S21353: Accelerating Large-Scale GW Calculations in Material Science

Mauro Del Ben[1], Charlene Yang[2]

[1]Computational Research Division, Lawrence Berkeley National Laboratory
[2]National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory

March 25, 2020

GPU Technology Conference (GTC), San Jose

Outlook

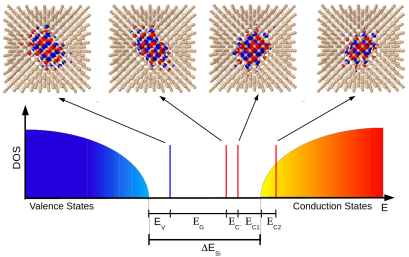
- Motivation and Introduction
- BerkeleyGW Software Package $\rightarrow$ Portability Strategy
- GPU support for `epsilon`
- GPU support for `sigma`
- Large Scale Application
- Summary

Accurate Optical and Electronic Properties of Complex Materials

Complex Materials: unique electronic and optical triggered by symmetry breaking

Important implications in many fields:

- Quantum Computing
- Energy Storage/Conversion
- Photovoltaics
- Nanoelectronics
- Catalysis



Example: schematic representation of the electronic structure of a divacancy in crystalline Silicon

Accurate predictions requires:

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

Introduction: The GW Method

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

Solve Dyson's equation:

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

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

Bottlenecks:

1. Evaluation of the Polarizability $\rightarrow O(N^4)$ static and/or frequency dependent
2. Evaluation of the Self-Energy $(\Sigma) \rightarrow O(N^3) - O(N^4)$

State of the Art

Application of GW to thousands atoms systems still a challenge

Reduce time to solution and extend applicability:

- Improve single node performance and parallel scalability
- Develop methods to reduce prefactor and scaling with system size

HPC on the Path to Exascale: Hybrid Architectures

Code optimization for HPC applications will be focused on hybrid GPU-CPU systems, in this talk some of the portability strategies to implement GPU support for the BerkeleyGW software package will be discussed.

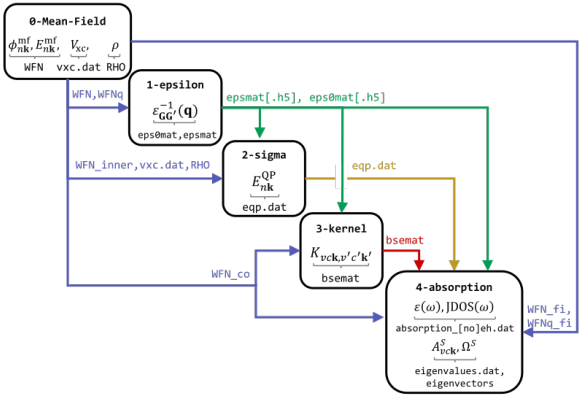
The BerkeleyGW Software Package

BerkeleyGW

https://berkeleygw.org/

- Compute electron excited-state properties of materials via GW, Bethe-Salpeter equation (BSE) and beyond
- Parallelization: Hybrid MPI / OpenMP (CPU) / GPU (Cuda, OpenACC)
- On many-core architectures: scaling up to 100,000's cores achieving high fraction of peak performance
- Basic algorithmic kernels:
  - Large distributed matrix multiplication (tall and skinny matrices)
  - Large distributed linear algebra (LU decomposition, invesion, eigenproblems, etc. . .)
  - Non-distributed fast Fourier transformations (FFT)
  - Dimensionality reduction and low-rank approximations

The BerkeleyGW Workflow



Four major executables:

- `epsilon` → Polarizability and Dielectric function of the material
- `sigma` → GW quasi-particles energy (band structure)
- `kernel` → BSE matrix elements
- `absorption` → Interpolate BSE kernel matrix elements and solve BSE

`epsilon` and `sigma` perform the GW part of the workflow → Full GPU support for both executables

Portability Strategy on Hybrid Architecture: Combining the Strengths

CPU - Speed

Task Parallelism

GPU - Throughput

Data Parallelism

Images from Wikipedia

Portability Strategy on Hybrid Architecture

Achieving best performance $\rightarrow$ Keep device busy + Hide latency

- Use miniapps siulating full app running at scale to develop best porting strategy
- Take advantage of asynchronous operations for memcopy and kernels execution
- Keep data on device $\rightarrow$ implement intermediate kernels $\rightarrow$ avoid useless memcopy
- Use streams (queues) to enable high concurrency on device
- Enable for independent execution on host and device (overlap communication)
- If possible use available optimized libraries

Portability Strategy: Benchmark for Performance Measurement

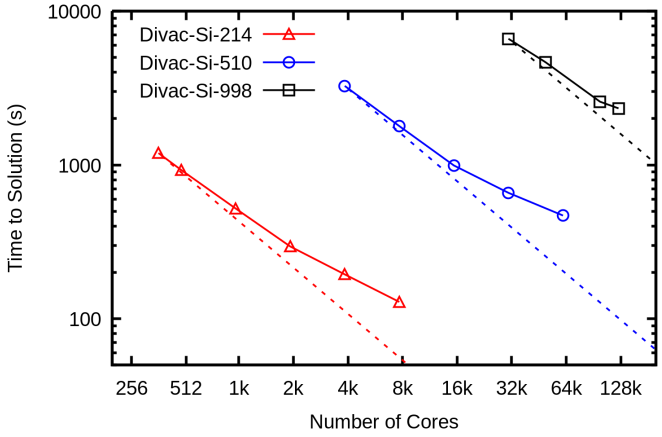
Assessing performance across architecture and track improvements:

- Systematically assess performance (strong scaling, weak scaling, etc..)
- Well defined metrics: Flops, memory usage, I/O requirements , etc...

	Divac-Si-214	Divac-Si-510	Divac-Si-998
N_G^ψ	31,463	74,653	145,837
N_G^χ	11,075	26,529	51,627
N_n	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

Divacancy defect in Silicon, three supercell, with 214, 510 and 998 atoms respectively

Baseline Performance



Strong Scaling for the `epsilon` code measured on Edison@NERSC (Cray XC30, Ivy Bridge processors)

GPU support for epsilon

Introduction: The GW Method

Solve Dyson's equation:

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

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

In BerkeleyGW:

1. `espilon`: Evaluation of Polarizability / Dielectric Function $\epsilon \rightarrow O(N^4)$
2. `sigma`: Evaluation of Self-Energy $(\Sigma) \rightarrow O(N^3) - O(N^4)$

Dielectric Function ϵ and its inverse needed to compute the self-energy Σ

Epsilon Code: Inverse Dielectric Matrix ϵ^{-1}

Input: $\psi_{m\mathbf{k}}$, $\epsilon_{m\mathbf{k}}$, $\{\mathbf{q}\text{-points}\}$, $\{\omega_i\}$

1. Calculate plane-waves matrix elements (FFT's):

$$M^G_{ja\mathbf{k}}(\mathbf{q}) = \langle \psi_{j\mathbf{k}+\mathbf{q}} | \, e^{i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}} \, | \psi_{a\mathbf{k}} \rangle$$

2. Calculate Static RPA polarizability (Matrix-Multiplication):

$$\chi(\mathbf{q},\omega_i) = \mathbf{M}(\mathbf{q})^\dagger \mathbf{\Delta}_{ja\mathbf{k}}(\epsilon_{j\mathbf{k}}, \epsilon_{a\mathbf{k}}, \mathbf{q}, \omega = 0)\mathbf{M}(\mathbf{q})$$

   $\mathbf{\Delta}$ diagonal matrix

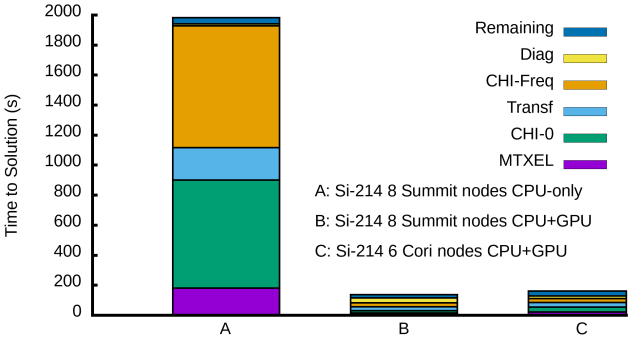
3. Low rank approximation for the frequency dependent part $(\omega \neq 0)$
4. Dielectric matrix ϵ and inversion: $\epsilon^{-1}(\mathbf{q},\omega_i) = (I - v\chi(\mathbf{q},\omega_i))^{-1}$

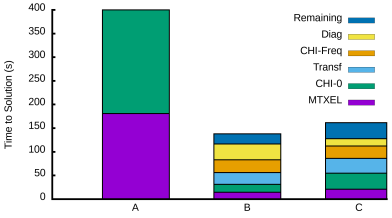
Five major computational kernels: (1) Matrix Elements, (2) Static Polarizability, (3) Diagonalization, (4) Basis Transformation and (5) Frequency Dependence

Epsilon: Hybrid GPU-CPU Implementation

1. Matrix Elements (`MTXEL`): Unfavorable $O(N^3)$ vs $O(N^3 \log N)$ data/flops
   - cuFFT library $\rightarrow$ no benefit by just linking
   - Asynchronous data transfer $\rightarrow$ pinned host memory/data streams
2. Static Polarizability (`CHI-0`): Favorable $O(N^3)$ vs $O(N^4)$ data/flops
   - Use cuBLAS library $\rightarrow$ Asynchronous host to device data transfer
   - Non-blocking cyclic MPI communication scheme
   - Overlap CPU-communication/GPU-computation
3. Diagonalization (`Diag`): $O(N^3) \rightarrow$ ELPA
4. Basis Transformation (`Transf`): $O(N^4)$ memory bottlenecks for both host/device
   - Batch communication over eigenvectors $\rightarrow$ control host memory usage
   - Batch computation over wavefunctions $\rightarrow$ control device memory usage
5. Frequency Dependence (`CHI-Freq`): $O(N^4)$ multiple matrix multiplications
   - Smaller matrices ($N_G/N_b \simeq 5-10$) at multiple frequencies
   - Data streams over frequency index $\rightarrow$ allows for concurrent execution on device

Epsilon: CPU-Only *vs* Hybrid GPU-CPU (Summit@OLCF)



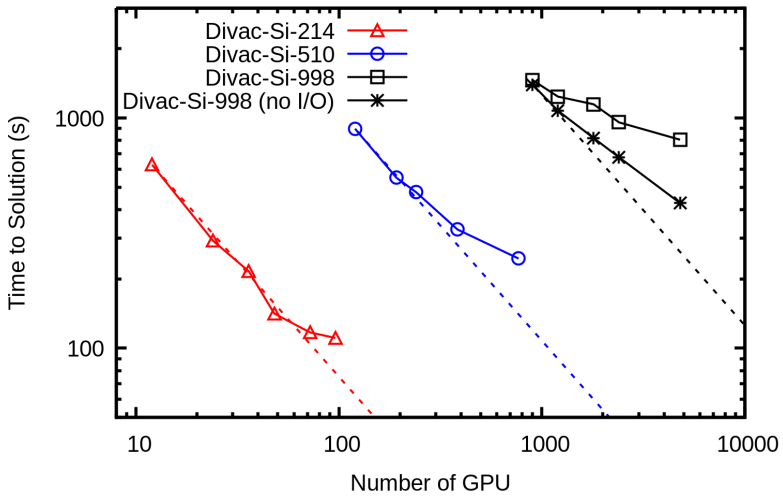


Same as left plot, zoomed in.

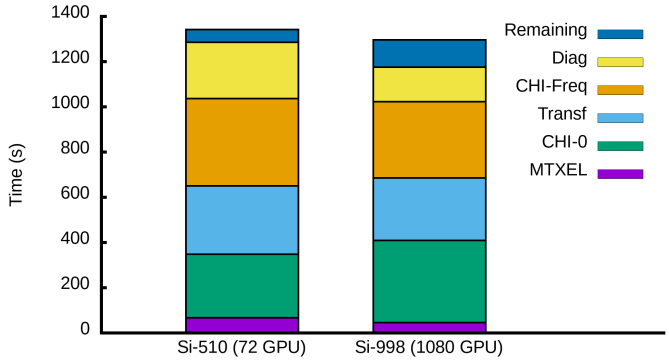
Summit node: 2 IBM POWER9 CPUs (21 cores each) and 6 NVIDIA V100 (Volta) GPUs, aggregate performance 42 TFlops. Cori-GPU node: 2 sockets of 20-core Intel Skylake + 8 NVIDIA Volta GPUs.

Overall 15x speed-up for the hybrid implementation vs CPU-only

Summit@OLCF: Strong Scaling

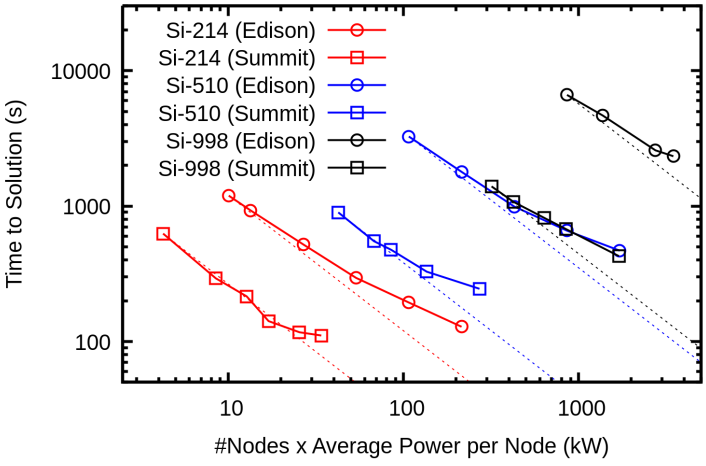


Summit@OLCF: Weak Scaling



Favorable $O(N^3)$ *vs* $O(N^4)$ scaling of memory *vs* Flops, larger batch sizes by increasing computational resources proportionally to Flops

Comparison Across Architectures: Time to Solution *vs* Power



Average power per node (from Top500 website), Edison: 0.67 kW, Summit 2.12 kW.

GPU support for sigma

Introduction: The GW Method

Solve Dyson's equation:

$$\left[-\frac{1}{2}\nabla^2 + V_{\text{Nuc}} + V_{\text{H}} + \Sigma(E_n)\right]\phi_n = E_n\phi_n, \tag{3}$$

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

In BerkeleyGW:

1. `espilon`: Evaluation of Polarizability Dielectric Function $\epsilon \rightarrow O(N^4)$
2. `sigma`: Evaluation of Self-Energy $(\Sigma) \rightarrow O(N^3) - O(N^4)$

The Self-Energy Matrix Elements $\Sigma_{lm}(E)$

Self-Energy matrix element for a given pair of orbital functions $\{\phi_l, \phi_m\}$

$$\Sigma_{lm}(E) = \frac{i}{2\pi} \int_0^\infty d\omega \sum_n \sum_{GG'} M_{nl}^{-G} \frac{\epsilon_{GG'}^{-1}(\omega) \cdot v(G')}{E - E_n - \omega} M_{nm}^{-G'}$$

Frequency Treatment:

- Full-Frequency (FF):
  - Analytical integration over frequency
  - Require frequency dependent dielectric matrix
- Generalized Plasmon Pole (GPP) Model:
  - Analytical approximation to the frequency dependence
  - Require only the static dielectric matrix

The Generalized Plasmon Pole Model (GPP)

For the Coulomb-Hole (CH) term (similar expression for the SX):

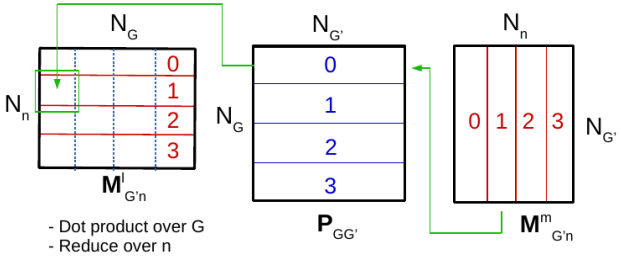
$$\Sigma^{\mathsf{CH}}_{lm}(E) = \frac{1}{2}\sum_{n}\sum_{GG'} M^{-G}_{nl}\frac{\Omega^2_{GG'}(1 - i\tan\phi_{GG'})}{\tilde{\omega}_{GG'}(E - \epsilon_n - \tilde{\omega}_{GG'})}v(G')M^{-G'}_{nm}$$

$$= \frac{1}{2}\sum_{n}\sum_{GG'} M^{-G}_{nl}P^{\mathsf{CH}}_{GG'}[E - \epsilon_n]v(G')M^{-G'}_{nm}$$

Ω, $\tilde{\omega}$ and $\phi \rightarrow$ effective bare plasma frequency, GPP mode frequency and phase of renormalized Ω^2

- Coupling between $[E - \epsilon_n]$ and $\mathbf{GG'} \rightarrow$ can not be reformulated as a matrix multiplication (contrary to the FF case)
- Basic algorithm motif:
  - For each $n \rightarrow$ matrix-vector multiplication
  - Dot product
  - Reduction over n

Sigma GPP Code: Two Level Parallelization



Schematic example of the data layout and operations for task 0 at the second cycle of the process's loop of the parallel algorithm.

Pool of processes each working on a subset of the total number of $\{\Sigma_{lm}(E)\}$, for each Σ matrix element:

- Compute matrix elements $\mathbf{M}^m/\mathbf{M}^l$ distributed over columns $n \rightarrow$ `MTXEL` kernel
- Prepare intermediates to compute $\mathbf{P}^{\mathrm{CH/SX}}$ distributed over rows $G \rightarrow$ `PREP`
- Communication only within the pool $\rightarrow$ Broadcast or Non-Blocking Cyclic
- Most of the computation is performed in the `Sigma-GPP` kernel

The Sigma-GPP Kernel

Outer loop over processes in the pool, at each iteration perform the contraction with the received $\mathbf{M}^m/\mathbf{M}^l$ $(N_G^{\text{tot}}, N_n^{\text{distr}})$ and local $\mathbf{P}^{\text{CH/SX}}$ $(N_G^{\text{tot}}, N_G^{\text{distr}})$:

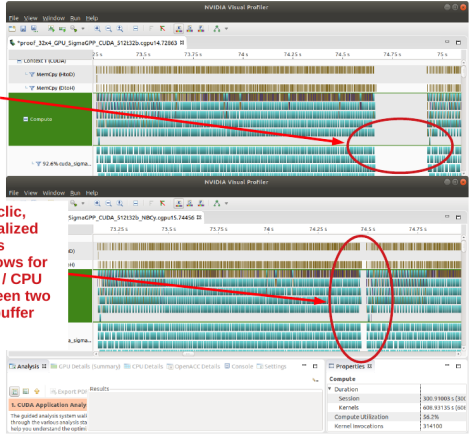
Sigma-GPP Kernel: Stride loop over collapsed two outermost loops

```
loop my_igp < NG_distr
. loop ig < NG_tot
. . loop n1_loc < N_block_size
. . . Contract P with M
. . . Accumulate SCH and SSX
Reduce SCH and SSX over threads
for each band/thread block
```

On Host:

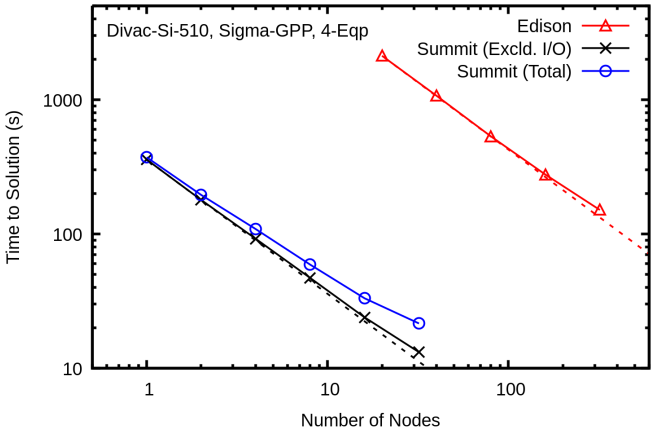
1. Loop over block of bands, one stream for each band block, for each stream launch Sigma-GPP kernel with N_{TB} (64) thread blocks with N_T (256) threads per block
2. Synchronize communication (overlap kernel execution MPI comm.)
3. Loop over block of bands, synchronize band stream, and finalize reduction

Sigma GPP Code: Profiling



Non-Blocking Cyclic communication scheme allows for overlap computation (GPU) and communication (CPU). Also shown is the concurrent executions of `Sigma-GPP` kernels on device.

Strong Scaling: Summit@OLCF *vs* Edison@NERSC



A 1:1 node comparison give over 100x speed-up on Summit *vs* Edison

Sigma GPP Code: Flop Count

Obtaining the Flop count:

- For large runs $> 99\%$ of the flops are performed by the `Sigma-GPP` + `zgemm` kernels $\rightarrow$ neglecting the flops on host and other device kernels (FFTW, etc...)
- From the scaling: Flops $= a \times N_{\text{eqp}} \times N_n \times N_G^2$, obtain the prefactor a by fitting the flop count for a series of calculations wrf $(N_{\text{eqp}}, N_n, N_G^2)$:

$$\text{Flops} = (a_{\text{gpp}} + a_{\text{zgemm}}) \times N_{\text{eqp}} \times N_n \times N_G^2$$

$$a_{\text{gpp}} = 153.44 \quad ; \quad a_{\text{zgemm}} = 8$$

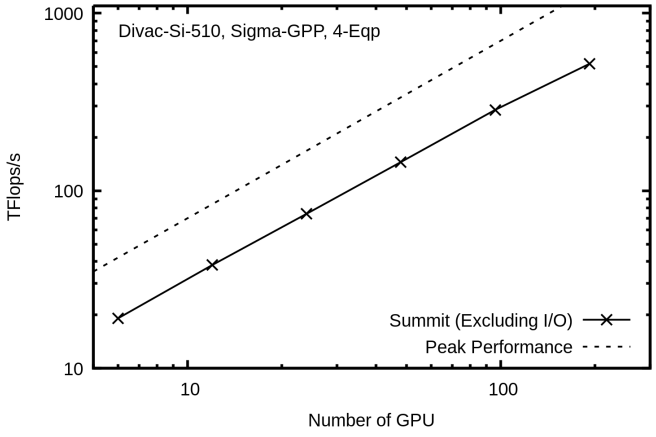
- Check formula by comparing with a set of independent calculations

Sigma GPP Code: Validating Flop Count Formula

System	N_{eqp}	N_n	N_G	TFlop Measured	TFlop Estimated	% Est./Meas.
Divac-Si-510	2	1322	3287	4.641	4.612	99.4
Divac-Si-510	2	2631	3287	9.212	9.178	99.6
Divac-Si-510	2	3223	9315	90.49	90.30	99.8
Divac-Si-510	2	4261	9315	119.54	119.37	99.8
Divac-SiC-214	4	2123	2103	6.070	6.063	99.8
Divac-SiC-214	4	1113	2945	6.259	6.234	99.6
Divac-SiC-214	4	2309	2945	12.96	12.93	99.8
Divac-SiC-214	4	3409	6979	107.23	107.22	99.9

Table: Fitting performed for the Divac-Si-214 system, the Flop count include both the `Sigma-GPP` and `zgemm`.

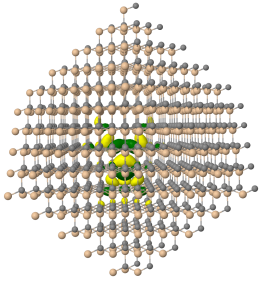
Flop Rate on Summit@OLCF

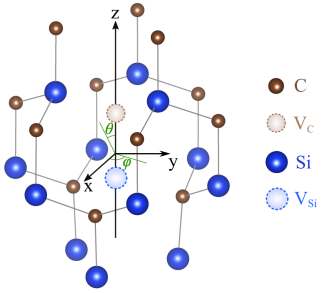


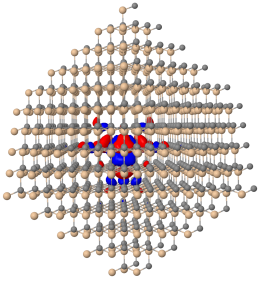
Summit node: 42 TFlops/s peak performance (6 GPU's). Sigma-GPP: 1-Summit node 19.1 TFlops/s (45.5% peak) ; 32-Summit nodes 519 TFlops (38.6% peak).

Large Scale Application

Divacancy Defect in SiC: 998 Atoms Supercell





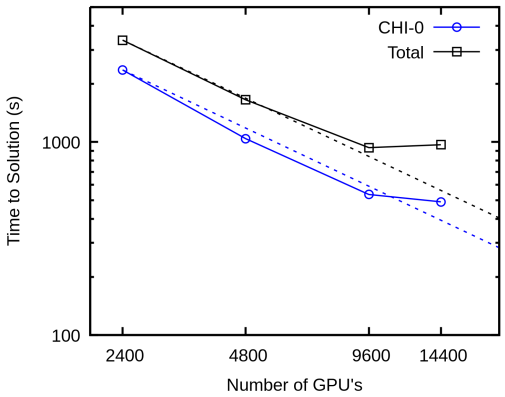


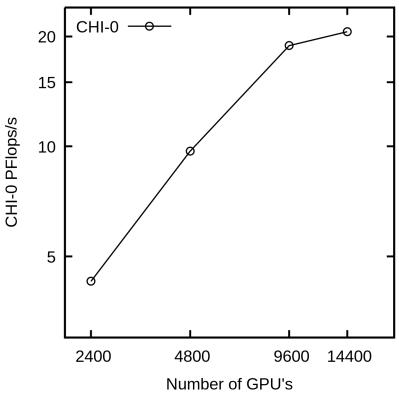
Prototype for solid state QBit

Divacancy Defect in SiC: Calculation Size

	Divac-SiC-998	
N_{spin}	2	
N_G^{ψ}	422,789	
N_G^{χ}	149,397	
N_n	16,153	
N_v	1,997 ($\uparrow$) / 1,995 ($\downarrow$)	
$N_c * N_v$	28.2 M	
N_{eqp}	80 $\times N_{\text{spin}}$	
	`epsilon`	`sigma`
I/O Read (Gb)	230	537
I/O write (Gb)	333	0
Min. Memory (Tb)	135	0.74 $\times$ pool
Min. Flops (EFlops)	10.1	9.31

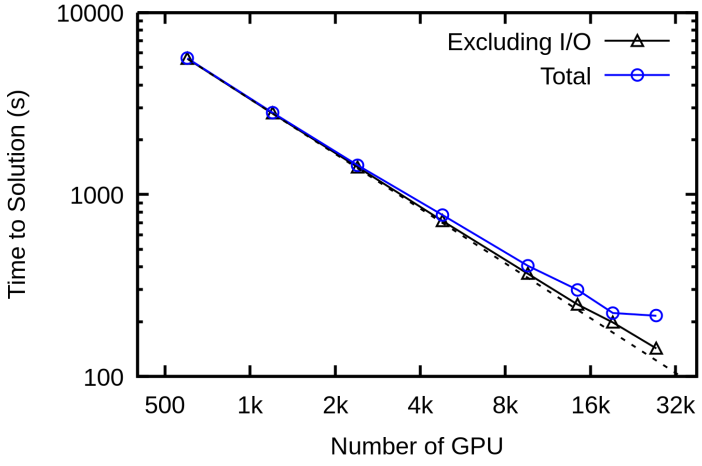
Divacancy Defect in SiC: epsilon CHI-0





On 1600 Summit Nodes (9600 GPU's): time to solution 15 mins total

Divacancy Defect in SiC: `sigma` GPP (80 E_{qp} per spin)



Scaling up to 4,560 Summit nodes (27,360 GPU's) 99% of entire system.

Divacancy Defect in SiC: sigma GPP

	100 Nodes	4560 Nodes
Number of GPU's	600	27,360
Number of Sigma-Pools	5	80
GPU's per Pool	120	342
I/O time (s)	33	71
Compute time (s)	5575	142

Summary

GPU support in BerkeleyGW:

- More than $10\times$ acceleration compare to CPU architectures
- Good strong / weak scaling with high fraction of peak performance
- Order of magnitude improvement in energy per flop efficiency
- Excellent time to solution for systems made of thousands of atoms

Acknowledgment: Work supported by the Center for Computational Study of Excited-State Phenomena in Energy Materials (C2SEPEM) at LBL, funded by the U.S. DOE under Contract No. DE-AC02-05CH11231. Computational resources provided by NERSC within NESAP program and by OLCF within INCITE program.