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

BerkeleyGW: Algorithm Development and Code Optimization

M. Del Ben[1], C. Yang[3], F.H. da Jornada[1,2], S.G. Louie[1,2] and J. Deslippe[3]

[1]Lawrence Berkeley National Laboratory, [2]University of California at Berkeley, [3]National Energy Research Scientific Computing Center (NERSC)

The *GW* 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 electron are obtained as solution of the Solve Dyson's equation:

$$\left[-\frac{1}{2}\nabla^2 + V_{\rm KS} + \Sigma(E_n)\right]\phi_n = E_n\phi_n, \qquad (1)$$

$\Sigma(E_n) \rightarrow$ 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 ϵ^{-1} of the system

In `BerkeleyGW` [1]:

- `Epsilon` code $\rightarrow$ Compute ϵ^{-1}, $O(N^4)$
- `Sigma` code $\rightarrow$ Compute W from ϵ^{-1} and solve Eq.1, $O(N^4)$

`Epsilon` code: Inverse Dielectric Matrix ϵ^{-1} and its Frequency Dependence

Static Dielectric Matrix $\epsilon^{-1}(\omega = 0)$, input: $\psi_{m\mathbf{k}}$ wavefunctions and $\epsilon_{m\mathbf{k}}$ energies

1. Calculate plane-waves matrix elements: $O(N_v N_c N_G \log N_G) \rightarrow$ `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 polarizability $\chi(\omega = 0)$: $O(N_v N_c N_G^2) \rightarrow$ `CHI-0` kernel.
   Large distributed matrix-multiplication between tall and skinny matrices.
3. Dielectric matrix ϵ and inversion: $O(N_G^3) \rightarrow$ `Inv` (ScaLAPACK)
   Algebraic inversion (LU factorization + triangular inversion) of a square complex matrix

Frequency Dependence $\epsilon^{-1}(\omega_i)$ **for** $\omega_i \neq 0$ within the static subspace approximation [2,3,4], input: **M** from step 1 and $\chi(\omega = 0)$ from step 2

1. Eigendecomposition $\bar{\chi}(0) = \mathbf{C}^{0\dagger}\mathbf{x}\mathbf{C}^0$, define define new subspace basis $\mathbf{C}_s^0$ of size N_{eig} according $t_{\rm eigen}$ threshold $O(N_G^3) \rightarrow$ `Diag` (ScaLAPACK, ELPA)
2. Basis transormation of **M** into the subspace spanned by $\mathbf{C}_s^0$ $O(N_v N_c N_{eig} N_G) \rightarrow$ `Transf` kernel
   Distribute matrix multiplication: $\overline{\mathbf{M}}_s^0 = \mathbf{M} v^{\frac{1}{2}} \mathbf{C}_s^0$ ($\mathbf{v}^{\frac{1}{2}}$ diagonal)
3. For $\omega_i \neq 0$: direct computation of $\bar{\chi}_s(\omega_i)$ $O(N_\omega N_v N_c N_{eig}^2) \rightarrow$ `CHI-freq` kernel
   Distribute matrix multiplication at multiple frequency: $\bar{\chi}_s(\omega_i) = \overline{\mathbf{M}}_s^{0\dagger} \mathbf{\Delta}(\omega_i) \overline{\mathbf{M}}_s^0$ with $\mathbf{\Delta}$ diagonal matrix containing the band energy/frequency dependence
4. Final evaluation of $\bar{\epsilon}^{-1}(\omega_i)$ from $\bar{\chi}_s(\omega_i)$ in the subspace basis $O(N_\omega N_{eig}^3) \rightarrow$ `Inv-Sub` (ScaLAPACK)

	Execution	Memory
Matrix Element (`MTXEL`)	$O(N_v N_c N_G \log N_G)$	$O(N_v N_c N_G)$
Polarizability $\omega = 0$ (`CHI-0`)	$O(N_v N_c N_G^2)$	$O(N_G^2)$
$\mathbf{C}_s^0$ or ϵ^{-1} (`Diag` / `Inv`)	$O(N_G^3)$	$O(N_G^2)$
Basis transformation: $\overline{\mathbf{M}}_s^0$ (`Transf`)	$O(N_{eig} N_v N_c N_G)$	$O(N_v N_c N_{eig})$
Polarizability $\omega \neq 0$ (`CHI-freq`)	$O(N_\omega N_v N_c N_{eig}^2)$	$O(N_\omega N_{eig}^2)$
Inversion (`Inv-Sub`)	$O(N_\omega N_{eig}^3)$	$O(N_\omega N_{eig}^2)$
I/O	$O(N_G N_{eig} + N_\omega N_{eig}^2)$	$O(N_G N_{eig} + N_\omega N_{eig}^2)$

Table: (1) Computational cost associated with each of the kernels of the `epsilon` code. The parameters in the table are N_v number of valence bands, N_c number of conduction (empty) bands, N_G plane waves basis set size, N_{eig} subspace basis size, and N_ω number of frequencies.

References

1. J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, Comput. Phys. Commun. 183, 1269 (2012).
2. M. Del Ben, F.H. da Jornada, A. Canning, N. Wichmann, K. Raman, R. Sasanka, C. Yang, S.G. Louie and J. Deslippe, Comput. Phys. Commun. 235, 187-195 (2019).
3. M. Del Ben, H. Felipe, G. Antonius, T. Rangel, S.G. Louie, J. Deslippe and A. Canning, Phys. Rev. B 99 (12), 125128 (2019).
4. M. Govoni and G. Galli, J. Chem. Theory Comput. 11, 2680 (2015).

GPU Support for Computational Kernels

Matrix Elements Kernel $\rightarrow$ `MTXEL`

- Double loop over valence (outer) and conduction (inner) wavefunctions/bands

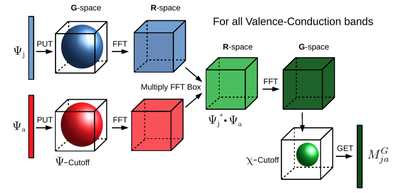


Figure: Schematic representation of the operations performed in the inner loop of the `MTXEL` kernel.

- Use cuFFT library to perform the fast Fourier transforms.
- Use data streams and host pinned memory, one stream for each conduction wavefunction (inner loop index) $\rightarrow$ asynchronous memory transfer + high concurrency on device .
- Conduction bands (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 $\rightarrow$ `CHI-0`

- Large distributed matrix multiplication

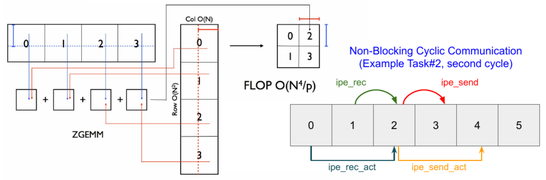


Figure: Schematic representation of the data layout and communication scheme for the `CHI-0` kernel.

- Use cuBLAS to perform the local ZGEMM
- Data stream and host pinned memory for the matrix buffers $\rightarrow$ asynchronous memory transfer and calls to cuBLAS
- Non-blocking cyclic communication scheme: communication is finalized before stream is synchronized $\rightarrow$ overlap communication (CPU) computation (GPU)
- Batch over conduction wavefunction index to control memory usage on device

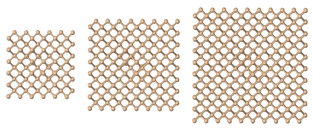
Basis Transformation $\rightarrow$ `Transf`

- Similar communication scheme as `CHI-0`, potential memory bottlenecks for both host/device $\rightarrow$ all eigenvectors need to be communicated to all MPI tasks
- Batch communication over eigenvector index $\rightarrow$ control host memory usage
- Batch computation over (valence) wavefunction index $\rightarrow$ control device memory usage

Frequency-dependent Polarizability Kernel $\rightarrow$ `CHI-freq`

- Similar communication scheme as `CHI-0`
- Many matrix multiplication at multiple frequencies, matrices smaller than `CHI-0` ($N_G/N_{eig} \simeq 5-10$)
- Data streams over frequency index $\rightarrow$ concurrent cuZGEMM execution on device
- Batch over (conduction) wavefunction index to control memory usage on device

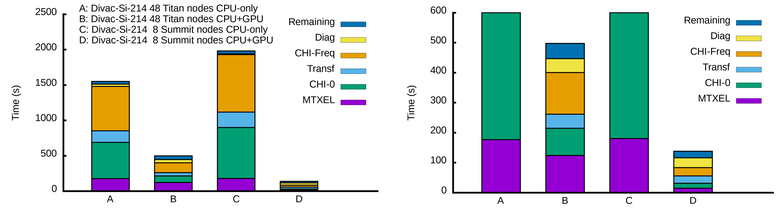
Benchmarks for Performance Measurement



	Dvac-Si-214	Dvac-Si-510	Dvac-Si-998
N_G^ψ	31,463	74,653	145,837
N_G	11,075	26,529	51,627
N_b	6,397	15,045	29,346
N_v	428	1,020	1,996
N_c	5,969	14,025	27,350
N_{eig}	3,500	7,000	14,000
N_ω	10	10	10
Epsilon Min PFlops	5.8	157.9	2335.7
Epsilon Min Memory (Tb)	0.6	7.7	57.5

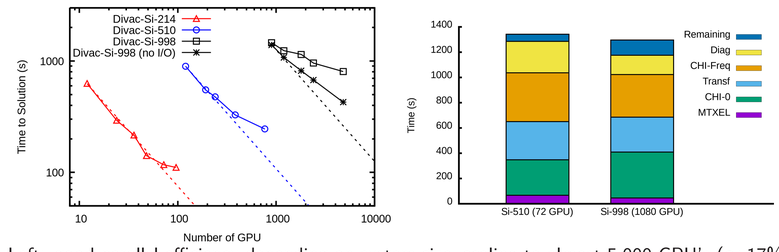
Benchmark systems for performance measurement. The benchmarks are based on a silicon divacancy defect in silicon (a prototype of a solid state qbit). The table shows the computational characteristics for each benchmark, see Table (1) for the actual computational cost.

CPU-Only vs CPU+GPU



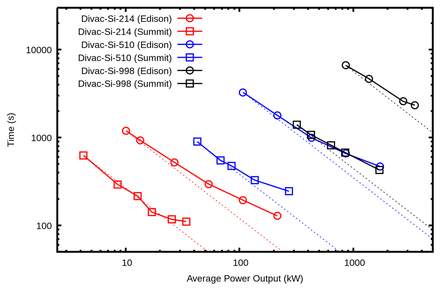
Comparison between the CPU-only and CPU+GPU executions measured on Titan and Summit (OLCF). The right plot is the zoom-in version of the left. All optimized kernels show great acceleration from GPU support, with a 14x speedup for the overall execution on Summit.

Strong and Weak Scaling on Summit@OLCF



Left: good parallel efficiency depending on system size, scaling to almost 5,000 GPU's ($\simeq$ 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 $O(N^3)$ and flops increases to $O(N^4) \rightarrow$ 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) $\rightarrow$ constant energy consumption increasing power. GPUs are 16x more power efficient than CPUs consistently through all three benchmarks.

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 I/O and accelerate libraries for eigensolvers and matrix inversions!

Acknowledgments

This work was supported by the Center for Computational Study of Excited-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 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.