Accelerating Large-Scale GW Calculations on Hybrid CPU-GPU Architectures

BerkeleyGW: Algorithm Development and Code Optimization

M. Del Ben1, C. Yang1, F.H. da Jornada1,2, S.G. Louie1,2 and J. Deslippe3
1Lawrence Berkeley National Laboratory, 2University of California at Berkeley, 3National Energy Research Scientific Computing Center (NERSC)

The Method

BerkeleyGW is a massively parallel software package for studying the electron excited-state properties of materials employing the GW and Bethe-Salpeter equation approach and beyond. The dynamical properties of the electrons are obtained as solution of the Solov’ Dyson’s equation:

\[ \Sigma(E) = \text{self-energy (non-Hermitian, non-local, energy-dependent operator)} \]

- Perturbative expansion on the screened Coulomb interaction \( W \rightarrow \) At first order \( \Sigma = -iGW \)
- \( W \) obtained from the inverse dielectric matrix \( \epsilon^{-1} \) of the system

In BerkeleyGW [1]:
- Epsilon code: Compute \( \epsilon^{-1} \), Obtain \( \Sigma(N^*) \)
- Sigma code: Compute \( W \) from \( \epsilon^{-1} \) and solve Eq. 1, Obtain \( \Sigma(N^*) \)

Epalsin code: Inverse Dielectric Matrix \( \epsilon^{-1} \) and its Frequency Dependence

Static Dielectric Matrix \( \epsilon^{-1}(\omega = 0) \), input: \( \omega \) from \( \epsilon \), \( \omega \) is at \( \epsilon \) energy, \( \omega \) to \( \epsilon \) matrix, \( \epsilon \) is \( \omega \) matrix, \( \epsilon \) is \( \omega \) matrix, \( \omega \) to \( \epsilon \) matrix. 

1. Calculate plane-waves matrix elements: \( O(N, N_0) \rightarrow \text{MTXEL kernel} \). Matrix elements are calculated by many node-local fast Fourier transforms (FFTs), matrix elements are distributed in a tall and skinny matrix \( M \).
2. Static polarization \( \omega = 0 \): Obtain \( O(N, N_0) \rightarrow \text{CHI-0 kernel} \). Large distributed matrix-multiplication between tall and skinny matrices.
3. Dielectric matrix inversion: \( \epsilon^{-1}(\omega) \rightarrow \text{Invert (ScalAPACK)} \). Algebraic inversion (LU factorization + triangular inversion) of a square complex matrix

Frequency Dependence \( \omega \rightarrow \omega \) for \( \omega \neq 0 \) within the static subspace approximation [2,3,4], input: \( M \) from step 1 and \( \omega \) from step 2

1. Eigendecomposition \( \Omega = \text{Diag}(\epsilon) \rightarrow \text{Obtain \( \epsilon \) from \( \omega \)} \).
2. Basis transmision of \( M \) into the subspace spanned by \( \Omega \) from \( O(N, N_0) \rightarrow \text{Transf kernel} \). 

Distribute matrix multiplication: \( M \rightarrow M \times \text{Diag}(\epsilon) \rightarrow \text{Diag (ScalAPACK, ELPA)} \).
3. For \( \omega 
eq 0 \) : direct computation of \( \Omega(N_0, N) \rightarrow \text{CHI-freq kernel} \). 

Distribute matrix multiplication at multiple frequencies: \( \Omega(N_0, N) \rightarrow \Delta(\omega) \Omega(N_0, N) \) with \( \Delta(\omega) \) diagonal matrix containing the band energy/frequency dependence
4. Final evaluation of \( \epsilon^{-1}(\omega) \) from \( \Omega(N_0, N) \) in the subspace basis \( O(N_0, N_2) \rightarrow \text{Invert (SubscalAPACK)} \).

Gpu Elements Kernel \( \rightarrow \text{MTXEL} \)
- Double loop over valence (outer) and conduction (inner) wavefunctions/bands

GPU Support for Computational Kernels

- Use cuFFT library to perform the fast Fourier transforms.
- Use data streams and host pinned memory, one stream for each conduction wavefunction (FFT boxes) offloaded to device over batches to avoid out of memory.
- CUDA kernels for Put/Multiply/Get to keep intermediates on device.

Static Polarizability Kernel \( \rightarrow \text{CHI-0} \)
- Large distributed matrix multiplication

Batch communication over eigenvector index to control memory usage on device

Basis Transformation \( \rightarrow \text{Transf} \)
- Similar communication scheme as CHI-0, potential memory bottlenecks for both host/device
- All eigenvectors need to be communicated to all MPI tasks
- Batch communication over eigenvector index to control host memory usage

Batch computation over (valence) wavefunction index to control device memory usage

Frequency-dependent Polarizability Kernel \( \rightarrow \text{CHI-freq} \)
- Similar communication scheme as CHI-0
- Many matrix multiplication at multiple frequencies, matrices smaller than CHI-0

- Data streams over frequency index to concurrent use GPU+CPU execution on device
- Batch over (conduction) wavefunction index to control memory usage on device

Benchmarks for Performance Measurement

Table: (1) Computational cost associated with each of the kernels of the epsilons code. The parameters in the table are \( N_0 \), number of valence bands, \( N_2 \), number of conduction (empty) bands, \( N_3 \), plane wave basis set size, \( N_{\text{sub}} \), subspace basis size, and \( N_4 \), number of frequencies.

References


Summary

- BerkeleyGW large-scale GW calculations are moving to GPUs!
- 14x speedup obtained from GPU acceleration and lots of lessons learned during optimization!
- Excellent time to solution (minutes) is achieved for systems made of thousands of atoms!
- More efforts are needed to optimize parallel 1/0 and accelerate libraries for eigensolvers and matrix inversions!

Acknowledgments

This work was supported by the Center for Computational Study of Electrified State Phenomena in Energy Materials at the Lawrence Berkeley National Laboratory, which is funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division under Contract No. DE-AC02-05CH11231, as part of the Computational Materials Sciences Program. This research used resources of the NERSC and OLCF which are supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231 and DE-AC05-00OR22725.

CPU Only vs CPU+GPU

Comparison between the CPU-only and CPU+GPU executions measured on Titan and Summit (OLCF). The right plot is the zoom-in version of the left. All optimized kernels show great acceleration from GPU support, with a 14x speedup for the overall execution on Summit.

Strong and Weak Scaling on Summit/OLCF

Left: good parallel efficiency depending on system size, scaling to almost 5,000 GPUs (~17% of full machine). Parallel I/O issues for large scale calculation (HDF5 library). Right: good weak scaling; as problem size increases, memory grows to \( O(N) \) and flops increases to \( O(N^2) \) more memory available for larger systems, less communication and better parallel performance.

Comparison Across Architectures: Time vs Power

Comparing performance in term of energy efficiency between Edison@NERSC and Summit@OLCF. Ideal scaling (dotted lines) → constant energy consumption increasing power. GPUs are 14x more power efficient than CPUs consistently through all three benchmarks.