter in Bohr radii.

- The line

```
nat = 2
```

indicates the **n**umber of **at**oms in the unit cell.

- The line

```
ntyp = 2
```

indicates the **n**umber of **typ**es of atoms in the unit cell.

- The line

```
ecutwfc = 20.0
```

indicates the value of the kinetic-energy cutoff in Rydberg units. This value determines how many plane waves will be used in the expansion of Kohn-Sham wavefunctions during the iterative solution of the Kohn-Sham equations.

- The lines

```
    mixing_beta = 0.7
    conv_thr = 1.0d-8
```

are the parameters which specify the numerical details for the iterative solution of the Kohn-Sham equation. The final result (e.g. the total energy) must not depend on the value of these parameters. These default values were optimized (for all systems, not only CaO) and should not be changed. However, there are cases when one has to change these parameters. In this lab, there will be no need to modify these parameters.

- The line

```
    conv_thr = 1.0d-8
```

indicates the convergence threshold in Rydberg units during the iterative solution of the Kohn-Sham equation. When this threshold is reached, the calculation will stop.

- The lines

```
ATOMIC_SPECIES
    Ca  1.0    Ca_pseudo_dojo_v0.4.1.upf
    O   1.0    O_pseudo_dojo_v0.4.1.upf
```

indicate the label of the atom, its mass[1], and the name of the pseudopotential file.

- The lines

```
ATOMIC_POSITIONS alat
    Ca 0.00 0.00 0.00
    O  0.50 0.00 0.00
```

indicate the atomic positions in the unit cell in units of the the lattice parameter (`celldm(1)`).

- The lines

```
K_POINTS automatic
    4 4 4 0 0 0
```

indicate the **k** points sampling of the Brillouin zone. In this case, the automatic $4\times4\times4$ **k** point sampling will be used using the Monkhorst-Pack procedure [2]. The last three numbers indicate that no shift with respect to the center of the Brillouin zone will be made.

[1]Technically this should be equal to the atomic mass of the element, but because we're not performing molecular dynamics we've just used a dummy value of `1.0`

1.1.2 Visualizing crystals

Before you start your first calculation, let us introduce a useful visualization tool called XCrySDen. XCrySDen is a crystalline and molecular structure visualization program that allows us to visualize our structure and measure its certain properties such as inter atomic distances and angles. In the test folder, try typing `xcrysden --pwi CaO.scf.in` which launches the XCrySDen program, and tells to it that we want to visualize a PWscf input (`--pwi`) and the name of the file is `CaO.scf.in`. The program will read the above input file we have seen and visualize it.

Some useful features are:

- Dimensions pop-up: The first thing you see when you launch the program is to specify if you would like to visualize a 0D, 1D, or 2D structure. In this exercise we are visualizing a 3D crystal so choose the ‘‘`do not reduce dimensionality`’’ option and click `OK`.
- Display tab on top have many features. Choose to display the crystal cell. You can click and rotate the cell and visualize it from different angles. Experiment with it. Although trivial for this example, visualization tools can be very useful for complicated systems.
- Go to `Modify` tab and further down into `Number of units drawn`. Play with the numbers there to get a feel of your crystal.
- `Atoms Info, Distance, Angle, Dihedral` tabs below allow you to get info on the atoms or measure distances/angles between atoms.

1.1.3 Running a calculation and obtaining the total energy

Now you have seen the crystal structure of bulk CaO, and we are ready to run our first first-principles calculation with PWscf. Close the XCrySDen program and just type `pw.x < CaO.scf.in > CaO.scf.out`. This tells the shell to launch the executable `pw.x`, pipe the file `CaO.scf.in` to its `stdin` and pipe its `stdout` to the file `CaO.scf.out`). After a few seconds (if everything goes well) you should have in your directory a new file called `CaO.scf.out`. This is the output file of your calculation.

If you view the file `CaO.scf.out`[2], you will find that in the beginning of the file there is a lot of information which summarizes the calculation, such as number of atoms, unit cell size, number of plane waves etc. Then you can follow the output file and see how at each iteration the code finds a solution with a lower energy. And close to the end, a line containing something like

```
!    total energy              =    -115.70439474 Ry
```

will appear. This is the final result from the program for the total energy of the unit cell. Note that the final occurrence of “total energy” will have an exclamation mark next to it, as something to make this line easy to find in a long output file, for example by typing, on the terminal, the following:

[2]To view this file, run `less CaO.scf.out` or `vi CaO.scf.out`, or use any other viewer you are comfortable with

```
$ grep ! *.out
```

which will search all files terminated by `.out` in the current directory, looking for lines that have an exclamation mark in them.

There is other useful information in the output file. View it again and go to the end. For example, the output file shows how much CPU and WALL time the PWscf program spends for various parts of the calculation:

```
     init_run     :      0.08s CPU      0.08s WALL (       1 calls)
     electrons    :      0.24s CPU      0.24s WALL (       1 calls)
     forces       :      0.01s CPU      0.01s WALL (       1 calls)
     stress       :      0.02s CPU      0.02s WALL (       1 calls)
     .
     .
     .
     PWSCF        :      0.47s CPU      0.48s WALL
```

As the lectures progress you will get more familiar with the output format and all the information it contains.

1.2 Running PWscf efficiently

1.2.1 Using bash scripts

In order to check the numerical convergence of your calculations with respect to kinetic-energy cutoff and the number of **k** points, you will have to run PWscf multiple times with different values for those parameters. Of course you may create a new input file for every set of parameters, but there are faster, automatized ways of doing this. One such approach is using `bash` scripts.

You can either write a bash script by yourself or use the one provided. If you open the provided script, you will see that there are three lists of input parameters that will be created in order:

```
LISTA="10.5"
LISTECUT="20 30"
LISTK="2 4"
```

The bash script will loop over these values to perform (in this case) four different calculations automatically, without having to change the input files by hand.

You can run the script by doing

```
$ ./script.sh
```

or, if you want to do something else while you wait for the calculations to finish:

```
$ nohup ./script.sh &
```

(the `&` symbol at the end of the line allows you to keep using the terminal without having to wait for the work to finish, and the `nohup` part allows the program to keep running even if you close the terminal in the computer).

At the end you will find several output files in your output directory. Go in the output directory and type `ls -l` to get a list of the produced files.

In order to solve the problems you will have to modify the above script accordingly to do different calculations you need.

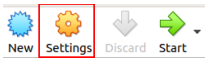
1.2.2 Using the virtual machine with multiple processors

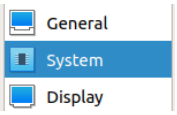
For most of the exercises that you will have to do, executing `pw.x` with only one core will be enough to finish them in a matter of a few minutes. Nevertheless, the last exercises will be more demanding than the others, as it requires to run relax calculations on a supercell. These will take about 100 times longer than the `scf` calculations on the primitive cell you did during the earlier exercises.

If this length of time seems too unreasonable there are ways to speed it up:

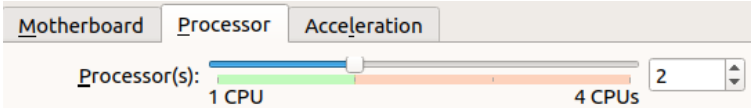
- Run the calculation on the supercomputer (see 4): it should take between 10-20 minutes using 1 node with 28 cores
- If you have a fast computer, you can increase the number of CPUs assigned to the VM.

In order to increase the number of CPUs assigned to the VM you have to:

- Shutdown the VM (full shut-down).
- Select the VM in the VBox interface
- Click on 'Settings' 

- Click on 'System' in the left sidebar 

- Click on 'Processor' in the top bar and change the number of CPUs (I would recommend half of the maximum value or before you reach the red part of the slider bar). If you can't change this it means the VM was not totally turned off.



- Press 'Ok' to save the changes

In order to make use of having multiple processors you will need to launch the calculation using `mpirun`:

- From the command line:

```
$ mpirun pw.x < input_name > output_name
```

- From a script: locate the line that sets the executable

  ```
  PW_LAUNCH='pw.x'
  ```

  And change it to:

  ```
  PW_LAUNCH='mpirun pw.x'
  ```

  Depending on the system, the command `mpirun` could be different (e.g. `srun` on the `helvetios` supercomputer).

With a small number of CPUs, the improvement you should experience in run time should be linear (eg. twice the CPUs will roughly halve the time). This will be more true of long calculations, for which the serial part of the code execution will be negligible compared to the parallel part. For shorter calculations the scaling will be different.

2 Convergence

Any PWscf calculation (e.g. the total energy in this case) must be checked for a convergence with respect to:

- kinetic-energy cutoff `ecutwfc`;
- **k** points sampling of the Brillouin zone;
- other parameters where relevant (e.g. smearing for metals; not relevant in this exercise)

Below you will find some basic concepts in order to understand convergence better. Good explanations can also be found in Refs. [3] and [4].

2.1 Robustly defining convergence

A quantity O is converged with respect to a parameter X to within some threshold T when $|O(X) - O_{\text{true}}| < T$. Here, O_{true} is the value of the quantity O in the absence of any approximations introduced by the parameter X. (For example, for convergence with respect to the cutoff energy, this would be the value of O if we never truncated the plane-wave expansion of the wave functions). Of course, we can't actually calculate O_{true}; in practice one instead takes O at the highest cutoff (or k-grid, or equivalent) that you calculated — provided that O as a function of X appears well-converged.

Note that this is very different from the alternative criterion $|O(X_{n+1}) - O(X_n)| < T$. Can you think why this is, and why the former definition is correct and not the latter?

Also note that it is also possible to get "false" convergence, where (for example) your energy at a $2 \times 2 \times 2$ k-grid may be the same as the energy at a $8 \times 8 \times 8$ k-grid, but the energy at a $4 \times 4 \times 4$ might be very different from both of these. In this case, you aren't really converged at a $2 \times 2 \times 2$ k-grid — the energies agree out of sheer luck and we wouldn't expect the $2 \times 2 \times 2$ calculation to consistently yield other parameters that agree well with an $8 \times 8 \times 8$ calculation.

2.2 Plane-wave expansion

When performing calculations with a plane-wave code, always remember that we are dealing with infinite systems using periodic boundary conditions. This means that we can use the Bloch theorem to help us to solve the (Schrödinger-like) Kohn-Sham equation. The Bloch theorem states that the wavefunction can be written in the form:

$$\psi_{n,\mathbf{k}}(\mathbf{r}) = e^{i\mathbf{k}\cdot\mathbf{r}} u_{n,\mathbf{k}}(\mathbf{r}), \tag{1}$$

where n is the band index, $\mathbf{k}$ is the point in the Brillouin zone, $e^{i\mathbf{k}\cdot\mathbf{r}}$ is the phase, and $u_{n,\mathbf{k}}(\mathbf{r})$ is the *lattice-periodic* part of the wavefunction which can be written as

$$u_{n,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{G}} c_{n,\mathbf{k}+\mathbf{G}}\, e^{i\mathbf{G}\cdot\mathbf{r}}, \tag{2}$$

$$u_{n,\mathbf{k}}(\mathbf{r}+\mathbf{R}) \equiv u_{n,\mathbf{k}}(\mathbf{r}), \tag{3}$$

where $\mathbf{G}$ are the reciprocal lattice vectors, $\mathbf{R}$ is the vector of the unit cell in real space, and $c_{n,\mathbf{k}+\mathbf{G}}$ are the coefficients of the expansion. The sum in Eq. (2) runs, in principle, over infinite number of the reciprocal lattice vectors $\mathbf{G}$. In practice, however, one has to truncate the sum (*plane wave cutoff*).

The basis functions, which we use to make an expansion in Eq. (2), are *plane waves*. They are called "plane waves", because they are surfaces of constant phase and are parallel planes perpendicular to the direction of propagation. The values of $\mathbf{G}$ are integer multiples of the three reciprocal-space lattice vectors, and hence they are compatible with the periodic boundary conditions of our direct lattice.

In actual calculations, we have to limit the plane-wave expansion by specifying the plane wave cutoff, which must be given in energy units (such as Rydberg or eV):

$$\frac{\hbar^2}{2m}|\mathbf{k}+\mathbf{G}|^2 \leq E_{\text{cut}}, \quad E_{\text{cut}} = \frac{\hbar^2}{2m} G_{\text{cut}}^2, \tag{4}$$

where $\hbar$ is the Planck constant, m is the electron mass, E_{cut} is the cutoff kinetic-energy, and G_{cut} is the cutoff reciprocal lattice vector. Given Eq. (4), Eqs. (1) and (2) read:

$$\psi_{n,\mathbf{k}}(\mathbf{r}) = \sum_{|\mathbf{k}+\mathbf{G}|\leq G_{\text{cut}}} c_{n,\mathbf{k}+\mathbf{G}}\, e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}. \tag{5}$$

All of the above have important ramifications for the behaviour of observables when we perform convergence with respect to the plane-wave energy cutoff.

2.3 k-point sampling of the Brillouin Zone

In order to solve the Kohn-Sham equations, one must calculate various integrals in the reciprocal space over $\mathbf{k}$ vectors ($\int d\mathbf{k}(...)$). In practice, the integrals over $\mathbf{k}$ vectors are replaced by the summations over descrite $\mathbf{k}$ points ($\sum_{\mathbf{k}}(...)$). Ideally, the number of $\mathbf{k}$ points must be infinite, but in practice one has to use a finite number and make tests for convergence.

In the output file of your calculation (`CaO.scf.out`) you will find something like this:

```
number of k points=     8
cart. coord. in units 2pi/alat
    k(    1) = (   0.0000000   0.0000000   0.0000000), wk =   0.0312500
    k(    2) = (  -0.2500000   0.2500000  -0.2500000), wk =   0.2500000
    k(    3) = (   0.5000000  -0.5000000   0.5000000), wk =   0.1250000
    k(    4) = (   0.0000000   0.5000000   0.0000000), wk =   0.1875000
    k(    5) = (   0.7500000  -0.2500000   0.7500000), wk =   0.7500000
    k(    6) = (   0.5000000   0.0000000   0.5000000), wk =   0.3750000
    k(    7) = (   0.0000000  -1.0000000   0.0000000), wk =   0.0937500
    k(    8) = (  -0.5000000  -1.0000000   0.0000000), wk =   0.1875000
```

This is a list of the irreducible (non-equivalent) **k** points that were used to sample the irreducible Brillouin zone. Remember, we specified in the input the uniform Monkhorst-Pack **k** points sampling of the Brillouin zone: $4 \times 4 \times 4 = 64$, but we see here only 8 **k** points. This is so, because some of these 64 **k** points are equivalent due to symmetry (give the same Kohn-Sham energy). Consider an example of a cube: it has 8 identical corners, so if the original **k** point mesh includes 8 corner points, it is computationally more convenient (to save CPU time) to calculate the energy using one of these corner points and weighing it with the appropriate multiplicity of that particular **k** point (the corner point of a cube has the multiplicity 8).

Different **k** points in the original 64 point mesh have different multiplicities in the Brillouin zone of the crystal. The `pw.x` code, using the symmetry, finds these multiplicities and lists the non-equivalent **k** points and the corresponding weights (`wk`). The weights add up to 2, because we perform unpolarized calculation, and, hence, every Kohn-Sham state is occupied by two electrons due to the Pauli principle (spin degeneracy).

The list of **k** points will be different, if you use a different **k** point mesh ($6 \times 6 \times 6$, $8 \times 8 \times 8$, etc.).

All of the above can tell us about the expected behaviour of observables when we perform convergence with respect to the **k** points sampling of the Brillouin zone.

3 FAQs

How precisely do I need to compute the lattice parameter? Lattice parameters are typically listed to within 0.01 Å. There are applications when higher precision is required; this is *not* one of them.

Why is the total energy higher when you displace an atom? At equilibrium the energy is the lowest. In equilibrium, the structure has a Ca atom at (0, 0, 0) and O at (0.50, 0.0, 0.0). Any displacement of atoms from these positions will increase the total energy. As a side note, the forces will give you an idea of how far you are from equilibrium (they tell you which direction the atoms "want" to move). The stresses tell you which direction the cell parameters "want" to change to reach equilibrium.

My energy versus lattice constant plot is jagged. What should I do? This indicates that the energy is not sufficiently converged to accurately predict the energy differences between the adjacent values you have chosen for the lattice parameter. Increase the

energy cutoff and/or increase the spacing between adjacent values of the lattice parameter. (Make sure you don't change the lattice parameter to unphysically large/small values!)

How do I kill a running job on a local machine? If you are running the calculation in the foreground (i.e. without `&` at the end of your command), `control-C` kills the job.

If you are running the calculation in the background (i.e. with `&` at the end of your command), type `top` in a separate window. This will provide a table of processes that are running on your machine. Each process has an ID number listed in the PID column. Type `k` (for "kill"), enter the PID of the process you want to kill, and then hit enter to kill that process.

What exchange-correlation functional are we using? Which functional QUANTUM ESPRESSO uses is determined by the pseudopotentials you provide it with. In this lab, the pseudopotentials correspond to PBEsol.

Why are my energies are really different from Lab 1? Remember that absolute energies do not have any physical meaning. Only the energy differences (other way speaking, the relative positions of the energy levels) are meaninful. Also, the reference energies are different. In lab 1, the reference energy (when atoms are infinitely far apart) is 0. In lab 2, this is not the case. The absolute energies can shift a lot, depending on which reference energies you take.

Why are we using k point grids with the same number of points per direction? For this material, we take the same number of **k** points per direction, because the three lattice directions are equivalent. This is not always true. The most common **k** point meshes are those that sample the reciprocal space evenly in all directions. Thus, for a tetragonal cell (with $a = b$, $c = 2a$, and all angles 90 degrees) we might take a $8 \times 8 \times 4$ mesh. Try to think why this is so. (Note that if a lattice vector is longer in real space, it is shorter in the reciprocal space).

4 Using supercomputers

Above we have shown how to run your calculations on the virtual box/local machine. Your virtual box has access to only one processor, but you can speed up your calculations by using the High Performance Computing (HPC) systems, such as the one of EPFL called `helvetios`. Here we demonstrate how to prepare a bash script for `helvetios`, and run the same calculations as above.

4.1 Connecting to `helvetios`

First, login to `helvetios` via a terminal. The login command is:

```
$ ssh your_user_name@helvetios.hpc.epfl.ch
```

you will be asked for your password (same as GASPAR):

```
Password:
```

4.2 The home and scratch directories

Once successfully logged in, you will be at your `HOME` directory on the cluster. You can see the current path by typing

```
$ pwd
```

and you will see

```
/home/your_user_name
```

This `HOME` directory is only able to read from the computing nodes. The `HOME` directory is a good place to store important files, because it is backed up regularly, but it is never a good place to run calculations. To run calculations you need to go to place on the HPC system where you are allowed to execute the `pw.x` code, write the PWscf temporary files etc. For the sake of computing time, all this should happen on a place with very fast communication. Every supercomputer has a place like this, and most often they are called the *scratch areas*. Let us go to our scratch area on `helvetios`:

```
$ cd /scratch/your_user_name/
```

4.3 Copying files onto the supercomputer

In order to move files onto the supercomputer, we use the command `scp`. For example, if you are in the directory that contains the folder `LAB2` then you can copy it onto `helvetios` by running

```
scp -r LAB2 your_user_name@helvetios.hpc.epfl.ch:/scratch/your_user_name
```

Alternatively, you can use `wget` to download files directly off the internet onto `helvetios`:

```
wget <link_to_download>
```

The appropriate download link can usually be obtained by right-clicking on a link in a browser and selecting `Copy Link Location`.

4.4 Scheduling on supercomputers with SLURM

In a HPC system there are usually thousands of computing units and tens or hundreds of users, and most of the time there are more calculations to be done than the amount of resources available. The use of the processors is managed by a special scheduler system (on `helvetios` it is called SLURM). Users (you) prepare job files and submit to this scheduler. Then scheduler places your jobs in the queue according to how many resources your job asks for, and how long your calculation is expected to run for, or sometimes only due to the permissions you are given on that HPC system. Then, in that queue, a priority number is assigned to your job, much like getting a ticket while waiting in line at the post office. When it is your job's turn, SLURM allows it and your job starts running. You can submit several jobs simultaneously (**Important note:** remember to take care to use different `prefix` and

`outdir` for different jobs in order not to have problems with mixing temporary files during runs).

Amongst the files we distributed to you there is the slurm submission script `slurm_script.sh`. If you open it you will see that the main part of the bash script is similar to the one used to run on the single CPU computer. The differences are mostly at the top of the file:

```
#!/usr/bin/env bash
#SBATCH --nodes 1
#SBATCH --ntasks 18
#SBATCH --cpus-per-task 1
#SBATCH --time=00:30:00
#SBATCH --mem=30000

# Load modules
module purge
module load intel
module load intel-oneapi-mpi
module load quantum-espresso
```

These lines tell the scheduler how to setup the number of CPUs, what the maximum run time of the calculation should be, which libraries need to be loaded etc.

If you wish to test the supercomputer with simple exercises (all of them but the last one), you can even reduce the amount of resources requested like this:

```
#!/usr/bin/env bash
#SBATCH --nodes 1
#SBATCH --ntasks 8
#SBATCH --cpus-per-task 1
#SBATCH --time=00:10:00
```

Note that requesting only a small amount of time (e.g. 10 to 30 minutes) should significantly reduce the amount of time the job will wait in the queue.

The one final important difference is how we call Quantum ESPRESSO: instead of `mpirun -n <number of procs>` we now instead use `srun`.

Feel free to change the body of the script so that it runs precisely what you want it to.

4.5 Submitting a job

Now that our submission script is ready, we are ready to submit it. While logged in to `helvetios`, simply run

```
$ sbatch slurm_script.sh
```

To see the status of your jobs, you can type:

```
$ squeue -u $USER
```

This will display a table of your queued and running jobs. A one or two-letter status code will indicate the status of each job (e.g. `R` = "running"; `PD` = "pending/queueing"). Completed jobs do *not* appear in this table.

To kill a job before it has finished, use the command

```
$ scancel <job_id_number>
```

4.6 Fetching files from the supercomputer

When jobs are finished, you can copy files back to your personal machine using `scp`:

```
$ scp -r username@helvetios.hpc.epfl.ch:/scratch/username/path/to/file .
```

5 Miscellaneous

5.1 Optional QE installation on personal workstation (not recommended)

In case you are using a Linux distrubution and you want to try to install Quantum ESPRESSO directly on your computer here are the steps (this process should also work for an Ubuntu subsystem on Windows 10 or on Mac using the brew package manager):

- Install the GNU C and FORTRAN compilers by running the following command for Debian based distros like Ubuntu (for other distros use the appropriate package manager):

  ```
  $ sudo apt install gcc gfortran
  ```

  (you will be prompted for your password as the `sudo` command means 'run as admin').

- From the GitHub repository of Quantum ESPRESSO download the source code

- Extract the content either by right-clicking, using the OS Graphical User Interface(GUI), or from terminal by running the command:

  ```
  $ tar -zxvf name_of_the_file_to_extract
  ```

- From inside the extracted folder run the following commands:

  ```
  $ ./configure
  $ make pw pp
  ```

- The Quantum ESPRESSO executables will have been created inside the `./bin` folder. Launch them using their full path or add this folder to the PATH environment variable:

  ```
  $ export PATH=$PATH:/path/to/qe/folder/bin
  ```

You can also add this command at the end of the file `~/.bashrc` so that it will be executed every time you open a new shell.

- Install other useful tools like XCrySden and XMGrace or GNUplot for data visualization:

```
$ sudo apt install xcrysden grace gnuplot
```

All the material required for this lab will be uploaded to Moodle. You can create a folder LAB2 inside the Shared folder on the Virtual Machine desktop from which you will work.

5.2 Other electronic structure codes

In addition to Quantum ESPRESSO, there are several popular first-principles codes, and some of them are listed below:

- ABINIT (http://www.abinit.org/). This is a DFT plane-wave pseudopotential code. It is also distributed under the GPL license.
- CP2K (https://www.cp2k.org/). This is a DFT code using the mixed Gaussian and plane-waves approaches. It is also distributed under the GPL license.
- CPMD (http://www.cpmd.org/). This is a DFT plane-wave pseudopotential code implementing Car-Parrinello molecular dynamics. It is also distributed under the GPL license.
- SIESTA (http://www.uam.es/siesta/). This is a DFT code that uses localized atomic basis sets. It is free for academics.
- VASP (http://www.vasp.at/). This is a DFT plane-wave pseudopotential code. Available with a moderate cost for academics.
- WIEN2k (http://www.wien2k.at/). This is a DFT Full-Potential Linearized Augmented Plane-Wave method (FLAPW). FLAPW is the most accurate implementation of DFT, but the slowest. Available with a small cost for academics.
- Gaussian (http://www.gaussian.com/). This is a quantum chemistry code that includes Hartree-Fock (HF) and higher-order correlated electrons approaches. Moderate cost for academics.
- Crystal (http://www.crystal.unito.it/). This is a HF and DFT code. Small cost for academics.
- Molpro (http://www.molpro.net/). This is a DFT code. Small cost for academics.

An extended list of codes: http://nomad-coe.eu/index.php?page=codes

References

[1] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.

[2] H. J. Monkhorst and J. D. Pack, "Special points for brillouin-zone integrations," *Physical review B*, vol. 13, no. 12, p. 5188, 1976.

[3] D. Sholl and J. A. Steckel, *Density functional theory: a practical introduction*. John Wiley & Sons, 2011.

[4] R. M. Martin, *Electronic structure: basic theory and practical methods*. Cambridge university press, 2004.