# Achieving Performance Portability on Hybrid GPU-CPU Architectures for a Large Scale Materials Science Code: the BerkeleyGW Case Study

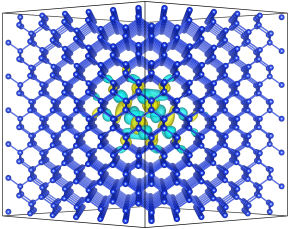
Mauro Del Ben

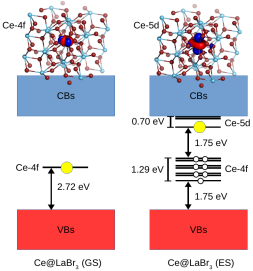
Computational Research Division, Lawrence Berkeley National Laboratory

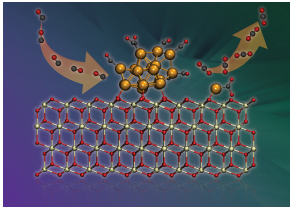
September 2, 2020

2020 Performance, Portability, and Productivity in HPC Forum
01-02 September 2020, Virtual

# Materials Science/Chemistry at Exascale





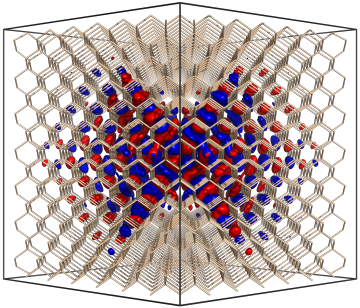



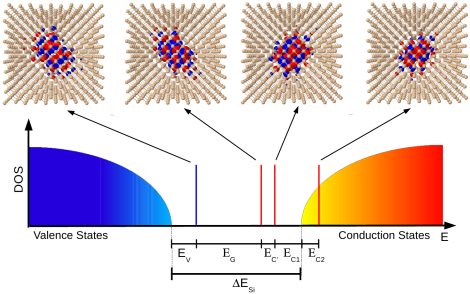
Mat. Sci & Chem apps, such as VASP, Quantum ESPRESSO, NWChem, GAMESS, QMCPACK, BerkeleyGW, and CP2K are among the most heavily used DOE facilities.

**Used to design and understand the fundamental components of novel devices**

- Applications: Quantum Computers, Batteries, Photovoltaics, Catalysis, Hydrogen Storage, Carbon Sequestration, etc...
- Density Functional Theory (DFT): computational workhorse for over three decades

# Accurate Electronic Excited-State Properties of Materials: Beyond DFT






Example: Divacancy point defect in crystalline silicon, prototype of a solid-state QBit.

**Accurate predictions requires:**

- Accuracy beyond standard (DFT) approaches $\rightarrow$ $GW$ and $GW$ + BSE
- System size beyond previous simulations $\rightarrow$ Many thousands of atoms

# The $GW$ Method: State of the Art

The $GW$ method represents one of the most effective and accurate approach to predict excited-state properties in a wide range of materials

Application of $GW$ to thousands atoms systems still a challenge

Reduce time to solution and extend applicability:

- Develop methods to reduce prefactor and scaling with system size
- Improve performance of existing implementation

**HPC on the Path to Exascale → GPUs**

In this talk the portability strategies employed to implement GPU support for the BerkeleyGW software package will be discussed.

# The BerkeleyGW Software Package

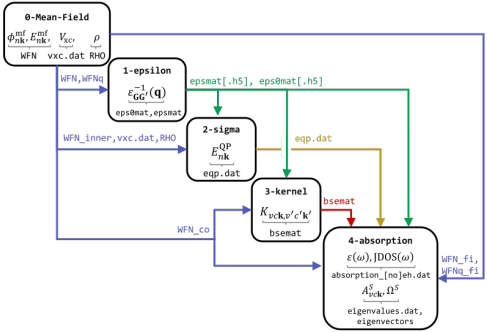
BerkeleyGW

https://berkeleygw.org/

Compute the electronic excited-state properties of materials via $GW$, Bethe-Salpeter equation (BSE) and beyond.

Basic algorithmic kernels:

- Large distributed matrix multiplication (short and fat)
- Large distributed linear algebra (LU decomposition, matrix invesion, eigenproblems, etc. . . )
- Non-distributed fast Fourier transformations (FFT)
- Dimensionality reduction and low-rank approximations




BerkeleyGW workflow for the $GW$+BSE procedure.

Four major modules, `epsilon` and `sigma` perform the $GW$ part of the workflow

# Porting BerkeleyGW on Hybrid Architecture

- Design: Fast development of porting strategy
  - Employ a collection of miniapps/kernels simulating the full app running at scale
  - Assess performance/reliability/interoperability of programming models
- Implementation: Achieving best performance → Keep device busy + Hide latency
  - Exploit asynchronous operations for memcopy and kernels execution
  - Keep data on device → avoid useless memcopy
  - Use streams (queues) → high concurrency on device
  - Enable independent execution on host and device → overlap MPI communication
  - Exploit available optimized libraries
- Assess Improvements: Benchmark collection for performance assessment
  - Systematically assess performance (strong scaling, weak scaling, FLOP rate, etc..)
  - Well defined metrics: FLOPs, memory usage, I/O requirements , etc...

# The `espilon` and `sigma` Modules

# The $GW$ Method in BerkeleyGW: The espilon and sigma Modules

Solve Dyson's equation:

$$\left[-\frac{1}{2}\nabla^2 + V_{\mathrm{Nuc}} + V_{\mathrm{H}} + \Sigma(E_n)\right]\phi_n = E_n\phi_n, \tag{1}$$

$\Sigma(E_n) \rightarrow$ self-energy (non-Hermitian, non-local, energy-dependent operator)

**Two Major Computational Bottlenecks:**

1. `espilon`: Polarizability / Dielectric Function $\epsilon \rightarrow O(N^4)$
2. `sigma`: Self-Energy $\Sigma$ (screened-Coulomb interaction) from $\epsilon \rightarrow O(N^4)$

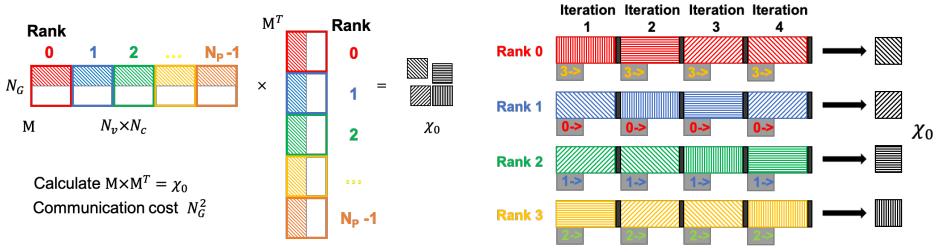
# espilon: Computational Kernels

Three Major Computational Kernels:

- Matrix Elements (MTXEL) $\rightarrow O(N^3 \log N)$
  - Fast Fourier Transformations (FFTs)
  - Node local cuFFT library
- Static Polarizability (CHI-0) $\rightarrow O(N^4)$
  - Large distributed matrix multiplications (short and fat)
  - Most computationally intensive kernel (100s k rows and 10s M cols)
  - Node local cuBLAS library (for the ZGEMMs)
- Inverse Dielectric Function (inversion): Matrix Inversion $\rightarrow O(N^3)$
  - Performed on host using ScalaPACK (for now)
  - Small fraction of total execution time

espilon exploits accelerated libraries (cuFFT, cuBLAS): achieve best performance $\rightarrow$ control memory usage, optimize data transfer and inter-node communication patterns

# `espilon`: CHI-0 Kernel




Data layout (left) and non-blocking cyclic communication pattern (right) for `CHI-0` kernel.

- **Offloading Data Preparation:** Data preparation in $\chi_0$ layout is performed on GPU after offloading $M$ (potential memory bottleneck)
- **Batching Mechanism:** Offload columns of $M$ columns depending on device's maximum memory (avoid hitting OOM), repeat communication pattern for each batch
- **Non-Blocking Cyclic Communication:** Perform communication over a ring topology allowing to overlap ZGEMM on GPU and MPI communication on host

# sigma

sigma computes a set $(10 - 1000)$ self-energy matrix elements $\{\Sigma_{lm}\}$ to solve the Dyson's equation. Working equation with Generalized Plasmon Pole model (GPP):

$$\Sigma_{lm}(E) = \frac{1}{2} \sum_{n} \sum_{GG'} M_{nl}^{-G} P_{GG'}[E - \epsilon_n] v(G') M_{nm}^{-G'}$$

$\mathbf{P}[E - \epsilon_n] \rightarrow$ computed on the fly using $\epsilon^{-1}$ matrix (from epsilon) and band energy $\epsilon_n$

The evaluation of each $\Sigma_{lm}$ represents a data reduction across different matrices with a complex matrix-vector interdependence $\rightarrow$ the GPP kennel

sigma implements a two level paralleization:

- Inter-pool parallelization: Pools working independently on a subset of $\{\Sigma_{lm}\}$
- Intra-pool parallelization: Evaluation of each $\Sigma_{lm}$ within a pool

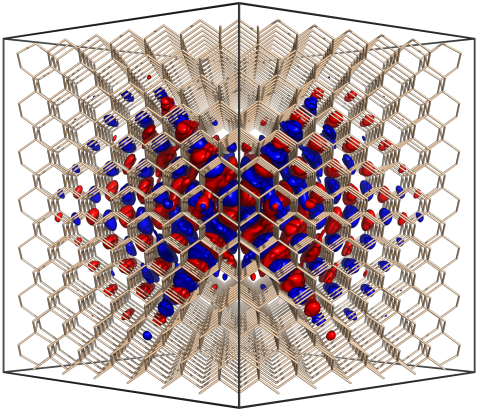
# sigma: the GPP kennel

For large scale applications, almost the entire computational workload in sigma is performed by the GPP kennel → developed our own kernel

loop $G' < N_G^{\text{distr}}$
. loop $G < N_G$
. . loop $n < N_{b-\text{block}}^{\text{distr.}}$
. . . Contract $P_{GG'}$ with $M_{Gn}^{l}$ and $M_{G'n}^{m}$
. . . Accumulate $\sigma_{b-\text{block}}$ (shared memory)
Reduce $\sigma$ over GPU thread blocks, CUDA streams, and MPI ranks

- The first two loops are collapsed, the $N_b^{\text{dist}}$ loop is unrolled on each thread
- The $n$ loop is divided into band blocks ($N_{b-\text{block}}^{\text{distr}}$ size to fit L1/L2 caches)
- Each band block is placed on different CUDA stream (concurrent execution)
- Partial reduction on GPU and finalized on host, overlap data-transfer/computation
- Non-blocking cyclic communication: overlap computation/communication

# Performance Results



# Benchmarks for Performance Measurement

| Parameters | Si-214 | Si-510 | Si-998 | SiC-998 | Si-2742 |
|---|---|---|---|---|---|
| $N_{\text{spin}}$ | 1 | 1 | 1 | 2 (↑/↓) | 1 |
| $N_G^{\psi}$ | 31,463 | 74,653 | 145,837 | 422,789 | 363,477 |
| $N_G$ | 11,075 | 26,529 | 51,627 | 149,397 | 141,505 |
| $N_b$ | 6,397 | 15,045 | 29,346 | 16,153 | 80,694 |
| $N_v$ | 428 | 1,020 | 1,996 | 1,997/1,995 | 5,484 |
| $N_c$ | 5,969 | 14,025 | 27,350 | 14,156/14,158 | 75,210 |
| $N_\Sigma$ | Variable, up to 128 per spin | | | | |
| Epsilon PFLOPs | 2.5 | 80.5 | 1164 | 10,091 | 66,070 |
| Epsilon Memory (TB) | 0.45 | 6.07 | 45.1 | 135 | 934 |
| Sigma PFLOPs | 0.127 | 1.71 | 12.6 | 58.2 | 260.7 |
| Sigma Memory (GB) | 6.19 | 34.3 | 133.8 | 791.4 | 1006 |

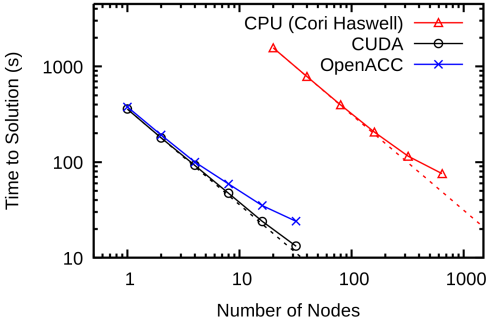
↑ and ↓ represent spin up and spin down (when $N_{\text{spin}} = 2$). For Sigma PFLOPs are given per self-energy matrix element, and that the Sigma Memory is per pool, accounting for device memory only.

**Both epsilon and sigma for the largest applications require for their completion a minimum amount of FLOPs on the order of tens of ExaFLOPs.**

# GPU *vs* CPU Speedup

| | MTXEL | CHI-0 | Invert | Total |
|---|---|---|---|---|
| CPU Only | 616 | 1120 | 10.3 | 1794 |
| GPU Host-Prep | 24.2 | 100.4 | 9.3 | 146.3 |
| GPU Full-Offload | 24.4 | 47.7 | 9.6 | 96.3 |

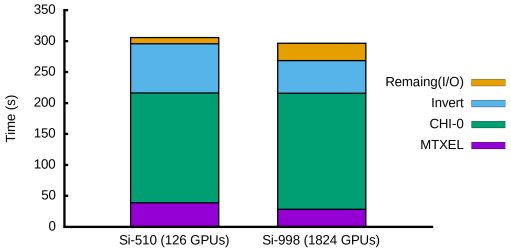
Runtime comparison (in seconds) of CPU and GPU implementations for **epsilon** on Cori-GPU (2 nodes) at NERSC for Si-214. `Epsilon` exhibits an overall 18.6× speedup.



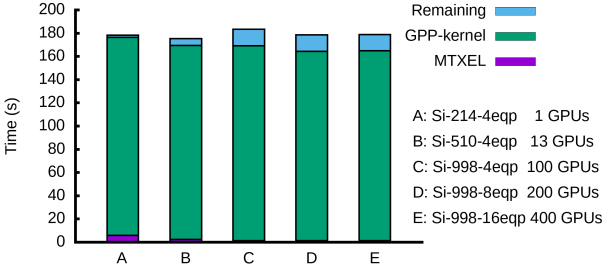

Runtime comparison of **sigma** on CPU on Cori Haswell and GPU on Summit (CUDA/OpenACC) for the Si-510 system (4 $\Sigma$ elements). A 1-to-1 node comparison give a 86× speedup.

CUDA and OpenACC implementation are within 10% in runtime.

# Weak Scaling

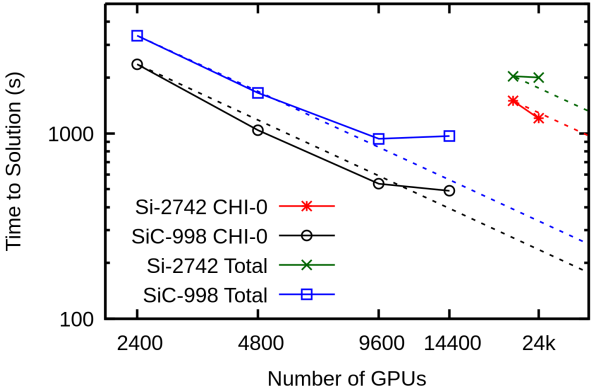



Weak scaling performance of **epsilon** on Summit. The number of GPUs is scaled according to the computational complexity $O(N^4)$ of `CHI-0`.




Weak scaling performance of **sigma** on Summit. The number of GPUs is scaled according to the $O(N^3)$ computational complexity in Cases A, B and C, and to the number of $\Sigma$ matrix elements in Cases C, D and E.

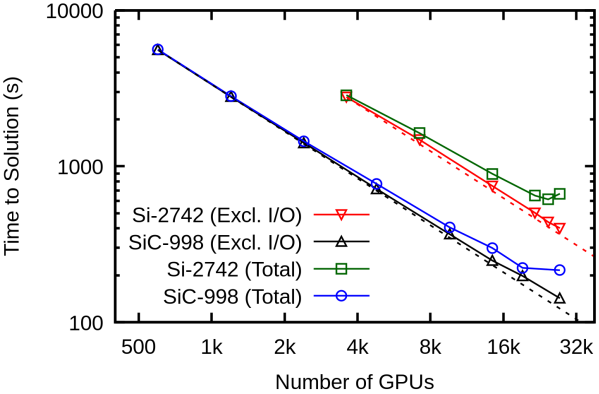
# `epsilon`: Strong Scaling and Flop Rate




SiC-998: 15 mins total on 1600 Summit nodes (9600 GPUs). `CHI-0`: 18.9 PFLOP/s (27% of peak)

Si-2742: 33 mins total on 4000 Summit nodes (24000 GPUs). `CHI-0`: 54.8 PFLOP/s (31.4% of peak)

# sigma: Strong Scaling and Flop Rate



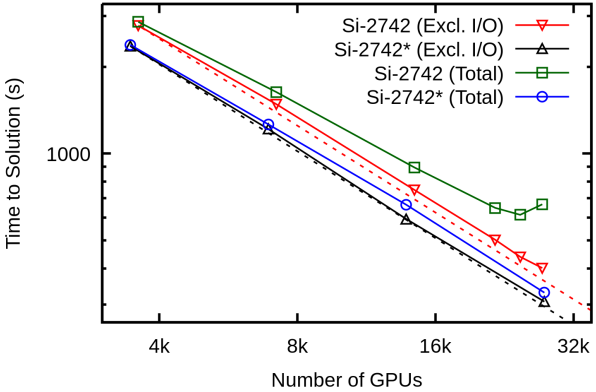

Scaling up to 4560 Summit nodes (27,360 GPU's) 99% of entire Summit. GPP kernel performance for Si-2742 system: 78.0 PFLOP/s (39.2% of peak)

# Optimizing sigma for Large Scale Applications

- Improve I/O exploiting solid-state device memory (SSD)
  - Data are prepared into a distributed form before the run
  - Preparation need to be performed only once for each pool size at no cost
  - Data are pre-staged to node-local SSD memory before execution
  - At execution each MPI task read data in distributed form directly from local SSD
- Improve GPP kernel performance
  - To reduce data movement, the outer collapsed $GG'$ loop is tiled with small blocks, allowing $G$ and $G'$ local arrays to be kept at least on L2 cache
  - Some small, frequently used arrays are kept into GPU's shared memory
  - Parameters such as number of thread blocks / threads per block / number of CUDA streams / band-block size are optimized to problem size to achieve best performance
  - Pool size is tailored to Summit's cabinet size to optimize communication performance

Results obtained with the optimized sigma are labelled with *

# Optimized (*) sigma: Strong Scaling and Flop Rate




Scaling up to 4608 Summit nodes (27,648 GPU's) 100% of entire Summit. Overall time to solution 330s, I/O time 23s. GPP kernel performance for Si-2742 system: 102.1 PFLOP/s (50.9% of peak).

# Summary

Porting BerkeleyGW on GPUs:

- More than $10\times$ acceleration compare to CPU architectures
- Good strong / weak scaling with high fraction of peak performance
- Excellent time to solution for systems made of thousands of atoms

We also show that OpenACC can achieve similar performance as CUDA providing a proof of concept that a directive-based programming model (syntactically similar to OpenMP) can be nearly as performant as a vendor specific programming model (CUDA). This is a promising avenue for portability strategies for large scale legacy codes especially looking forward to all major DOE pre-exascale and exascale super computers such as Perlmutter, Frontier, El Capitan and Aurora.

# Thank You!

**Collaborators**: Charlene Yang (NERSC), Zhenglu Li (UC Berkeley & LBNL), Felipe H. da Jornada (Stanford University), Steven G. Louie (UC Berkeley & LBNL) and Jack Deslippe (NERSC).

**Acknowledgments**: This work was supported by the Center for Computational Study of Excited-State Phenomena in Energy Materials (C2SEPEM) at Lawrence Berkeley National Laboratory (LBNL), 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. Computational resources were provided by Oak Ridge Leadership Computing Facility through the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program, which is a DOE Office of Science User Facility supported under Contract No. DE-AC05-00OR22725. Computational resources were also provided by the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.