Efficient materials modelling on HPC with Quantum ESPRESSO, Siesta and Yambo

In recent years, computing technologies underlying materials modelling and electronic structure calculation have evolved rapidly. High-performance computing (HPC) is transitioning from petascale to exascale, while individual compute nodes are increasingly based on heterogeneous architectures that every year become more diversified due to different vendor choices. In this environment, electronic structure codes also have to evolve fast in order to adapt to new hardware facilities. Nowadays, state-of-the-art electronic structure codes based on modern density functional theory (DFT) methods allow treating realistic molecular systems with a very high accuracy. However, due to the increased complexity of the codes, some extra skills are required from users in order to fully exploit their potential.

This training material gives a broad overview of important fundamental concepts for molecular and materials modelling on HPC, with a focus on three of the most modern codes for electronic structure calculations (QUANTUM ESPRESSO, SIESTA and Yambo). Theory sections are interleaved with practical demonstrations and hands-on exercises.

QUANTUM ESPRESSO is one of the most popular suites of computer codes for electronic-structure calculations and materials modelling at the nanoscale, based on density-functional theory, plane waves, and pseudopotentials. It is able to predict and give fundamental insights about many properties of materials, molecular systems, micro and nanodevices, biological systems, in many fields, providing a huge amount of data for data-driven science applications.

SIESTA is a pseudopotential-based DFT software whose strength lies in its use of atomic-like strictly-localised basis sets: the use of a “good first approximation” to the full problem decreases the number of basis functions needed to achieve a given accuracy, and the finite support of the orbitals leads to sparsity in the Hamiltonian and overlap matrices, thus enabling the use of reduced-scaling methods. The functionalities of SIESTA include, amongst others, the calculation of energies and forces, molecular-dynamics simulations, band structures, densities of states, spin-orbit couplings, van der Waals functionals, DFT+U for correlated systems, real-time TDDFT, and non-equilibrium calculations with TranSIESTA.

YAMBO is an open-source code implementing first-principles methods based on Green’s function (GF) theory to describe excited-state properties of realistic materials. These methods include the GW approximation, the Bethe Salpeter equation, nonequilibrium GF

(NEGF) and TDDFT, allowing for the prediction of accurate quasiparticle energies (e.g. ARPES band structures), linear and non-linear optical properties, capturing the physics of excitons, plasmons, and magnons. It is also possible to calculate temperature-dependent electronic and optical properties via electron-phonon coupling and nonequilibrium and non-linear optical properties via NEGF real-time simulations (pump-probe experiments, etc).

MAX (MAterials design at the eXascale) is a European Centre of Excellence which enables materials modelling, simulations, discovery and design at the frontiers of the current and future High-Performance Computing (HPC), High Throughput Computing (HTC) and data analytics technologies. MaX's challenge lies in bringing the most successful and widely used open-source, community codes in quantum simulations of materials towards exascale and extreme scaling performance and make them available for a large and growing base of researchers in the materials' domain.

⚙ Prerequisites

- Some familiarity with density functional theory (DFT), self-consistent field (SCF) calculations and plane wave basis sets is desirable as the workshop will not cover the fundamental theory of these topics.
- Familiarity with working in a Linux environment and some experience with working on an HPC system is needed to participate in the hands-on exercises.

Who is the course for?

This workshop is aimed towards researchers and engineers who already have some previous experience with materials modelling and electronic structure calculations.

Day 1: Quantum ESPRESSO I

Lectures

Overview of Quantum ESPRESSO suite of codes and main features

Quantum ESPRESSO on HPC systems

Day 2: Quantum ESPRESSO II

Lectures

Introduction to Density Functional Perturbation Theory

Introduction to Time Dependent Density Functional Perturbation Theory

Phonons for HPC and GPUs

SIESTA setup

Directory with tutorials

For every practical there is a folder in the shared directory `/leonardo_work/EUHPC_TD02_030/siesta-tutorials` that contains all the files you will need for said practical. At the start of every tutorial, please copy the required files from that shared directory to your scratch folder. For example, for the first tutorial:

```
$ cp -pr /leonardo_work/EUHPC_TD02_030/siesta-tutorials/01-FirstEncounter_I .
```

Please do NOT copy these files before the start of the practical, in case they are updated shortly before the practical starts.

Running SIESTA

You will find a sample script on how to run SIESTA in `/leonardo_work/EUHPC_TD02_030/softwares/siesta-5.0beta1/runsample.sh`:

```
#!/bin/bash
#SBATCH -J tutorialXX
#SBATCH -n 8
#SBATCH -t 0:30:00
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --partition=boost_usr_prod
#SBATCH -D .

# DO NOT CHANGE THIS LINE
source /leonardo_work/EUHPC_TD02_030/softwares/siesta-5.0beta1/siestarc.sh

# EDIT THE CORRECT INPUT AND OUTPUT FILES.
srun -n 8 siesta < input.fdf > output.out
```

Note that `input.fdf` and `output.out` are generic input and output file names, for each execution of siesta you will need to change them to the actual names of your input and output files. It is also convenient to change the name of the job (`#SBATCH -J option`) to something that allows you to identify (vs. your other submissions) when checking the status of the queue.

Suggested Software for Visualization

If you run visualizations in your own PC, you may want to have a few of these installed. Some are redundant, and will depend on your own preferences.

- gnuplot
- xcrysden
- vesta
- vmd
- ovito
- Python with matplotlib

Day 3: SIESTA I

SIESTA basics

Lecture by Prof. Emilio Artacho (CIC NanoGUNE, Ikerbasque, and University of Cambridge).

Slides available here: SIESTA-Intro.pdf.

A first contact with SIESTA: inputs, execution and outputs

Practical session led by Dr. Federico Pedron (ICN2).

Tutorials covered:

- A First Encounter - Part 1: Running SIESTA. Files available at:

```
/leonardo_work/EUHPC_TD02_030/siesta-tutorials/day3-Wed/01-FirstEncounter_I
```

- A First Encounter - Part 2: Choosing your level of theory. Files available at:

```
/leonardo_work/EUHPC_TD02_030/siesta-tutorials/day3-Wed/02-FirstEncounter_II
```

Introductory slides available here: SIESTA-First_encounter.pdf.

Basis sets

Lecture by Dr. Miguel Pruneda (CINN-CSIC).

Slides available here: SIESTA-Basis_sets.pdf.

Basis set optimization

Practical session led by Dr. Federico Pedron (ICN2).

Tutorials covered:

- Basis set optimization

- Basis sets - Tips and tricks

Files for the tutorial:

```
/leonardo_work/EUHPC_TD02_030/siesta-tutorials/day3-Wed/03-BasisSets
```

Introductory slides available here: SIESTA-Basis_set_optimization.pdf.

Convergence (k points, mesh, mixing)

Practical session led by Dr. Catalina Coll (ICN2).

Tutorials covered:

- The real-space grid. Files available at:

  ```
  /leonardo_work/EUHPC_TD02_030/siesta-tutorials/day3-Wed/04a-GridConvergence
  ```

- Sampling of the BZ with k-points. Files available at:

  ```
  /leonardo_work/EUHPC_TD02_030/siesta-tutorials/day3-Wed/04b-KPointConvergence
  ```

- The self-consistent-field cycle. Files available at:

  ```
  /leonardo_work/EUHPC_TD02_030/siesta-tutorials/day3-Wed/04c-SCF
  ```

Introductory slides available here: SIESTA-Convergence.pdf.

Day 4: SIESTA II

Molecular Dynamics with SIESTA

Practical session led by Dr. Ernane de Freitas (ICN2).

Tutorials covered:

- Molecular Dynamics. Files available at:

  ```
  /leonardo_work/EUHPC_TD02_030/siesta-tutorials/day4-Thu/01-MolecularDynamics
  ```

Introductory slides available here: SIESTA-MD.pdf.

Analysis tools

Practical session led by Dr. Miguel Pruneda (CINN-CSIC).

Tutorials covered:

- Analysis tools. Files available at:

```
/leonardo_work/EUHPC_TD02_030/siesta-tutorials/day4-Thu/02-Analysis
```

Introductory slides available here: SIESTA-Analysis_tools.pdf.

Features available in SIESTA: spin-orbit couplings, TranSIESTA, and others

Lecture by Dr. Nick Papior (Technical University of Denmark).

Slides available here: SIESTA-Features.pdf.

Pushing the boundaries of SIESTA: accelerated and massively parallel solvers

Practical session led by Dr. Alberto García (ICMAB-CSIC).

Tutorials covered:

- ELSI-ELPA.
- ELSI-PEXSI.

Files for the tutorial:

```
/leonardo_work/EUHPC_TD02_030/siesta-tutorials/day4-Thu/03-SiestaSolvers
```

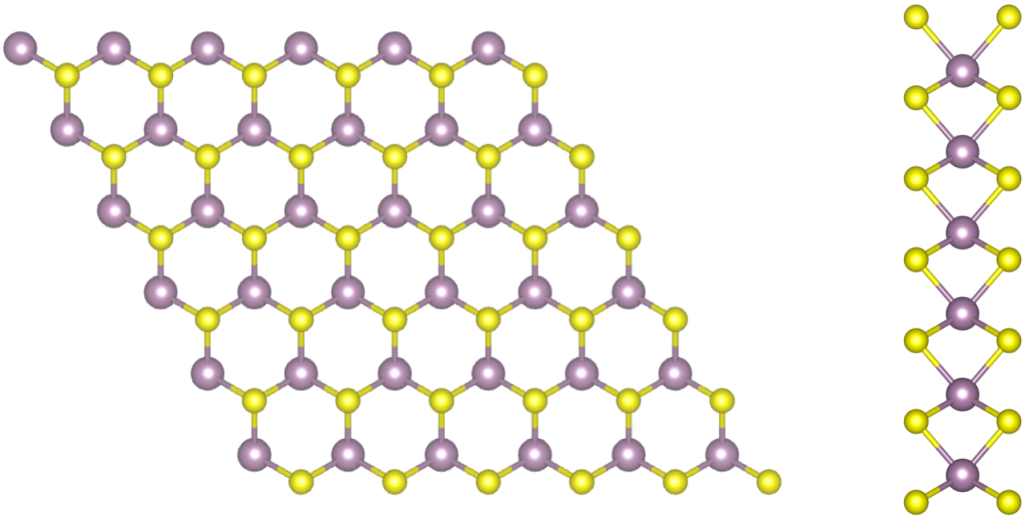
Slides available here: SIESTA-Solvers.pdf.

Yambo tutorial: Quasiparticles in the GW approximation

In this tutorial you will learn how to run a GW simulation using Yambo on a HPC machine.

You will compute the quasiparticle corrections to the band structure of a free-standing single layer of MoS\(_2\) while learning about convergence studies, parallel strategies, and GPU calculations.

In the end, you will obtain a quasiparticle band structure based on the simulations, the first step towards the reproduction of an ARPES spectrum. Beware: we will not use fully converged parameters, so the final result should not be considered very accurate.



MoS\(_2\) monolayer (top and side views). Gray: Mo atoms, yellow: S atoms.

Many-body corrections to the DFT band gap

We want to describe the electronic energy levels using a better description of electron-electron interactions than DFT is capable of.

Essentially, we want to solve the non-linear quasiparticle equation at first order in the GW self-energy \(\Sigma\):

$$E^{QP}_{nk}=\epsilon_{nk}+Z_{nk}[\Sigma]\langle\psi_{nk}|\Sigma(\epsilon_{nk})-V_{xc}|\psi_{nk}\rangle$$

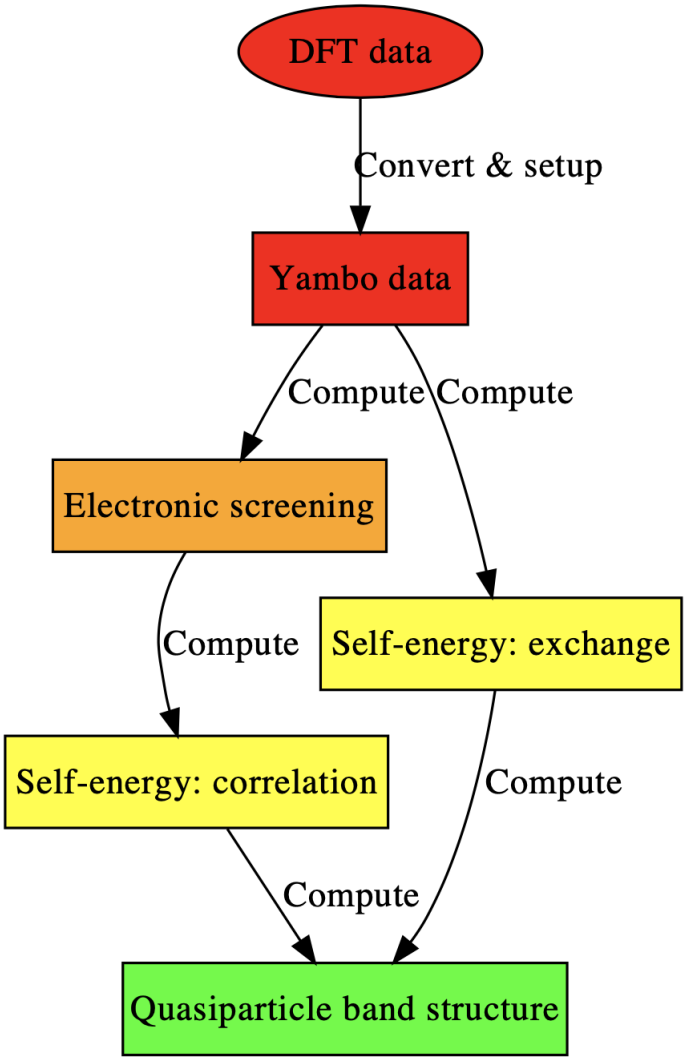
Here \(\epsilon_{nk}\) and \(\psi_{nk}\) are the Kohn-Sham energies and wavefunctions, respectively, while \(V_{xc}\) is the DFT exchange-correlation potential.

For each electronic state \(nk\), the self-energy can be separated into two components: a static, gap-opening term called the exchange self-energy (\(\Sigma^x\)), and an energy-dependent, usually gap-closing term called the correlation self-energy (\(\Sigma^c\)). These contributions are tackled separately by the code:

$$\Sigma_{nk}(\omega)=\Sigma_{nk}^x+\Sigma^c_{nk}(\omega)$$

The energy-dependent dynamical electronic screening \(\varepsilon^{-1}(\omega)\) is included in \(\Sigma^c\) and must therefore be calculated as well.

In this way, we can compute the “quasiparticle” corrections \(E^{QP}_{nk}\) to the single-particle Kohn-Sham eigenvalues \(\epsilon_{nk}\). The typical workflow for a GW calculation is:



Set up a Yambo calculation

Go to your user work directory and copy the material from the shared folder

```
cp $WORK/MoS2_HPC_tutorial_Leonardo.tar.gz .
```

(as an alternative you can download the materials for the tutorial, 1.2GB, it may take a couple of minutes):

```
wget https://media.yambo-
code.eu/educational/tutorials/files/MoS2_HPC_tutorial_Leonardo.tar.gz
```

After one of the two commands, just extract the data

```
tar -xvzf MoS2_HPC_tutorial_Leonardo.tar.gz
```

You can now enter the tutorial directory

```
cd MoS2_HPC_tutorial_Leonardo
```

Yambo `SAVE` folder

First of all, we need to convert some of the data produced in a previous non-self-consistent DFT calculation (in this case using Quantum ESPRESSO) into a convenient format for Yambo.

The QE save folder for MoS\(_2\) is already present at `00_QE-DFT`. We should move inside it and then run the `p2y` executable to generate the uninitialised `SAVE`.

But first, we need to access a node interactively:

```
srun --nodes=1 --ntasks-per-node=1 --gres=gpu:1 --cpus-per-task=8 --mem=490000 --
account=EUHPC_TD02_030 --partition=boost_usr_prod --qos=boost_qos_dbg --time=0:30:00 --
pty /bin/bash
```

Then we need to load the yambo-specific modules in in the cluster. On Leonardo Booster, we have

```
module load profile/chem-phys
module load yambo/5.2.0--openmpi--4.1.4--nvhpc--23.1
```

Finally:

```
cd 00_QE-DFT/mos2.save
mpirun -np 1 p2y
```

Now, we need to run the initialization step. Every Yambo run **must** start with this step. This will be automatically performed when you run Yambo on a `SAVE` directory for the first time. Just type

```
mpirun -np 1 yambo
```

and then check the yambo log called `l_setup`. The initialization step determines the \(G\)-vector shells and the \(k\)- and \(q\)-point grids based on the DFT calculation. If you check inside the `SAVE` you will see two types of databases. The static ones, starting with `ns.*`, are directly converted in the `p2y` step, while the dynamical ones, `ndb.*` are generated during the initialisation.

```
ls SAVE/
```

```
ndb.gops       # info on G-vector shells, etc
ndb.kindx      # info and k and q meshes
ns.db1         # info on geometry and KS bands
ns.kb_pp_pwscf # pseudopotential info
ns.wf          # wave functions info
...
```

The databases are written in netCDF format. Yambo has produced also a human readable output, `r_setup`, reporting relevant information such as lattice parameters, symmetries, atomic positions, k-points, DFT eigenvalues and band gaps. We can have a quick look at the sections.

```
vim r_setup
```

```
[02.01] Unit cells # Lattice geometry info
==================
  ...
[02.02] Symmetries # Symmetry ops. written explicitly
==================
  ...
[02.03] Reciprocal space  # Reciprocal lattice info
========================
...
[02.04] K-grid lattice # k-point coords. written explicitly
======================
...
[02.05] Energies & Occupations # Info on band gaps and occupations
============================== # DFT eigenvalues
...
[03] Transferred momenta grid and indexing # q-points (momentum transfer grid)
=========================================
```

Finally, let us move the `SAVE` and the report file to the directory where we will run the first GW calculation.

```
mv SAVE r_setup ../../01_GW_first_run/
cd ../../01_GW_first_run/
```

Yambo input file

Now that we have a working `SAVE`, it is time to generate the input file we will be using for our first GW calculation.

This can be done by the `yambo` executable via command-line instructions.

If you type

```
yambo -h
```

You will get a list of the possibile options:

```
 ___ __  _____  __ __  _____   _____
|   Y  ||  _  ||  Y  ||  _  \ |  _  |
|   |  ||. |  ||.    ||. |  / |. |  |
 \   _/ |. _  ||.\ / ||. _  \ |. |  |
  |: |  |: |  ||: |  ||: |   \|: |  |
  |::|  |:.|:.||:.|:.||::.   /|::.  |
  `--"  `-- --"`-- --"`-----" `-----"
 'A shiny pot of fun and happiness [C.D.Hogan]'

 This is      : yambo
 Version      : 5.1.0 Revision 21422 Hash fde6e2a07
 Configuration: MPI+OpenMP+SLK+HDF5_MPI_IO

 Help & version:
 -help            (-h) <string>   :<string> can be an option (e.g. -h optics)
 -version                         :Code version & libraries

 Input file & Directories:
 -Input           (-F) <string>   :Input file
 -Verbosity       (-V) <string>   :Input file variables verbosity (more with -h
Verbosity)
 -Job             (-J) <string>   :Job string
 -Idir            (-I) <string>   :Input directory
 -Odir            (-O) <string>   :I/O directory
 -Cdir            (-C) <string>   :Communication directory

 Parallel Control:
 -parenv          (-E) <string>   :Environment Parallel Variables file
 -nompi                           :Switch off MPI support
 -noopenmp                        :Switch off OPENMP support

 Initializations:
 -setup           (-i)            :Initialization
 -coulomb         (-r)            :Coulomb potential

 Response Functions:
 -optics          (-o) <string>   :Linear Response optical properties (more with -h
optics)
 -X               (-d) <string>   :Inverse Dielectric Matrix (more with -h X)
 -dipoles         (-q)            :Oscillator strenghts (or dipoles)
 -kernel          (-k) <string>   :Kernel (more with -h kernel)

 Self-Energy:
 -hf              (-x)            :Hartree-Fock
 -gw0             (-p) <string>   :GW approximation (more with -h gw0)
 -dyson           (-g) <string>   :Dyson Equation solver (more with -h dyson)
 -lifetimes       (-l)            :GoWo Quasiparticle lifetimes

 Bethe-Salpeter Equation:
 -Ksolver         (-y) <string>   :BSE solver (more with -h Ksolver)

 Total Energy:
 -acfdt                           :ACFDT Total Energy

 Utilites:
 -Quiet           (-Q)            :Quiet input file creation
 -fatlog                          :Verbose (fatter) log(s)
 -DBlist          (-D)            :Databases properties
 -walltime             <int>      :Walltime (more with -h walltime)
 -memory               <int>      :Memory (more with -h memory)
 -slktest                         :ScaLapacK test

 YAMBO developers group (http://www.yambo-code.org)
```

In order to build our input, we need to use the options for a GW calculation. We want to use the plasmon pole approximation for the dynamical screening, solve the quasiparticle equation with the Newton method, and add a truncation of the Coulomb potential which is useful for 2D systems. In addition, we want to set up explicitly the parameters for parallel runs. Therefore we type:

```
yambo -gw0 p -g n -r -V par -F gw.in
```

Now we can exit the computing node, since all remaining calculations will be run by submitting jobs with the `slurm` scheduler.

```
exit
```

You can now inspect the input file `gw.in` and try to familiarize with some of the parameters. The input will come with default values for many parameters that we might need to change.

We discuss them below step by step.

Parameters for a GW calculation

We start with the runlevels:

```
gw0                                # [R] GW approximation
ppa                                # [R][Xp] Plasmon Pole Approximation for the Screened
Interaction
el_el_corr                         # [R] Electron-Electron Correlation
dyson                              # [R] Dyson Equation solver
rim_cut                            # [R] Coulomb potential
HF_and_locXC                       # [R] Hartree-Fock
em1d                               # [R][X] Dynamically Screened Interaction
```

Runlevels

- The **[R]** keyword refers to the runlevels: these flag tell Yambo which parts of the code should be executed. Each runlevel enables its own set of input variables. In particular here we have:
  - `rim_cut`: Coulomb potential random integration method and cutoff (enables [RIM] and [CUT] variables).
  - `gw0`: Yambo learns that it has to run a GW calculation (enables [GW] variables).
  - `HF_and_locXC`: calculation of exchange part of the self-energy \(\Sigma^x\) (i.e., Hartree-Fock approximation).

- `em1d` : enables the calculation of the dynamical screening of the electrons, i.e. the dielectric matrix ([X] variables). In this way Yambo can go beyond Hartree-Fock and compute \(\Sigma^c\).
- `ppa` : tells Yambo that the dynamical screening should be computed in the plasmon pole approximation ([Xp] variables).
- `dyson` : Yambo will solve the Dyson-like quasiparticle equation.

Going through the file we find:

```
EXXRLvcs=  37965           RL    # [XX] Exchange    RL components
VXCRLvcs=  37965           RL    # [XC] XCpotential RL components
```

Recall that we have, for the exchange self-energy:

$$\Sigma^x_{nk} = - \sum_G\sum_v \int \frac{d^3q}{2\pi^3} v(q+G)|\rho_{nv}(k,q,G)|^2f_{vk-q}$$

Exchange self-energy

- `EXXRLvcs` controls the number of Reciprocal Lattice vectors (i.e., G-vectors) used to build the exchange self-energy, while `VXCRLvcs` does the same for the exchange-correlation potential reconstructed from DFT. Since these two quantities are to be subtracted, it is important to keep the same values here (and possibly not change the default maximum value).

Let us now have a look at the parameters for the calculation of the correlation part of the self-energy. Recall that we have:

$$\Sigma^c_{nk} = i \sum_m \int \frac{d^3q}{2\pi^3} \sum_{GG'} v(q+G) \rho_{nmk}(q,G) \rho^*_{nmk}(q,G') \int d\omega' G^0_{mk-q}(\omega-\omega')\varepsilon^{-1}_{GG'}(q,\omega')$$

(Here, the \(\rho\)-terms represent the screening matrix elements which are computed separately by yambo and stored in their own database.)

The calculation is divided in two steps. First, the response function in the plasmon pole approximation (`em1d ppa`), under the keywords `[X]` and `[Xp]` , i.e., \(\varepsilon^{-1}_{GG'}(q,\omega)\).

```
Chimod= "HARTREE"               # [X] IP/Hartree/ALDA/LRC/PF/BSfxc
% BndsRnXp
   1 | 300 |                    # [Xp] Polarization function bands
%
NGsBlkXp= 1                 RL  # [Xp] Response block size
% LongDrXp
 1.000000 | 0.000000 | 0.000000 |        # [Xp] [cc] Electric Field
%
PPAPntXp= 27.21138          eV  # [Xp] PPA imaginary energy
XTermKind= "none"               # [X] X terminator ("none","BG" Bruneval-Gonze)
```

Response function

- `Chimod= "Hartree"` indicates that we compute the response function in the Random Phase Approximation (RPA).
- `BndsRnXp` represents the electronic states included in the response function \(\varepsilon\), and is a convergence parameter.
- `NGsBlkXp` is the number of G-vectors used to calculate the RPA response function \(\varepsilon^{-1}_{GG'}\). It is a convergence parameter and can be expressed in number of reciprocal lattice vectors (RL) or energy (Ry, suggested).
- `LongDrXp` represents the direction of the long-range auxiliary external electric field used to compute \(\varepsilon^{-1}_{GG'}(q)\) at \(q,G\rightarrow 0\). In general you have to be mindful of the system symmetries. In our case, we will put `1 | 1 | 1` to cover all directions.
- `PPAPntXp= 27.21138 eV` is the energy of the plasmon pole. We don't normally change this.
- `XTermKind` is used to specify a "terminator": this accelerates numerical convergence with respect to the number of bands `BndsRnXp`.

Next, we have the `[GW]` group of parameters controlling the next part of the correlation self-energy calculation:

```
% GbndRnge
   1 | 300 |                        # [GW] G[W] bands range
%
GTermKind= "none"               # [GW] GW terminator ("none","BG" Bruneval-Gonze,"BRS"
Berger-Reining-Sottile)
DysSolver= "n"                  # [GW] Dyson Equation solver ("n","s","g")
%QPkrange                       # [GW] QP generalized Kpoint/Band indices
1|7|1|300|
%
```

Correlation self-energy

- `GbndRnge` is the number of bands used to build the correlation self-energy. It is a convergence parameter and can be accelerated with the terminator `GTermKind`.

- `DysSolver="n"` specifies the method used to solve the linearised quasiparticle equation. In most cases, we use the Newton method `"n"`.
- `QPkrange` indicates the range of electronic (nk) states for which the GW correction \(\Sigma_{nk}\) is computed. The first pair of numbers represents the range of k-point indices, the second pair the range of band indices.

We now take a look at the parameters relative to the Coulomb interaction at small momenta and for 2D systems, which we should **edit now once and for all**.

```
RandQpts=1000000                        # [RIM] Number of random q-points in the BZ
RandGvec= 100                RL     # [RIM] Coulomb interaction RS components
CUTGeo= "slab z"                    # [CUT] Coulomb Cutoff geometry:
box/cylinder/sphere/ws/slab X/Y/Z/XY..
```

Coulomb potential

- The **[RIM]** keyword refers to a Monte Carlo random integration method performed to avoid numerical instabilities close to \(q=0\) and \(G=0\) in the \(q\)-integration of the bare Coulomb interaction - i.e. \(4\pi/(q+G)^2\) - for 2D systems.
- The **[CUT]** keyword refers to the truncation of the Coulomb interaction to avoid spurious interaction between periodically repeated copies of the simulation supercell along the \(z\)-direction (we are working with a plane-wave code). Keep in mind that the vacuum space between two copies of the system should be converged: here we are using 20 bohr but a value of 40 bohr would be more realistic.

Finally, we have the parallel parameters. We are going to discuss them at the end of the parallel section, we can skip them for now.

```
X_and_IO_CPU= ""                    # [PARALLEL] CPUs for each role
X_and_IO_ROLEs= ""                  # [PARALLEL] CPUs roles (q,g,k,c,v)
X_and_IO_nCPU_LinAlg_INV=-1         # [PARALLEL] CPUs for Linear Algebra (if -1 it is
automatically set)
X_Threads=0                         # [OPENMP/X] Number of threads for response functions
DIP_CPU= ""                         # [PARALLEL] CPUs for each role
DIP_ROLEs= ""                       # [PARALLEL] CPUs roles (k,c,v)
DIP_Threads=0                       # [OPENMP/X] Number of threads for dipoles
SE_CPU= ""                          # [PARALLEL] CPUs for each role
SE_ROLEs= ""                        # [PARALLEL] CPUs roles (q,qp,b)
SE_Threads=0                        # [OPENMP/GW] Number of threads for self-energy
```

In a GW calculation, the most important parameters to be numerically converged are:

- kpoint mesh (requires multiple nscf DFT runs)
- `BndsRnXp`

- `NGsBlkXp`
- `GbndRnge`
- [2D system]: vacuum separation with Coulomb cutoff (requires multiple scf+nscf DFT runs)

From the above discussion you can easily guess that many-body perturbation theory calculations are much more numerically expensive than DFT calculations.

The first run

We will start by running a single GW calculation. Here we will focus on the magnitude of the quasiparticle gap. This means that we only need to calculate two quasi-particle corrections, i.e., valence and conduction bands at the k-point where the minimum gap occurs. This information can be found by inspecting the report file `r_setup` produced when the `SAVE` folder was initialised. Just search for the string 'Direct Gap' and you'll see that the latter occurs at k-point 7 between bands 13 and 14:

```
[X] Filled Bands                                    :  13
[X] Empty Bands                                     :   14  300
[X] Direct Gap                                      :  1.858370 [eV]
[X] Direct Gap localized at k                       :  7
```

In addition, we will set the number of bands in `BndsRnXp` and `GbndRnge` to a small value, just to have it run fast. Hence, we modify the input file accordingly (check `BndsRnXp`, `GbndRnge`, `LongDrXp`, `QPkrange`):

```
rim_cut                          # [R] Coulomb potential
gw0                              # [R] GW approximation
ppa                              # [R][Xp] Plasmon Pole Approximation for the Screened
Interaction
dyson                            # [R] Dyson Equation solver
HF_and_locXC                     # [R] Hartree-Fock
em1d                             # [R][X] Dynamically Screened Interaction
X_Threads=0                      # [OPENMP/X] Number of threads for response functions
DIP_Threads=0                    # [OPENMP/X] Number of threads for dipoles
SE_Threads=0                     # [OPENMP/GW] Number of threads for self-energy
RandQpts=1000000                 # [RIM] Number of random q-points in the BZ
RandGvec= 100              RL    # [RIM] Coulomb interaction RS components
CUTGeo= "slab z"                   # [CUT] Coulomb Cutoff geometry:
box/cylinder/sphere/ws/slab X/Y/Z/XY..
% CUTBox
 0.000000 | 0.000000 | 0.000000 |        # [CUT] [au] Box sides
%
CUTRadius= 0.000000              # [CUT] [au] Sphere/Cylinder radius
CUTCylLen= 0.000000              # [CUT] [au] Cylinder length
CUTwsGvec= 0.700000              # [CUT] WS cutoff: number of G to be modified
EXXRLvcs=  37965           RL    # [XX] Exchange    RL components
VXCRLvcs=  37965           RL    # [XC] XCpotential RL components
Chimod= "HARTREE"                # [X] IP/Hartree/ALDA/LRC/PF/BSfxc
% BndsRnXp
   1 | 20 |                        # [Xp] Polarization function bands
%
NGsBlkXp= 1                RL    # [Xp] Response block size
% LongDrXp
 1.000000 | 1.000000 | 1.000000 |        # [Xp] [cc] Electric Field
%
PPAPntXp= 27.21138         eV    # [Xp] PPA imaginary energy
XTermKind= "none"                # [X] X terminator ("none","BG" Bruneval-Gonze)
% GbndRnge
   1 | 20 |                        # [GW] G[W] bands range
%
GTermKind= "none"                # [GW] GW terminator ("none","BG" Bruneval-Gonze,"BRS"
Berger-Reining-Sottile)
DysSolver= "n"                   # [GW] Dyson Equation solver ("n","s","g")
%QPkrange                        # [GW] QP generalized Kpoint/Band indices
7|7|13|14|
%
```

We are now ready to run this calculation. Since you should never run a Yambo calculation on the login node, we will need a submission script to add our job to the queue. A submission script optimized for Leonardo Booster (running on GPUs) is provided as an example. **Modify it to suit your specific machine**.

```
vim run_first_job.sh
```

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=8
#SBATCH --partition=boost_usr_prod
#SBATCH --time=0:05:00
#SBATCH --gres=gpu:2
#SBATCH --account=EUHPC_TD02_030
#SBATCH --job-name=first_job

export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=close

# load yambo
module load profile/chem-phys
module load yambo/5.2.0--openmpi--4.1.4--nvhpc--23.1

# run yambo
mpirun -np ${SLURM_NTASKS} --map-by socket:PE=8 --rank-by core \
        yambo -F gw.in -J job_00_first_run -C out_00_first_run
```

We will ignore all details regarding parallelization, as it will be covered in the next section. Since there are no lowercase flags after `yambo`, it is not going to generate an input file, but rather, run the one specified by `-F`. Now, go ahead an submit this job

```
sbatch run_first_job.sh
```

The status of the jobs can be monitored via:

```
squeue -u $USER        # to inspect the status of jobs
                       # (hint: make a unix alias, if you like)
scancel <jobid>        # to delete jobs in the queue
```

The newly generated databases will be stored in the job directory, as specified by `-J`, while the report, log and output files will be stored in the communications directory (`-C`). As this is your first `yambo` run, take a moment to inspect the report and log files, which you can find inside the `-C` directory. In these report and log files, you can see the steps performed by `yambo`. For instance, the code calculates the screening at every k-point and stores it in the PPA database called `ndb.pp`. By opening the report

```
vim out_00_first_run/r-job_00_first_run_HF_and_locXC_gw0_dyson_rim_cut_em1d_ppa
```

you will see

```
[07] Dynamic Dielectric Matrix (PPA)
 ====================================

 [WR./job_00_first_run//ndb.pp]--------------------------------
```

Then, the actual GW section will use this calculated dielectric screening to construct the correlation part of the self-energy:

```
[09.01] G0W0 (W PPA)
  ====================

  [  GW  ] Bands range     :   1  20
  [GW/PPA] G damping       :  0.100000 [eV]


  QP @ state[ 1 ] K range:   7   7
  QP @ state[ 1 ] b range:  13  14

  [RD./job_00_first_run//ndb.pp]------------------------------
```

Now, inspect the output file

```
vim out_00_first_run/o-job_00_first_run.qp
```

```
# Vxc  =Slater exchange(X)+Perdew & Zunger(C)
# Vnlxc=Hartree-Fock
#
#    K-point            Band               Eo [eV]           E-Eo [eV]          Sc|Eo
[eV]
#
         7                 13                 0.000000          -0.025774
0.543987
         7                 14                 1.858370           3.496193
-0.417555
#
```

In this file, `Eo` is our starting point (DFT) while `E-Eo` shows the GW correction one should apply to obtain the quasi-particle energies. In order to calculate the gap (automatically from the command line), we'll use some simple commands. First, we get everything that is not a `#` symbol `grep -v '#'` and we pass that to another command with a "pipe" `|`. Then, `tail -n 1` / `head -n 1` will retain the first/last line, and `awk '{print $3+$4}'` will get us the sum of the third and fourth columns. Altogether, this would be as follows

```
grep -v '#' out_00_first_run/o-job_00_first_run.qp|head -n 1| awk '{print $3+$4}'
-0.025774
grep -v '#' out_00_first_run/o-job_00_first_run.qp|tail -n 1| awk '{print $3+$4}'
5.35456
```

These two commands give us the quasiparticle energies we've calculated - their difference is the GW-corrected optical gap.

GW convergence

In this part of the tutorial, we will study convergence with respect to some of the parameters mentioned above. In order to complete this tutorial within a single hands-on session, we will restrict ourselves to a very coarse k-point grid. Hence, we will perform our convergence studies on top of a DFT calculation done with a $6 \times 6 \times 1$ k-point grid and without spin-orbit coupling: the `SAVE` we generated earlier. While this will speed up calculations and could be run even on a single GPU card, you should be aware that such coarse sampling of the BZ is significantly underconverged and should only be used for educational purposes. In addition, spin-orbit interaction is extremely relevant for the valley physics of MoS2 and should not be neglected in realistic calculations. Let's move into the appropriate directory

```
cd ../02_GW_convergence
```

Response function $\varepsilon^{-1}_{GG'}$ - Bands and G-vectors

We are now ready to start our convergence tests. We'll begin with the variables controlling the polarization function, i.e., `NGsBlkXp` for the number of G-vectors and `BndsRnXp` for the number of bands. For this, we will keep `GbndRnge` constant at a reasonably high value - you can inspect the input `i01-GW`

```
vim i01-GW
```

and check that you have:

```
% GbndRnge
   1 | 80 |                        # [GW] G[W] bands range
%
```

Since we need to run `yambo` for several values of `NGsBlkXp` and `BndsRnXp`, it makes sense to use two nested loops. That is exactly what we did in the submission script `run01_converge_pol.sh`. Since this will take a few minutes, save time by submitting it straight away and we will have a look at it while it runs:

```
sbatch run01_converge_pol.sh
```

Monitoring

You can monitor that the job is running by the squeue command

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

and also by checking the files created in your folder

```
ls -ltr
```

```
i01-GW_Xp_20_bands_6_Ry
job_Xp_20_bands_6_Ry
out_Xp_20_bands_6_Ry
i01-GW_Xp_20_bands_8_Ry
out_Xp_20_bands_8_Ry
job_Xp_20_bands_8_Ry
i01-GW_Xp_20_bands_10_Ry
out_Xp_20_bands_10_Ry
job_Xp_20_bands_10_Ry
i01-GW_Xp_20_bands_12_Ry
out_Xp_20_bands_12_Ry
job_Xp_20_bands_12_Ry
summary_01_20bands.txt
i01-GW_Xp_40_bands_6_Ry
...
```

Finally you can monitor how runs are proceeding by looking into the log files

```
tail -f out_Xp_*_bands_*/LOG/*_1
```

```
==> out_Xp_20_bands_6_Ry/LOG/l-
job_Xp_20_bands_6_Ry_HF_and_locXC_gw0_dyson_rim_cut_em1d_ppa_CPU_1 <==
 <15s> P1: [TIMING]                                          io_WF :      1.1353s
CPU (34 calls,   0.033 sec avg)
 <15s> P1: [TIMING]                                    WF_load_FFT :      1.1538s
CPU ( 7 calls,   0.165 sec avg)
 <15s> P1: [TIMING]                                    io_KB_pwscf :      1.1918s
CPU ( 6 calls,   0.199 sec avg)
 <15s> P1: [TIMING]                               DIPOLE_transverse :      2.4771s
CPU
 <15s> P1: [TIMING]                                    io_fragment :      2.6220s
CPU (46 calls,   0.057 sec avg)
 <15s> P1: [TIMING]                                           io_X :      3.1953s
CPU (19 calls,   0.168 sec avg)
 <15s> P1: [TIMING]                                       Dipoles :      4.4012s
CPU
 <15s> P1: [11] Memory Overview
 <15s> P1: [12] Game Over & Game summary
 <15s> P1: [TIMING]            [Time-Profile]: 15s

==> out_Xp_20_bands_8_Ry/LOG/l-
job_Xp_20_bands_8_Ry_HF_and_locXC_gw0_dyson_rim_cut_em1d_ppa_CPU_1 <==
 <11s> P1: X@q[6] |                                             | [000%] --(E) --(X)
 <11s> P1: X@q[6] |##############################################| [100%] --(E) --(X)
 <11s> P1: [PARALLEL distribution for RL vectors(X) on 2 CPU] Loaded/Total
(Percentual):31878/64009(50%)
 <11s> P1: [PARALLEL distribution for RL vectors(X) on 2 CPU] Loaded/Total
(Percentual):18975/64009(30%)
 <11s> P1: [X-CG] R(p) Tot o/o(of R):   153   504   100
 <11s> P1: Xo@q[7] |                                             | [000%] --(E) --(X)
 <11s> P1: Xo@q[7] |##############################################| [100%] --(E) --(X)
 <11s> P1: [PARALLEL distribution for X Frequencies on 1 CPU] Loaded/Total
(Percentual):2/2(100%)
 <11s> P1: X@q[7] |                                             | [000%] --(E) --(X)
 <11s> P1: X@q[7] |##############################################| [100%] --(E) --(X)
 <11s> P1: [PARALLEL distribution for RL vectors(X) on 2 CPU] Loaded/Total
(Percentual):31878/64009(50%)
 <11s> P1: [08] Local Exchange-Correlation + Non-Local Fock
```

Let's now have look into the job we just submitted.

```
vim run01_converge_pol.sh
```

First, we defined the double loop and we intialize a summary file for each iteration of the outer loop by printing a header to it. The input file `i01-GW` is used as a template for every calculation in the loops, so we assign it to a variable.

```
file0='i01-GW'

for POL_BANDS in 20 40 60 80; do

echo 'NGsBlkXp [Ry]   E_vale [eV]   E_cond [eV]' > summary_01_${POL_BANDS}bands.txt

for NGsBlkXp_Ry in 6 8 10 12; do

(...)

done
done
```

Inside the loops, we generate some useful labels which will come in handy to distinguish between runs. Then, we pass the variables from the loops to the `sed` command, in order to generate new files in an automated way (`sed` replaces any matching string with whatever is provided by the loop variable). Next, we run `yambo` using the labels to specify different job `-J` and communication `-C` directories every time. Finally, we get the quasiparticle energies with `grep` commands as shown before and append a new line to the summary file. So, inside each loop, we have

```
label=Xp_${POL_BANDS}_bands_${NGsBlkXp_Ry}_Ry
jdir=job_${label}
cdir=out_${label}
filein=i01-GW_${label}

sed "s/NGsBlkXp=.*/NGsBlkXp=${NGsBlkXp_Ry} Ry/;
      /% BndsRnXp/{n;s/.*  1 |  ${POL_BANDS} |/}" $file0 > $filein

# run yambo
mpirun -np ${SLURM_NTASKS} --map-by socket:PE=8 --rank-by core yambo -F $filein -J
$jdir -C $cdir

E_GW_v=`grep -v '#' ${cdir}/o-${jdir}.qp|head -n 1| awk '{print $3+$4}'`
E_GW_c=`grep -v '#' ${cdir}/o-${jdir}.qp|tail -n 1| awk '{print $3+$4}'`

echo ${NGsBlkXp_Ry} '        ' ${E_GW_v} '        ' ${E_GW_c}  >>
summary_01_${POL_BANDS}bands.txt
```

Finally, let us plot this data. First, check that the job has finished with

```
squeue -u $USER
```

```
[$USER@login01 02_GW_convergence]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[$USER@login01 02_GW_convergence]$
```

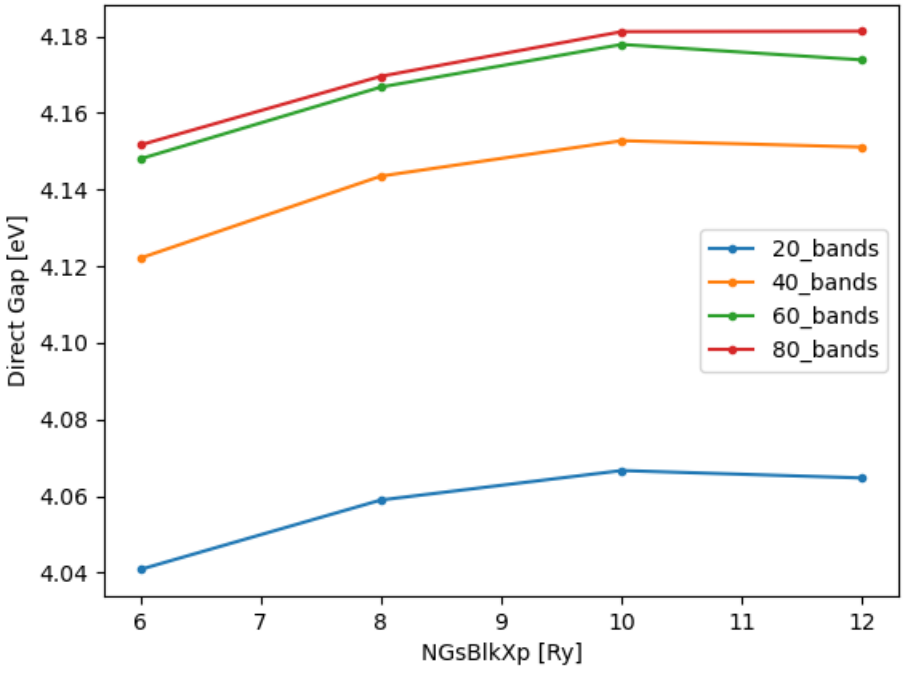
and verify that the energies were extracted correctly by inspecting the summary files. Remember to load the python module if you haven't done so yet, and then plot:

```
module load anaconda3/2023.03
python plot-01.py
```

The plot will produce a `fig-01.png` file. You can copy and open it in your local machine with something like

```
[Run this on another terminal in your local machine, fixing $USER, $LOGIN and
$TUTORIALPATH]
scp $USER@$LOGIN:$TUTORIALPATH/MoS2_HPC_tutorial_Leonardo/02_GW_convergence/fig-01.png
./
```

You should get:



For the purpose of the tutorial, we will choose 80 bands and 10 Ry as our converged parameters and move on. An error within 10 meV is usually acceptable. To retain the chosen variables, we'll make a copy of the corresponding input file:

```
cp i01-GW_Xp_80_bands_10_Ry i02-GW
```

Self-energy \(\Sigma^c\) - Bands

We will now proceed to converge the number of bands for the correlation part of the self-energy, i.e., `GbndRnge`. This step is actually simpler, since it only involves one loop. This is coded in the provided script `run02_converge_Gbnds.noBG.sh`. You can look into it

```
vim run02_converge_Gbnds.noBG.sh
```

and go ahead and submit it.

```
sbatch run02_converge_Gbnds.noBG.sh
```

✔ [OPTIONAL]: Use the terminator to accelerate convergence

While that runs, we'll have a look at the so-called Bruneval-Gonze (BG) terminator, which is a method to accelerate convergence with respect to empty bands. The variable that controls this for the bands in the correlation self-energy is `GTermKind`. This is currently set to "none" in `i02-GW`, so create a new input file `i02-GW_BG` and set this variable to "BG". We can do this in the command line by simply typing

```
sed 's/GTermKind=.*/GTermKind= "BG"/' i02-GW > i02-GW_BG
```

Note that `XTermKind` also offers the same terminator for the sum over bands of the polarization function (we just chose not to use it in the previous section of this excercise, and we will keep it as "none"). Now, copy the last submission script and edit it to run the same convergence test using the BG terminator.

```
cp run02_converge_Gbnds.noBG.sh run03_converge_Gbnds.BG.sh
```

Try and do this yourself first, and then continue reading to check your understanding. You will have to change the input file template, i.e., use `i02-GW_BG` where the terminator has been activated. Modify also the name of the newly generated input files in order to avoid overwriting. Change the name of the summary file for the same reason and, finally, modify the communications and job directories of `yambo`. Make sure you've done all the changes as outlined below.

```
file0='i02-GW_BG'
summaryfile=summary_03_BG.txt

(...)

label=Gbands_${G_BANDS}_BG
filein=i02-GW_${G_BANDS}_Gbands_BG
```

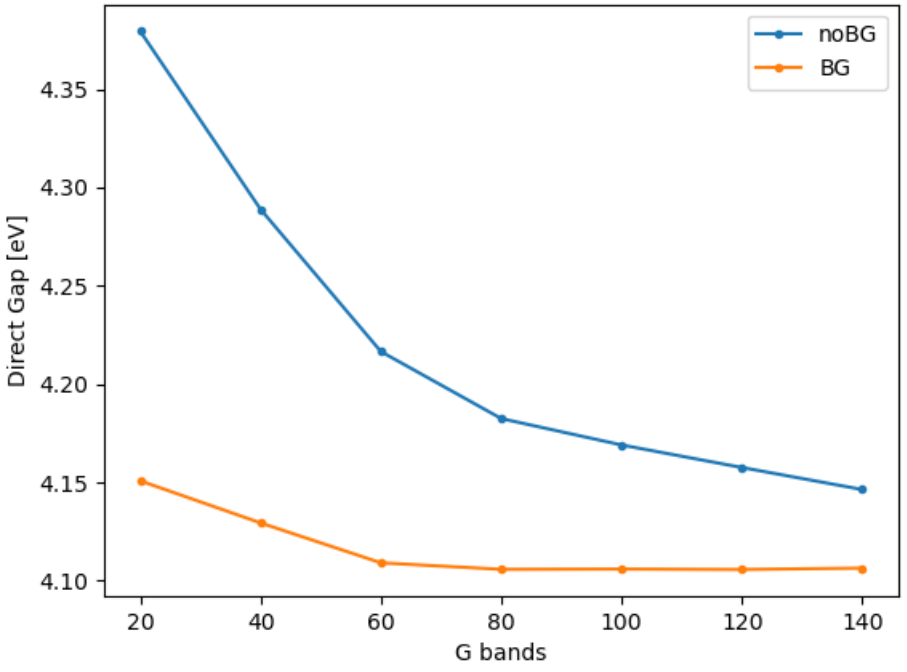
Now submit your newly edited script

```
sbatch run03_converge_Gbnds.BG.sh
```

While this runs, check if the previous job has finished, i.e., you should have a complete `summary_02_noBG.txt` file by now. For a visual result, proceed to plot them with

```
python plot-02.py
```

You should get

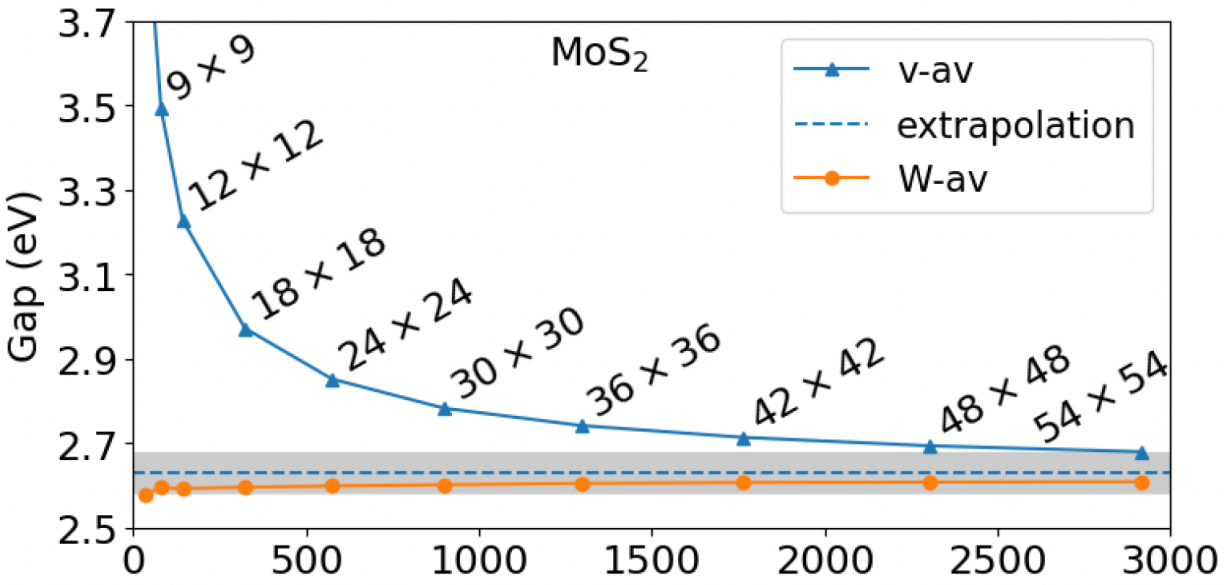


✔ OPTIONAL

If you also did the optional step, you can compare `summary_02_noBG.txt` with `summary_03_BG.txt` once `run03_converge_Gbnds.BG.sh` has finished - you'll see the effect of the terminator immediately. Just open the `plot-02.py` script and uncomment the line

`#list_of_files = ['summary_02_noBG.txt','summary_03_BG.txt']`, then rerun it with `python plot-02.py`. You can see that the terminator does a great job at accelerating convergence, and it allows us to use 60 bands incurring an error of only 3 meV (while the latter would have been larger than 0.1 eV had we not used the terminator).

We will end the convergence part of the tutorial with an important consideration about k-points convergence. The latter is the most cumbersome and computationally intensive among the various convergence tests, and it involves re-running the DFT step. For this reason (and for this reason only) it was ignored in this tutorial. However, it absolutely cannot be overlooked since it is crucial for the accuracy of the calculated GW corrections. You can read about k-points convergence in GW and, importantly, a very efficient workaround for 2D systems in a recent publication (here). MoS\(_2\) was one of the materials studied there, and it shows that our result, obtained with a \(6 \times 6 \times 1\) k-grid, is simply *off the chart* (blue line).



Guandalini, D'Amico, Ferretti & Varsano. npj Comput Mater 9

✓ [OPTIONAL]: Use the RIM-W accelerator

However you can try to get a reasonable correction via the RIM-W approach. Create a new input file copying a suitable one:

```
cp i02-GW_80_Gbands  i04-GW_80_Gbands_rimw
vim i04-GW_80_Gbands_rimw
```

Then, you just need to add the following two variables to the input file (for example just after the runlevel keywords)

```
RIM_W
RandGvecW= 15           RL
```

and prepare a submission script

```
cp  run02_converge_Gbnds.noBG.sh  run04_converge_rimw.sh
vim run04_converge_rimw.sh
```

Edit it by removing the loop and changing `summaryfile`, `label` and `filein`

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=8
#SBATCH --partition=boost_usr_prod
#SBATCH --time=0:10:00
#SBATCH --gres=gpu:4
#SBATCH --account=EUHPC_TD02_030
#SBATCH --job-name=mos2

export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=close

# load yambo
module load profile/chem-phys
module load yambo/5.2.0--openmpi--4.1.4--nvhpc--23.1

file0='i02-GW'
summaryfile=summary_04_rimw.txt

echo 'G bands     E_vale [eV]        E_cond [eV]      GAP [eV]' > $summaryfile

G_BANDS=80

label=Gbands_${G_BANDS}_rimw
jdir=job_${label}
cdir=out_${label}
filein=i04-GW_${G_BANDS}_Gbands_rimw

# run yambo
mpirun -np ${SLURM_NTASKS} --map-by socket:PE=8 --rank-by core yambo -F $filein -J
$jdir -C $cdir

E_GW_v=`grep -v '#' ${cdir}/o-${jdir}.qp|head -n 1| awk '{print $3+$4}'`
E_GW_c=`grep -v '#' ${cdir}/o-${jdir}.qp|tail -n 1| awk '{print $3+$4}'`

GAP_GW=`echo $E_GW_c - $E_GW_v |bc`

echo ${G_BANDS} '        ' ${E_GW_v} '        ' ${E_GW_c} '        ' ${GAP_GW} >>
$summaryfile
```

and then run

```
sbatch run04_converge_rimw.sh
```

How much do you get for the band gap ?

GW parallel strategies

MPI parallelization

For this section, let us enter the `03_GW_parallel` directory. If you were in the `02_GW_convegence` folder just do

```
cd ../03_GW_parallel
```

and inspect the input `gw.in`. You will see that we set low values for most of the convergence parameters except bands:

```
vim gw.in
```

```
FFTGvecs= 40        Ry    # [FFT] Plane-waves
EXXRLvcs= 2         Ry    # [XX] Exchange RL components
VXCRLvcs= 2         Ry    # [XC] XCpotential RL components
% BndsRnXp
    1 |  300 |              # [Xp] Polarization function bands
%
NGsBlkXp= 1             Ry    # [Xp] Response block size
% GbndRnge
    1 |  300 |              # [GW] G[W] bands range
%
%QPkrange                  # [GW] QP generalized Kpoint/Band indices
  1| 19| 23| 30|
%
```

Note that we also added `FFTGvecs` to reduce the size of the Fourier transforms (the default corresponds to Quantum ESPRESSO `ecutwfc`, i.e. 60 Ry in this case).

In addition, we have deleted all the parallel parameters since we will be setting them via the submission script.

Actually we are now dealing with a heavier system than before: as you can see from the `QPkrange` values, we have switched to a 12x12x1 k-point grid - having 19 points in the irreducible Brillouin zone - and turned spin-orbit coupling on in the DFT calculation (now the top valence band is number 26 instead of 13 because the bands are spin-polarized).

For this part of the tutorial, we will be using the `slurm` submission script `job_parallel.sh`, which is available in the calculation directory. If you inspect it, you will see that the script adds additional variables to the yambo input file. These variables control the parallel execution of the code:

```
DIP_CPU= "1 $ngpu 1"          # [PARALLEL] CPUs for each role
DIP_ROLEs= "k c v"            # [PARALLEL] CPUs roles (k,c,v)
DIP_Threads=  0               # [OPENMP/X] Number of threads for dipoles
X_and_IO_CPU= "1 1 1 $ngpu 1" # [PARALLEL] CPUs for each role
X_and_IO_ROLEs= "q g k c v"   # [PARALLEL] CPUs roles (q,g,k,c,v)
X_and_IO_nCPU_LinAlg_INV=1    # [PARALLEL] CPUs for Linear Algebra (if -1 it is
automatically set)
X_Threads=  0                 # [OPENMP/X] Number of threads for response functions
SE_CPU= "1 $ngpu 1"           # [PARALLEL] CPUs for each role
SE_ROLEs= "q qp b"            # [PARALLEL] CPUs roles (q,qp,b)
SE_Threads=  0                # [OPENMP/GW] Number of threads for self-energy
```

The keyword `DIP` refers to the calculations of the screening matrix elements (also called “dipoles”) needed for the screening function, `X` is the screening function itself (it stands for \(\chi\) since it is a response function), `SE` the self-energy. These three sections of the code can be parallelised independently.

We are running on GPUs. In particular, each node hosts four GPU cards. Yambo is coded in such a way that each MPI *task* is run on a single card, therefore `ntasks=ngpu`.

Note

- In this subsection we are mainly concerned with the **[PARALLEL]** variables which refer to MPI *tasks* (distributed memory).
- What about **[OPENMP]** parallelisation (i.e., adding *threads* with shared memory)? When Yambo is run on GPUs, the explicit threading that you can set in the input and submission script only applies to the very few sections of the code that are *not* run on GPU cards, but stay on the CPUs. Therefore, in a GPU calculation, CPU-only threads are not going to be a relevant factor in the performance of the code. We keep them fixed to 8 since on Leonardo Booster (32 CPUs and 4 GPUs per node) the best hybrid parallel setup *for CPUs* is 4 tasks times 8 threads. We will see an example of the impact of threads in a CPU-only calculation later.

We start by calculating the QP corrections using 4 MPI tasks / GPUs. We leave the number of openMP threads at 8, the optimized value for Yambo on Leonardo. Therefore, edit the submission script as:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
...
#SBATCH --gres=gpu:4
```

and submit the job

```
sbatch job_parallel.sh
```

This will create a new input file and run it. The calculation databases and the human-readable files will be put in separate directories. Check the location of the report `r-*` file and the log `l-*` files, and inspect them while the calculation runs. For simplicity you can just type

```
tail -f run_MPI4_OMP8.out/LOG/l-*_CPU_1
```

to monitor the progress in the master thread (`Ctrl+c` to exit). As you can see, the run takes some time, even though we are using minimal parameters.

Meanwhile, we can run other jobs increasing the parallelisation. Let's employ 16 MPI tasks / GPUs (i.e., 4 nodes on Leonardo). To this end modify the `job_parallel.sh` script changing

```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
...
#SBATCH --gres=gpu:4
```

This time the code should be much faster. Once the run is over, try to run the simulation also on 8 MPI tasks by changing `nodes` appropriately. Finally, you can try to produce a scaling plot.

The timings are contained in the `r-*` report files. You can already have a look at them typing

```
grep Time-Profile run_MPI*/r-*
```

The python script `parse_ytiming.py` is useful for the post-processing of report files. You can already find it in the directory, together with the reports for the longer calculations with 1 and 2 MPI tasks which have been provided.

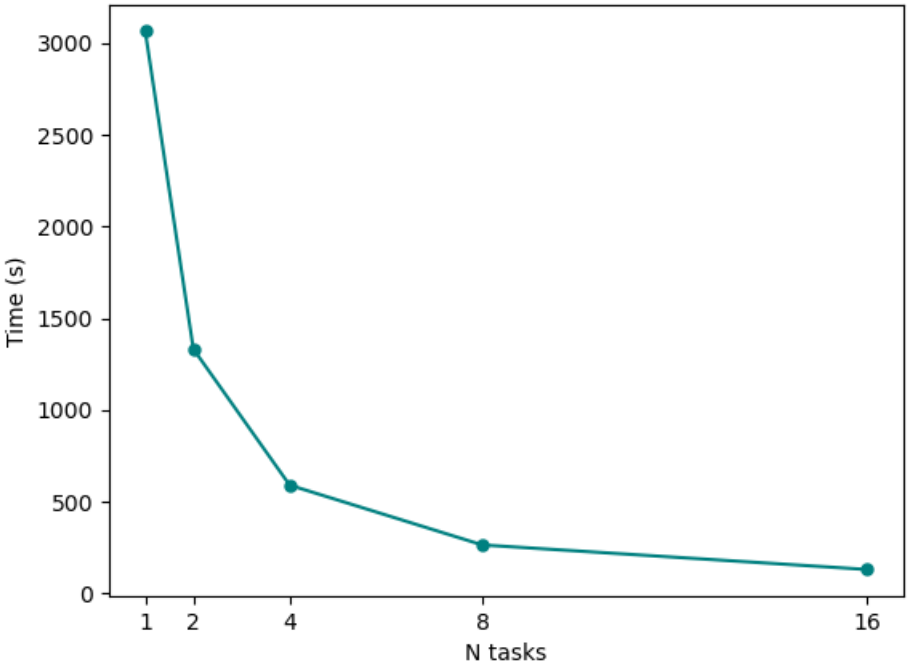
If you didn't do so already, load the python module

```
module load anaconda3/2023.03
```

Then, after your jobs have finished, run the script as

```
python parse_ytiming.py run_MPI
```

to look for a report file in each `run_MPI*.out` folder. **Make sure you have only one report file per folder.** You can also play with the script to make it print detailed timing information, however you should already see that it produced a png plot showing times-to-completion on y axis against number of MPI tasks (i.e., GPUs in this case) on the x axis.



What can we learn from this plot? In particular, try to answer the following questions:

- Up to which number of MPI tasks our system scales efficiently?
- How can we decide at which point adding more nodes to the calculation becomes a waste of resources?

Note

Keep in mind that the MPI scaling we are seeing here is not the true Yambo scaling, but depends on the small size of our tutorial system. In a realistic calculation for a large-sized system, **Yambo has been shown to scale well up to tens of thousands of MPI tasks**! (See the next optional box for an example)

✔ [OPTIONAL] Comparison with CPU calculation with hybrid parallelization strategy

We have run the same calculation using a version of Yambo compiled in order to run on CPUs. This is not the preferred approach in an accelerator-based machine like Leonardo, but it can be instructive.

For a CPU calculation, we can use a hybrid parallel structure with threads. The OPENMP threads are controlled by modifying `cpus-per-task` and `OMP_NUM_THREADS` in the submission file. The product of the number of OpenMP threads and MPI tasks is equal to the total number of CPUs.

For our test, we have used larger convergence parameters than in the previous run, and selected a hybrid parallel scheme with 16 MPI tasks per node, with 2 OPENMP threads (`ntasks*nthreads=ncpu=16*2=32`), since it gives the best scaling in this case.

❗ Note

In general (for larger systems) we have tested that the best CPU scaling on Leonardo is actually 4 MPI tasks times 8 OPENMP threads.

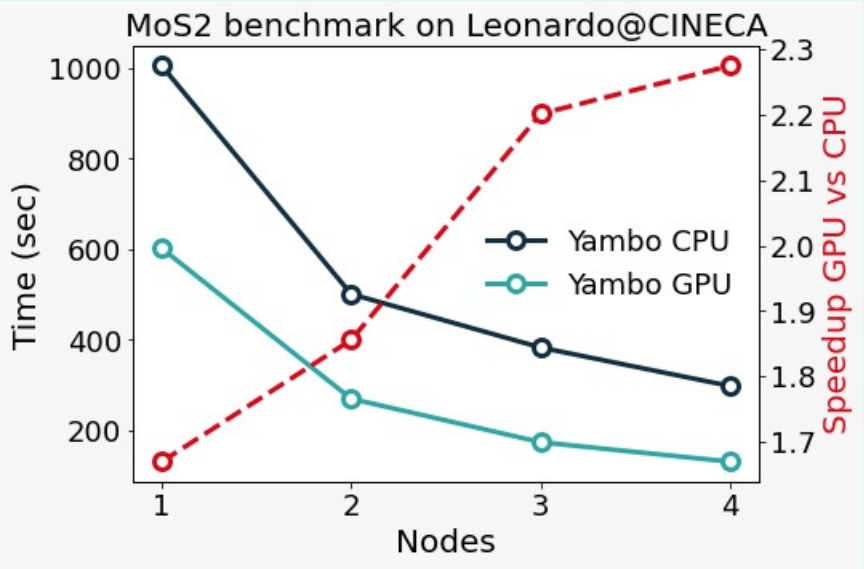
Therefore, in the new CPU submission script we have:

```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=16
#SBATCH --cpus-per-task=2
...
export OMP_NUM_THREADS=2
```

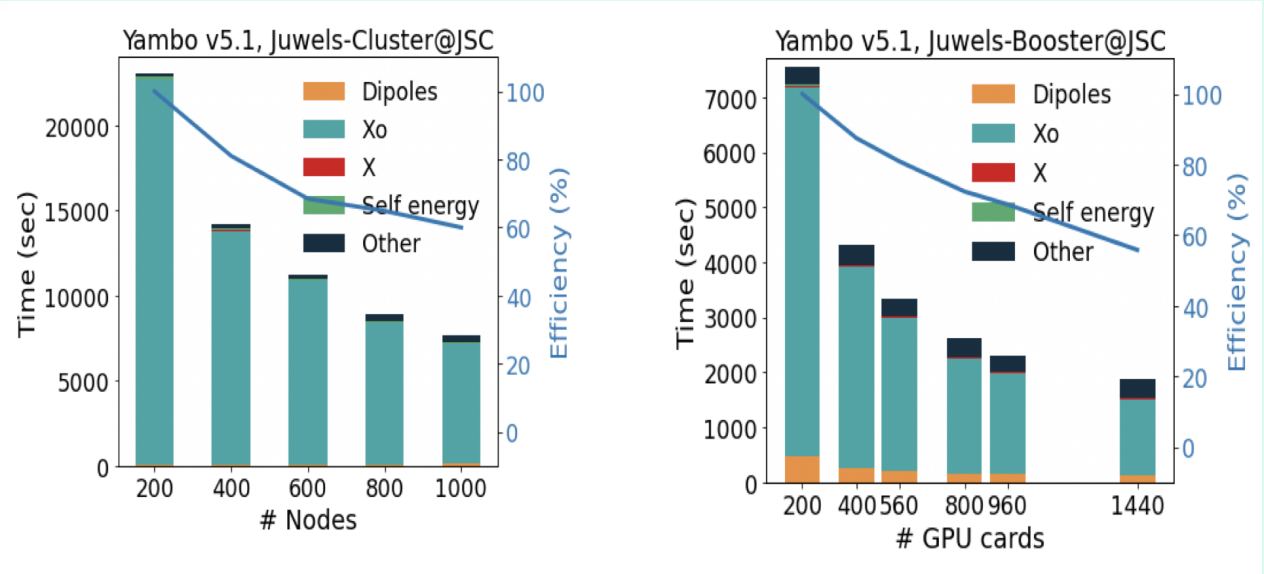
Actually, we don't need to change the openMP-related variables appearing in the yambo input, since the value `0` means "use the value of `OMP_NUM_THREADS` " and we have now set this environment variable to our liking via the submission script. Otherwise, any positive number can directly specify the number of threads to be used in each section of the code.

```
DIP_Threads=  0     # [OPENMP/X] Number of threads for dipoles
...
X_Threads=  0       # [OPENMP/X] Number of threads for response functions
...
SE_Threads=  0      # [OPENMP/GW] Number of threads for self-energy
```

Actively looking for the best scaling on both GPU and CPU for our enlarged MoS2 system we find:



We can see that already for this reasonably small and half-converged system run on a few nodes the GPU calculation easily reaches a speedup of 2x. The speedup vastly increases in larger systems where the calculations are more demanding, as you can see from the scaling tests below (run on the Juwels Booster machine) on a graphene-cobalt interface supercell.



Scaling comparison of graphene@Co(0001) interface on CPU (left, 48 cpus per node) and GPU (right, 4 GPUs per node). Tests done by Nicola Spallanzani. Data available at: http://www.gitlab.com/max-centre/Benchmarks

Note

- In real-life CPU-based calculations running on \(n_{cores} > 100\), as we have seen, it may be a good idea to adopt a hybrid approach.

- The most efficient scaling can depend both on your system and on the HPC facility you're running on. For a full CPU node on Leonardo (32 cores), using a large-scale system, we have found that 4 tasks times 8 threads gives the best performance.
- OpenMP can help lower memory requirements within a node. You can try to increase the OpenMP share of threads if you are getting Out Of Memory errors.

.

✔ [OPTIONAL]: Comparing different parallelisation schemes

Up to now we always parallelised over a single parameter, i.e. `c` or `qp`. However, Yambo allows for tuning the parallelisation scheme over several parameters broadly corresponding to "loops" (i.e., summations or discrete integrations) in the code.

To this end you can open again the `run_parallel.sh` script and modify the section where the yambo input variables are set.

For example, `X_CPU` sets how the MPI Tasks are distributed in the calculation of the response function. The possibilities are shown in the `X_ROLEs`. The same holds for `SE_CPU` and `SE_ROLEs` which control how MPI Tasks are distributed in the calculation of the self-energy.

```
X_CPU= "1 1 1 $ncpu 1"      # [PARALLEL] CPUs for each role
X_ROLEs= "q g k c v"        # [PARALLEL] CPUs roles (q,g,k,c,v)
X_nCPU_LinAlg_INV= $ncpu    # [PARALLEL] CPUs for Linear Algebra
X_Threads=  0               # [OPENMP/X] Number of threads for response functions
DIP_Threads=  0             # [OPENMP/X] Number of threads for dipoles
SE_CPU= "1 $ncpu 1"         # [PARALLEL] CPUs for each role
SE_ROLEs= "q qp b"          # [PARALLEL] CPUs roles (q,qp,b)
SE_Threads=  0              # [OPENMP/GW] Number of threads for self-energy
```

You can try different parallelization schemes and check the performances of Yambo. In doing so, you should also change the jobname `label=MPI${ncpu}_OMP${nthreads}` in the `run_parallel.sh` script to avoid confusion with previous calculations.

You may then check how speed, memory and load balance between the CPUs are affected. You could modify the script `parse_ytiming.py` to parse the new data, read and distinguish between more file names, new parallelisation options, etc.

❗ Note

- The product of the numbers entering each variable (i.e. `X_CPU` and `SE_CPU`) times the number of threads should always match the total number of cores (unless you want to overload the cores taking advantage of multi-threads).

- Using the `X_Threads` and `SE_Threads` variables you can think about setting different hybrid schemes between the screening and the self-energy runlevels.
- Memory scales better if you parallelize on bands (`c v b`).
- Parallelization on k-points performs similarly to parallelization on bands, but requires more memory.
- Parallelization on q-points requires much less communication between the MPI tasks. It may be useful if you run on more than one node and the inter-node connection is slow.

Full GW band structure

This is the final section of the tutorial, in which we want to compute the full correction to the band structure of single-layer MoS\(_2\).

This is a massive calculation, so run it right now and we'll discuss it in the meantime:

```
cd ../04_GW_bands
sbatch gpu_job.sh
```

In order to get somewhat realistic results, we will use the larger values for the convergence parameters we have identified in the convergence section. In addition, we also increased the vacuum separation (to 30 au) and the k-point mesh (to 18x18x1) in the DFT calculation, and of course we consider spin-orbit coupling.

Now we have a heavier calculation, and we have to do it not just for the band gap, but the entire band structure which includes 37 kpoints in the irreducible Brillouin zone, two spin-orbit-split valence bands, and two spin-orbit-split conduction bands. Let us check the new input:

```
vim gw.in
```

```
%QPkrange                       # [GW] QP generalized Kpoint/Band indices
1|37|25|28|
%
```

After about 3 minutes the calculation should be over and the results collected in folder `GW_bnds`. The quasiparticle corrections are stored in human-readable form in the file `o-GW_bnds.QP`, and in netCDF format in the quasiparticle database `ndb.QP`.

In order to visualize the results in the form of a GW band structure, we will first interpolate the calculated points - recall that we have just 37 points, few of which lie on high-symmetry lines - with `ypp`, the yambo pre- and post-processing executable.

Note

We also have a python-based interface for advanced treatment of all the Yambo databases, called Yambopy. You can check it out here on the Yambo wiki.

Let us enter a computing node interactively

```
srun --nodes=1 --ntasks-per-node=1 --gres=gpu:1 --cpus-per-task=8 --mem=490000 --
account=EUHPC_TD02_030 --partition=boost_usr_prod --qos=boost_qos_dbg --time=0:30:00 --
pty /bin/bash
```

and load the yambo module:

```
module load profile/chem-phys
module load yambo/5.2.0--openmpi--4.1.4--nvhpc--23.1
```

We can review the options with `ypp -h` and generate an input file for band structure interpolation with

```
ypp -s b -F ypp_bands.in
```

Let us modify the resulting input file by selecting the ‘boltztrap’ approach to interpolation, the last two valence and first two conduction bands, and a path in the Brillouin zone along the the points \(\Gamma-M-K-\Gamma\). We also set 100 points for each high-symmetry line.

```
electrons                          # [R] Electronic properties
bnds                               # [R] Bands
PROJECT_mode= "none"               # Instruct ypp how to project the DOS. ATOM, LINE,
PLANE.
INTERP_mode= "BOLTZ"                  # Interpolation mode (NN=nearest point,
BOLTZ=boltztrap aproach)
INTERP_Shell_Fac= 20.00000         # The bigger it is a higher number of shells is used
INTERP_NofNN= 1                    # Number of Nearest sites in the NN method
OutputAlat= 5.90008               # [a.u.] Lattice constant used for "alat" ouput format
cooIn= "rlu"                       # Points coordinates (in) cc/rlu/iku/alat
cooOut= "rlu"                      # Points coordinates (out) cc/rlu/iku/alat
% BANDS_bands
   25 | 28 |                           # Number of bands
%
CIRCUIT_E_DB_path= "none"          # SAVE obtained from the QE `bands` run (alternative
to %BANDS_kpts)
BANDS_path= ""                     # High-Symmetry points labels (G,M,K,L...) also using
composed positions (0.5xY+0.5xL).
BANDS_steps= 100                    # Number of divisions
#BANDS_built_in                  # Print the bands of the generating points of the
circuit using the nearest internal point
%BANDS_kpts                        # K points of the bands circuit
 0.00000 |0.00000 |0.00000 |
 0.00000 |0.50000 |0.00000 |
 0.33333 |0.33333 |0.00000 |
 0.00000 |0.00000 |0.00000 |
%
```

Now, let's run ypp:

```
mpirun -np 1 ypp -F ypp_bands.in
```

This run will produce the file `o.bands_interpolated`. You can inspect it and see that it contains a plottable band structure, but beware: these are the DFT eigevalues! We didn't tell `ypp` where to look for the quasiparticle corrections, so it went into the `SAVE` folder and interpolated the DFT data. Let's rename the output:

```
mv o.bands_interpolated o.DFT_bands
mkdir DFT_bands
mv o.spin* o.magn* DFT_bands/
```

In order to interpolate the quasiparticle database, we append its location to the `ypp` input:

```
vim ypp_bands.in
```

add this line at the end

```
...
GfnQPdb= "E < ./GW_bnds/ndb.QP"
```

and run `ypp` again.

```
mpirun -np 1 ypp -F ypp_bands.in
```

When it's done, let's rename the new output as

```
mv o.bands_interpolated o.GW_bands
mkdir GW_bands
mv o.spin* o.magn* GW_bands/
```

Now we are ready to visualize the band structures. In order to do so, you can use the script `plt_bands.py` that should be already available in the directory.

We load the python module

```
module load anaconda3/2023.03
```

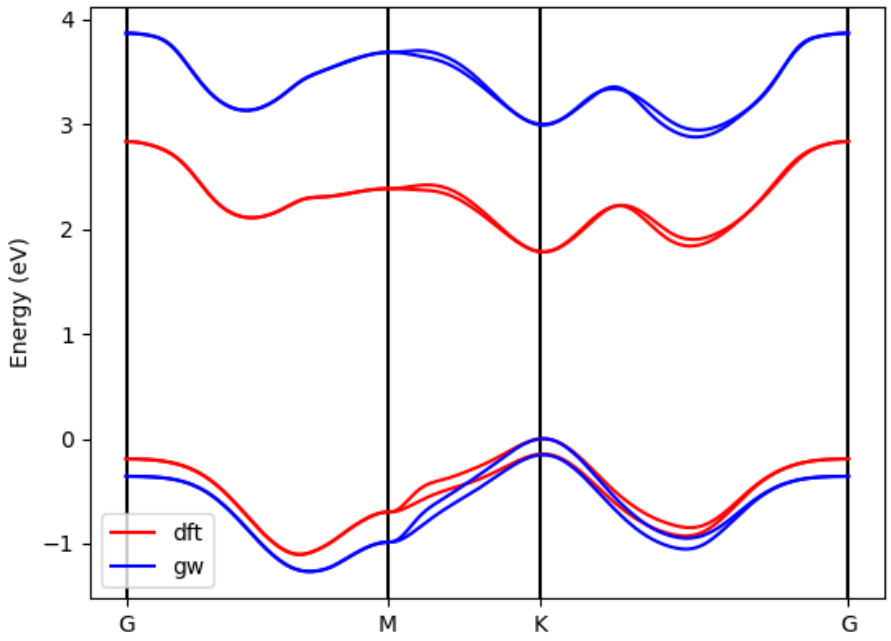
and run the script as

```
python plt_bands.py o.DFT_bands o.GW_bands 4
```

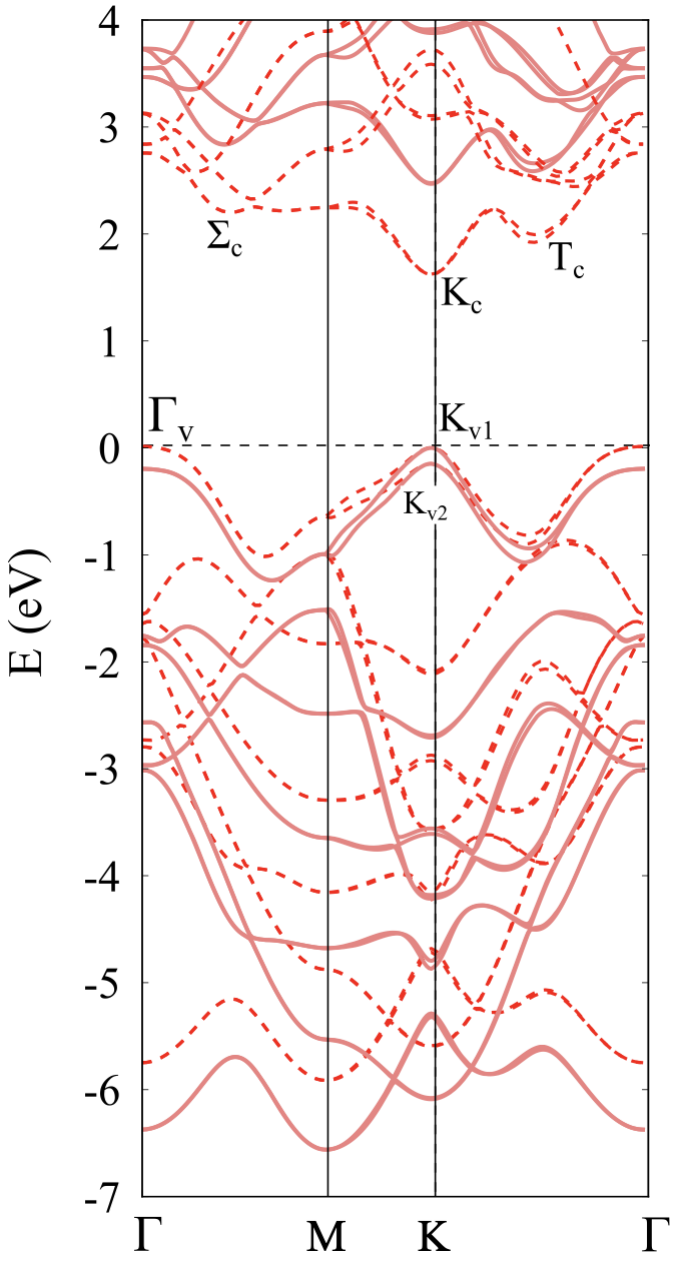
Now we can also exit the computing node

```
exit
```

The python script should have produced a `GW_bands.png` file containing the following visualization, which you can copy and open it in your local machine using `scp`:



You may compare this plot with a converged result from this paper (also done with Yambo):



Dashed lines: DFT, thick lines: GW.

As you can see, the general result is not too bad, but there are some differences both at the DFT and GW levels. The magnitude of the band gap is too large, and the relative energy of the two conduction band minima is not correct. One obvious issue is the lack of convergence of our tutorial calculations. As we know, we should include more vacuum space and many, many more k-points. Additionally, this is a transition metal dichalcogenide: for this class of systems, the details of the band structure can strongly depend on small variations in the lattice parameters and on the type of pseudopotential used. A great deal of care must be taken when performing these calculations!

In order to learn more about Yambo, we suggest visiting the Yambo website. For technical information and tutorials, you can check out the Yambo wiki. If you have issues and questions about installing and running the code, you can write about them on the Yambo forum.

Quick Reference

Instructor's guide

Why we teach this lesson

Intended learning outcomes

Timing

Preparing exercises

e.g. what to do the day before to set up common repositories.

Other practical aspects

Interesting questions you might get

Typical pitfalls

About the course

In this workshop, participants will learn how to launch the most common types of calculations (e.g. scf, phonons, quasi-particle energies, time-dependent properties) using QE, SIESTA and Yambo, how to prepare input files and how to read output files in order to extract the desired properties.

Best practices for efficient exploitation of HPC resources will be discussed. On the QE days, there will be particular emphasis on how to use the different schemes of data distribution (e.g. plane waves, pools, images) in combination with the different parallelization and acceleration schemes (MPI, OpenMP, GPU-offload) available. Regarding SIESTA, participants will learn about its algorithmic efficiency, and they will be able to explore its accelerated and massively parallel solvers.

Schedule

Day 1, QUANTUM ESPRESSO

Time	Section
09:00-09:15	Welcome and introduction to ENCCS
09:15-09:30	Introduction to Max-CoE and MaX flagship codes
09:30-10:00	Overview of the QE suite of codes and main features
10:00-10:25	Coffee break
10:25-13:00	PWSCF for HPC and GPU

Day 2, QUANTUM ESPRESSO

Time	Section
09:00-09:45	Introduction to Density Functional Perturbation Theory
09:45-10:15	Introduction to Time Dependent Density Functional Perturbation Theory
10:15-10:30	Coffee break
10:30-13:00	Phonons and time dependent properties on HPC and GPU

Day 3, SIESTA

Time	Section
09:00-09:45	SIESTA basics
09:45-10:30	Hands-on tutorial: A first contact with SIESTA: inputs, execution and outputs
10:30-10:45	Break
10:45-11:15	Basis sets
11:15-12:00	Hands-on tutorial: Basis set optimization
12:00-13:00	Hands-on tutorial: Convergence (k points, mesh, mixing)

Day 4, SIESTA

Time	Section
09:00-10:00	Hands-on tutorial: Moving atoms: geometry optimisation and beyond
10:30-11:00	Hands-on tutorial: Analysis tools
11:00-11:15	Break
11:15-11:45	Features available in SIESTA: spin-orbit couplings, TranSIESTA, and others
11:45-13:00	Hands-on tutorial: Pushing the boundaries of SIESTA: accelerated and massively parallel solvers

Day 5, Yambo

Time	Section
09:00-09:20	Overview of the Yambo code and its main features and performance
09:20-10:00	Introduction to the GW approximation
10:00-10:20	Coffee break

Time	Section
10:20-13:00	Hands-on tutorial: A guided tour through GW simulations

See also

- ENCCS: https://enccs.se/
- MAX-CoE: http://www.max-centre.eu/
- Follow ENCCS on LinkedIn, or Twitter
- Follow MAX-CoE on LinkedIn, or Twitter.

\[\]

Credits

Contributors to this workshop:

- Pietro Davide Delugas (SISSA)
- Ivan Carnimeo (SISSA)
- Oscar Baseggio (SISSA)
- Fabrizio Ferrari Ruffino (CNR-IOM)
- Paolo Giannozzi (CNR-IOM, UniUD)
- Iurii Timrov (Paul Scherrer Institut)
- Laura Bellentani (CINECA)
- Tommaso Gorni (CINECA)
- Aurora Ponzi (CNR-IOM)
- Emilio Artacho (CIC NanoGUNE and University of Cambridge)
- Catalina Coll (ICN2)
- José Mª Escartín (ICN2)
- Roberta Farris (ICN2)
- Ernane de Freitas (ICN2)
- Alberto García (ICMAB-CSIC)
- Arnold Kole (Utrecht University)
- Nick Papior (DTU)
- Federico Pedron (ICN2)
- Miguel Pruneda (CINN-CSIC)
- José Ángel Silva Guillén (IMDEA Nanociencia)
- D. Varsano (CNR-NANO)
- A. Ferretti (CNR-NANO)
- D. Sangalli (CNR-ISM)
- A. Guandalini (Univ. of Rome, La Sapienza)
- F. Paleari (CNR-NANO)
- M. D'Alessio (Univ. Modena and Reggio Emilia, CNR-NANO)
- G. Sesti (CNR-NANO)
- N. Spallanzani (CNR-NANO)

The lesson file structure and browsing layout is inspired by and derived from work by CodeRefinery licensed under the MIT license. We have copied and adapted most of their license text.

Licenses

General

Instructional Material

Software

SIESTA