exciting: porting a full all-electron full-potential DFT code to GPU

M. Raya-Moreno, M. Hossain, B. M. Maurer

Institut für Physik und CSMB Adlershof, Humboldt-Universität zu Berlin

14.Oct.2024, Hackathon: Optimizing for AMD GPUs (Brussels)

`exciting` in a nutshell

`exciting` is an open-source all-electron, full-potential package for first-principles electronic-structure calculations, using variants of the LAPW+lo method to achieve μHa precision, with a special emphasis on excited-state properties such as G_0W_0, BSE, XAS, XES, TD-DFT, and RTD-DFT.

Get the code at https://exciting-code.org

`exciting` in a nutshell (technical details)

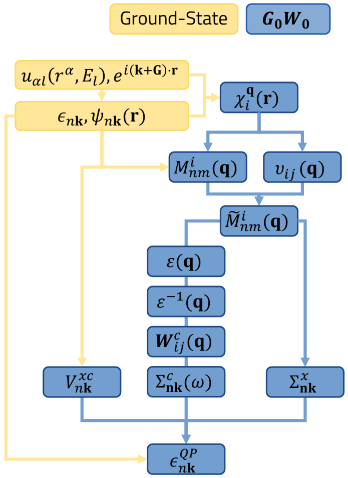
- **Licensing provisions:** GPL2, some components are provided under Apache 2.0.
- **Programming language:** `Fortran` 2018
- **Parallelization:** MPI+OpenMP
- **Dependencies:** FFTW3, LAPACK+BLAS (OpenBLAS/IntelMKL/LibSci/...), libXC, FoX, MPI, ScaLAPACK, HDF5, SIRIUS.
- **Build system:** make and CMake (WIP).
- **Compiler support:** GNU, Intel LLVM, Cray (WIP).
- **Testing:** Regression tests cover a significant portion of the code, and unit tests are used for more recent parts. Both are integrated continuously (CI) in our development process.

exciting and G_0W_0

exciting has a very large source with 286,139 lines of code in more than 1,356 files with lots of functionalities. We choose G_0W_0as a starting point for porting the code to GPU.

What is G_0W_0?
G_0W_0 is the state of the art approach for calculating precise quasi-particle band structures for crystals and molecules based on the many-body perturbation theory approach. $G_0W_0$is very costly to calculate, with a scaling of $\mathcal{O}(N^4)$. Here we see the largest potential for acceleration by porting to GPUs.

exciting-G_0W_0: workflow, scaling, and parallelization



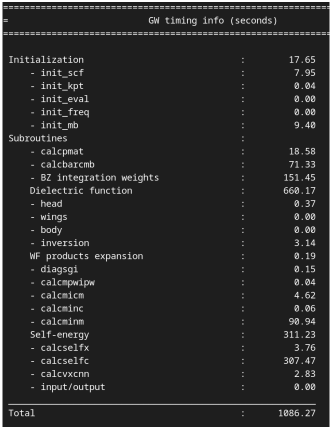
Ground-State
G_0W_0
$u_{\alpha l}(r^\alpha, E_l), e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}$
$\epsilon_{n\mathbf{k}}, \psi_{n\mathbf{k}}(\mathbf{r})$
$\chi_i^{\mathbf{q}}(\mathbf{r})$
$M_{nm}^i(\mathbf{q})$
$v_{ij}(\mathbf{q})$
$\widetilde{M}_{nm}^i(\mathbf{q})$
$\varepsilon(\mathbf{q})$
$\varepsilon^{-1}(\mathbf{q})$
$W_{ij}^c(\mathbf{q})$
$V_{n\mathbf{k}}^{xc}$
$\Sigma_{n\mathbf{k}}^c(\omega)$
$\Sigma_{n\mathbf{k}}^x$
$\epsilon_{n\mathbf{k}}^{QP}$

Flowchart of exciting's G_0W_0 implementation.

- **Computational cost:** high due to many matrix-matrix products, diagonalizations,...
- **Scaling:** $\mathcal{O}(N^4)$ where N is the system size.
- **MPI parallelization:** **q**/**k**-points are scattered with low communication.
- **OpenMP parallelization:** linear algebra and intensive do-loops.

Where to start porting?

- **Dielectric matrix:** Ca. 30-50% of the runtime is spent here.
- **Expansion coefficients:** Ca. 10% of the runtime is spent here. Can be up to 40% if full band set correction is required.



```
==============================================================
=                   GW timing info (seconds)
==============================================================

Initialization                                    :       17.65
    - init_scf                                    :        7.95
    - init_kpt                                    :        0.04
    - init_eval                                   :        0.00
    - init_freq                                   :        0.00
    - init_mb                                     :        9.40
Subroutines                                       :
    - calcpmat                                    :       18.58
    - calcbarcmb                                  :       71.33
    - BZ integration weights                      :      151.45
    Dielectric function                           :      660.17
    - head                                        :        0.37
    - wings                                       :        0.00
    - body                                        :        0.00
    - inversion                                   :        3.14
    WF products expansion                         :        0.19
    - diagsgi                                     :        0.15
    - calcmpwipw                                  :        0.04
    - calcmicm                                    :        4.62
    - calcminc                                    :        0.06
    - calcminm                                    :       90.94
    Self-energy                                   :      311.23
    - calcselfx                                   :        3.76
    - calcselfc                                   :      307.47
    - calcvxcnn                                   :        2.83
    - input/output                                :        0.00
 ____________________________________________________________
Total                                             :     1086.27
```

Timings of an `exciting`-G_0W_0run for ZrO_2 on a 2x2x2 **q**-mesh, with 800 empty bands, and corrected for 22 bands. The runs were performed on LUMI-G using 8 MPI processes, each with 7 threads. Note that this calculation is not converged with respect to the **q**-mesh.

Status of the GPU porting: Strategy

GPU offload prototype:

- Each MPI rank is associated with one GPU, and has the queues (streams) initialized.
- Data transfer and GPU memory control are implemented with OpenMP.
- Partial porting of simple loops is implemented with OpenMP.
- Expensive matrix-matrix products call MAGMA or (offloaded) Intel MKL routines.
- Code compiles with GNU compilers (NVIDIA and AMD GPUs) and ifx (Intel GPUs). Ongoing efforts are directed towards Cray compiler support.

Status of the GPU porting: initial results

Initial benchmark for the ported parts: ZrO_2 on a 2x2x2 **q**-mesh, with 800 empty bands, and corrected for 22 bands:

- **Intel@ifx** – *2 Intel(R) Xeon(R) Platinum 8480L + 4 Intel Data Center GPU Max 1550 MI250X GPUs*:
  a) 8 MPI processes with 28 threads and 1 GPU each: 17%

- **AMD@gfortran** – *AMD EPYC "Trento" CPU with 8 AMD MI250X GPUs, i.e. LUMI-G*:
  a) 1 MPI process with 7 OpenMP threads and 1 GPU: 20%
  b) 8 MPI processes with 7 threads and 1 GPU each: 36% in the master process, degradation in the others.

Objectives in this Hackathon

- Enable the compilation with offloading to GPUs with Cray compilers.
- Port the complex loop that computes the expansion coefficients (currently implemented using OpenMP for CPU execution).
- Analyze data transfers and identify areas for future optimization.
- Investigate why we see no acceleration in the slave processes, but we do in the master process, during multi-rank MPI runs when the code is compiled with GNU compilers.

Thanks for your attention