Thermo_pw: a FORTRAN driver for Quantum ESPRESSO routines

Andrea Dal Corso

SISSA, CNR-IOM, and MaX
Trieste (Italy)

THERMO_PW

Outline

thermo_pw: What is it?

- It is a Fortran driver of the Quantum ESPRESSO (QE) routines that allows the simplification of the calculation of selected material properties.
- It is a Fortran driver that calls `pw.x` and `ph.x` or any QE routine exploiting the image parallelization. The images can communicate in an asynchronous way so the work-load distribution can be done during the run.
- It is a set of pre-processing tools for reducing the number of information that must be provided by the user.
- It is a set of post-processing tools to produce plots directly comparable with experiments.

Why a new work-flow?

- Because the number of CPUs of parallel machines is growing rapidly and it is difficult to exploit them without increasing the size of the system.
- Many tasks require many `pw.x` or `ph.x` calculations and could exploit many CPUs, but presently the work must be coordinated by scripts.
  - A typical example are the thermodynamic properties where many phonon calculations are needed for many geometries.
  - Another example is the calculation of a frequency dependent dielectric constant, where many copies of `ph.x` could be run in parallel each one working on a different frequency.

`thermo_pw` drives these calculations through Fortran calls and simplifies the scripts necessary to calculate material properties.

Phonon parallelization: grid, images

Parallelization modes of QE:

- **G**-vectors.
- bands.
- **k**-points.

Additional parallelization of phonon:

- **q**-vectors.
- Irreducible representations.

Actually this is implemented using `grid` techniques: one **q** point per run or one `irrep` per run.
Another possibility is to use `images`. The total number of processors is split into several groups (images) each image running an independent copy of `ph.x`.

Phonon parallelization: grid, images

Problems with the grid:

- It requires complex scripts to coordinate and collect the results of different runs.

Problems with images:

- Images do not communicate among themselves, because different runs are independent.
- Load balacing is difficult.
- Final results need to be collected running `ph.x` another time.

thermo_pw

`thermo_pw` solves two of these problems:

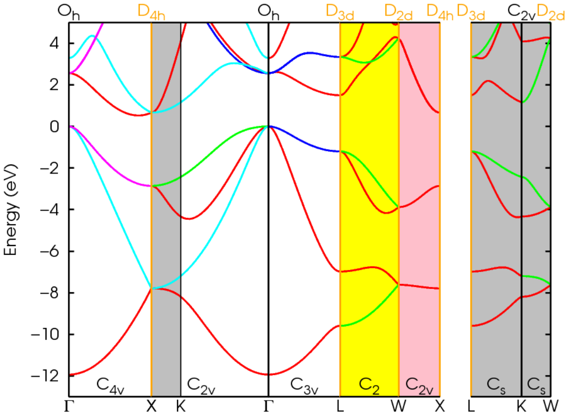
- Images can communicate through a master-slaves approach via MPI calls.
- The code can run in a synchronous and asynchronous mode. It can collect the final results automatically.
- The code can mix calls to `pw.x` and `ph.x` so that it is possible for instance to optimize the structure before calling `ph.x`, or call `ph.x` for several geometries and compute anharmonic properties.

thermo_pw as a pre-processing tool

- Automatic detection of the space group and optimization of the FFT mesh.
- Automatic generation of the **k**-point path for band structure and phonon dispersions calculation.
- Automatic reorientation of the Bravais lattice if not compatible with the symmetry analysis routine (for `ibrav=0` input).
- Automatic generation of strained lattices for elastic constants calculations.

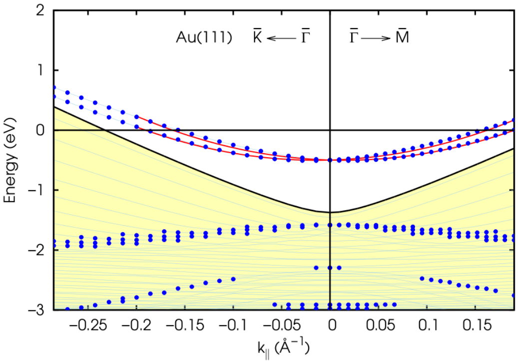
thermo_pw as a post-processing tool

Symmetry analysis of the bands and of the phonons in all the Brillouin zone, for all space groups (Total Energy poster).



thermo_pw as a post-processing tool

Computation of the projected bulk band structure and identification of surface states.



Surf. Sci. **637-638**, 106 (2015).

thermo_pw as a post-processing tool

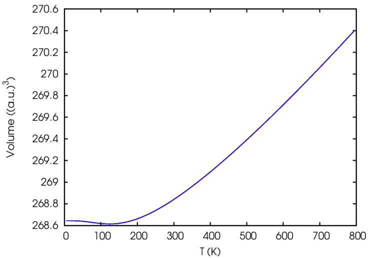


Figure: Si cell volume as a function of temperature.

Computation of harmonic and anharmonic thermodynamic properties (Total energy poster together with M. Palumbo). See also exercise at the Quantum-ESPRESSO tutorial.

... and now?

- The experience with `thermo_pw` has been very positive. The downloads of the code are incredibly high. I received some positive users' comments on its use and suggestions for the implementation of new features.
- So far `thermo_pw` has been only an experiment. It is not a production tool. Some features are still incomplete or working only for selected crystal systems.
- Now the code must be refined and cleaned, probably with less additions and more generalizations of the features already present. `thermo_pw` is becoming a real project.

Asynchronous parallelization via MPI routines

Both master and slaves compute all the tasks to do (for instance all the `irreps` and **q** points) and assign a number to each task.

Master:

1. During initialization calls a nonblocking receive of the `ready` variable from all the slaves (`mpi_irecv`).
2. Tests if some slave has sent the `ready` variable (`mpi_test`).
3. If not, it continues its work. If a slave has sent its `ready` variable it sends (with a blocking send) to the slave the number of the next task to do (`mpi_send`) or the `no_work` number if there is no more work to do.
4. Finally makes another nonblocking receive of the `ready` variable from the slave that has received the work to do and continues its work.

Asynchronous parallelization via MPI routines

Slave:

1. Sends (with a blocking send) the `ready` variable to the master (`mpi_send`).
2. Receives (with a blocking receive) the number of the task to do. When it receives it, it starts to do its work or exit if the task number corresponds to `no_work` (`mpi_receive`).
3. When it finishes its task it restarts from [1]

To coordinate the work it is sufficient to initialize the master doing [1] at the beginning of the asynchronous work and that the master calls as often as possible a routine that executes [2], [3], [4] (for instance after each scf step). The most often the master calls this routine the shorter is the inactivity interval of the slaves.