LetzElPhC Documentation

Muralidhar Nalabothula

March 17, 2024

Contents

Chapter 1

About the Code

LetzElPhC is a C code, which computes electron-phonon coupling matrix elements from the outputs of standard DFT and DFPT calculations. It currently only supports the Quantum Espresso code with long term plans to extend it to the Abinit code. The primary objective of this project is to support electron-phonon related calculations in the YAMBO code (>=5.2) and works only with *norm-conserving* pseudo-potentials. It is currently released under MIT license and is hosted on github.

1.1 Main features

- Exploits full crystal symmetries and the output quantities are fully compatible with the YAMBO code without any phase issues.
- Has multiple levels of parallelization: OpenMP, plane-wave, k-point and q-point.
- Fully parallel IO using parallel NetCDF-4/HDF5 libraries.
- Highly portable. The code can be compiled more or less on any CPU architecture and operating system with minimal/no changes.

Chapter 2

Installing the Code

2.1 Mandatory requirements

- GNU Make
- Your favourite C99 compiler with complex number support.
  Ex : GCC, Clang, ICC, AMD C-Compiler, MinGW (for windows), PGI, Arm C compilers etc.
- MPI implementation (must support at least MPI-standard 2.1 standard)
  Ex: Open-MPI, MPICH and its flavours, Intel MPI compiler, Microsoft MPI (for windows) etc.
- FFTW-3 or Intel-MKL
- HDF5 and NETCDF-4 libraries with Parallel IO support
  (must be compiled with MPI)
- Your favourite BLAS library.
  Ex : Openblas, Blis, Intel-MKL, Atlas etc.

2.2 Installing

LetzElPhC employs standard make build system. There are some sample make files in *sample_config* directory. Copy it to the *src* directory and rename it as *make.inc*. Now, go to the *src* folder and edit the *make.inc* file according to your needs and type (in the same *src* directory)

```
$ make
#### You can also compile the code in parallel with -j option
$ make -j n
```

```
#### where n is number of processess.
```

If you successfully compile the code, you should find "*lelphc*" executable in the *src* directory.

If you have hard time finding the libraries, you can go the YAMBO code installation directory, and open the *report* file in the *config* directory (of course after successfully installing YAMBO). This will list all the required libraries and include paths.

Here are the list of variables in the *make.inc* file with explanations.

```
CC                     :=  mpicc
#### MPI C compiler mpicc/mpiicc (for intel),
CFLAGS                 := -O3
#### -O3 is to activate compiler optimizations
LD_FLAGS               :=
#### use this to pass any flags to linker

#### **** OPENMP BUILD ***
#### If you wish to build the code with openmp support
#### uncomment the below line
# OPENMP_FLAGS         := -DELPH_OMP_PARALLEL_BUILD
#### Aditionally, you need to add openmp compiler flag to
#### CFLAGS and LD_FLAGS.
#### Just uncomment the below two lines
# CFLAGS               += -fopenmp ## use -qopenmp for intel
# LD_FLAGS             += -fopenmp ## use -qopenmp for intel

#### FFTW3 include and libs (see FFT flag in yambo config/report)
FFTW_INC               :=  -I/opt/homebrew/include
FFTW3_LIB              :=  -L/opt/homebrew/lib -lfftw3_threads -
    lfftw3f -lfftw3f_omp -lfftw3_omp -lfftw3
#### Note if using FFTW
#### Yambo uses double precision FFTW regardless of the precision
    with which Yambo is built. In contrast, you need to link single
    (double) precision FFTW for single (double) precision LetzElPhC.
     please refer to https://www.fftw.org/fftw3_doc/Precision.html .
     Also you refer to https://www.fftw.org/fftw3_doc/
    Multi_002dthreaded-FFTW.html  if compiling with openmp support.



#### Blas and lapack libs (see BLAS and LAPACK flag in yambo config
    /report)
BLAS_LIB               :=  -L/opt/homebrew/opt/openblas/lib -lopenblas
#### you need to add both blas and lapack libs for ex : -lblas -
    llapack

#### netcdf libs and include
#### (see NETCDF flag in yambo config/report)
NETCDF_INC             :=  -I/Users/murali/softwares/core/include
NETCDF_LIB             :=  -L/Users/murali/softwares/core/lib -lnetcdf

#### hdf5 lib (see HDF5 flag in yambo config/report)
```

```
HDF5_LIB            :=  -L/opt/homebrew/lib  -lhdf5

#### incase if you want to add additional include dir and libs
INC_DIRS            :=
LIBS                :=


#### Notes Extra CFLAGS
### add -DCOMPILE_ELPH_DOUBLE if you want to compile the code in
    double precession
### if you are using yambo <= 5.1.2, you need to add "-
    DYAMBO_LT_5_1" to cflags
### for openmp use -DELPH_OMP_PARALLEL_BUILD in CFLAGS and set -
    fopenmp in LD_FLAGS and CFLAGS
```

Chapter 3

Running the code

3.1 Running DFT and DFPT (Step 0)

First and foremost, we need to obtain the following ingreadiants

- Kohn-Sham wavefunctions (These are obtained by doing a non-selfconsistant calculation after obtaining the ground state density of the system.)
- Phonon eigenvectors and perturbed Hatree and Exchange potentials due to phonon modes (obtained by performing a DFPT run after finding the ground state)

With Quantum Espresso code, we do the following (you can find an example in **examples/qe/silicon**),

- Do a scf calculation to get the ground state.
- Now perform a DFPT calculation with ph.x executable to obtain dynamical matrices and change in potentials on uniform q-point grid
- And finally, perform a nscf calculation to obtain the wavefunctions on uniform kpoint grid

Note : It is very importart to set the *dvscf* flag in ph.x input, else you need to rerun the whole calculation again. Also, q-grid must be commensate with the k-grid used in the nscf calculation (use whatever for scf, it does not matter).

Once you performed all these steps, go to the nscf folder and enter into *prefix.save* directory, where the wavefunctions are stored. Then run *p2y* and *yambo* to obtain the **SAVE** folder

```
$ p2y
#### this will create yambo SAVE directory
$ yambo
#### This will additonally create some more files
```

If you finished the above steps successfully, you are ready to start using the LetzElPhC code.

3.2 Running the preprocessor (Step 1)

Once we have the SAVE directory, we need to create **ph_save** folder. To create this we need to go the phonon calculation directory and run the preprocessor with the flag *-pp*.

```
$ cd /path/to/phonon calculation directory
#### run the preprocessor
$ lelphc -pp --code=qe -F PH.X_input_file
#### where PH.X_input_file is input file of ph.x code that was used
    to compute phonons
```

After successfully running the preprocessor, you should see ph_save directory. This directory contains all the change in potentials, dynamical matrices and pattern files.

If you wish to create a soft symlink instead of copying these files, you need to set an environment variable

```
$ export ELPH_COPY_CMD="ln -s"
```

and run the preprocessor.

Similarly, you can also change the name of the ph_save directory by the another environment variable

```
$ export ELPH_SAVE_DIR=ph_save_name_you_want
```

Remarks:

- The new format XML dynamical matrix files are currently **not supported.**
- The preprocessor currently **does not** work with image parallelization. If you want to use image parallelization, you have to copy all the phonon files to _ph0 directory and run the preprocessor.

3.3 Performing the elph calculation (Final step)

Once you have SAVE and ph_save folders, you are all set to go. Run the following command with LetzElPhC input file in any directory where you want to perform the calculation

```
$ mpirun -n 4 lelphc -F LetzElPhC_input_file
#### here we are running with 4 mpi processess.
```

Below is detailed description of input file

```
    nkpool          = 1
    # k point parallelization (number of kpools)

    nqpool          = 1
    # q point parallelization (number of qpools)

    ## note (nkpool*nqpool) must divide total number of cpus.

    ## For example, if you run the code on 12 processess,
    ## and set nkpool = 3 and nqpool = 2
    ## then, we have 2 sets of cpus working subset of qpoints
    ## with each group having 3 sub groups which that work on
    ## subset of kpoints. So in total, we have 6 subgroups, each
    ## having 2 cpus that distribute plane waves

    ##  {1,2,3,4,5,6,7,8,9,10,11,12} (total cpus)
    ##   ____________|________________
    ##  |    divided in to 2 qpools    |
    ## (qpool 1) {1,2,3,4,5,6}    (qpool 2) {7,8,9,10,11,12}
    ## ________|________     ________|_________
    ## |        |        |    |       |         |
    ## kp1     kp2     kp3  kp1     kp2       kp3
    ## where kp1 are kpools each containg 2
    ## cpus work on subset of plane waves

    start_bnd       = 1
    # starting band to consider in elph calculation

    end_bnd         = 40
    # last band to consider in elph calculation

    save_dir        = /Users/murali/phd/one_phonon_raman/wse2/SAVE
    # SAVE dir you create

    ph_save_dir     = /Users/murali/phd/one_phonon_raman/wse2/
   ph_save
    # ph_save directory that was created with preprocessor

    kernel          = dfpt
    ## type of kernel, only dfpt supported right Now

    convention      = standard
    # standard/yambo, If standard (default)
    # <k+q|dV|k> is computed. if yambo, <k|dV|k-q> is outputed

    ###  ##, !, ; are considered as comments
```

Bibliography