Accelerating Large-Scale Excited-State GW Calculations on Leadership HPC Systems

Mauro Del Ben*, Charlene Yang*, Zhenglu Li*‡, Felipe H. da Jornada†, Steven G. Louie*‡ and Jack Deslippe*
mdelben@lbl.gov cjyang@lbl.gov lzlwell@berkeley.edu jornada@stanford.edu sglouie@berkeley.edu jrdeslippe@lbl.gov

*Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA
†Department of Materials Science and Engineering, Stanford University, Stanford, 94305, California, USA
‡Department of Physics, University of California at Berkeley, California 94720, USA

***Abstract*—Large-scale GW calculations are the state-of-the-art approach to accurately describe many-body excited-state phenomena in complex materials. This is critical for novel device design but due to their extremely high computational cost, these calculations often run at a limited scale. In this paper, we present algorithm and implementation advancements made in the materials science code BerkeleyGW to scale calculations to the order of over 10,000 electrons utilizing the entire Summit at OLCF. Excellent strong and weak scaling is observed, and a 105.9 PFLOP/s double-precision performance is achieved on 27,648 V100 GPUs, reaching 52.7% of the peak. This work for the first time demonstrates the possibility to perform GW calculations at such scale within minutes on current HPC systems, and leads the way for future efficient HPC software development in materials, physical, chemical, and engineering sciences.**

***Index Terms*—electronic structure, excited states, GW method, GPU acceleration, divacancy defects, quantum computing**

I. Justification for ACM Gordon Bell Prize

This work presents unprecedented GW calculations with up to 10,968 electrons on materials with divacancy defects, proxies for qubits, performed on full-scale Summit achieving 105.9 double-precision-PFLOP/s, 52.7% peak. With near linear scaling, the GPU BerkeleyGW implementation exhibits up to 86× speedup and 16× energy saving compared to its CPU implementation.

II. Performance Attributes

Category of Achievement	Scalability, Time-to-solution, Peak Performance
Type of Method Used	Both Explicit and Implicit
Results Reported	Full Application Excluding I/O
Precision Reported	Double Precision
System Scale	Measured on Full System
Measurement Mechanism	Timers, FLOP Count

III. Overview of the Problem

Electrons in excited states dictate many fundamental properties of materials, such as the efficiency of solar energy conversion, protein folding, photosynthesis reactions, and chemical reactions, to name a few. Understanding the excited-state properties of these electrons is hence critical to the advancement of many scientific domains including material sciences, condensed matter physics, chemistry, biophysics, and electronic engineering. Exactly solving for the properties of electrons using quantum mechanics requires a computational cost that increases exponentially with the problem size. This is intractable in practice, and approximations *must* be made.

First-principles computations aim at obtaining important materials' properties using fundamental theories without fitting parameters from experiments, and density-functional theory (DFT) [1] is one of the most common approaches. However, DFT is a ground-state theory and it presents serious errors for excited-state properties. For example, the band gap (an excited-state property) of silicon is underestimated by over 50% with the standard approximations in DFT [2].

The first-principles GW approach [2], [3] is the prevailing excited-state method for many materials ranging from photovoltaic cells to novel low-dimensional nanostructures. It directly confronts the many-body nature of the electron-electron interactions and is hence valid in describing excited electrons, beyond DFT; however, the resulting equations are much more complicated. One very important excited-state property is the quasiparticle excitation energy (i.e. energy level and band structure). For example, obtaining accurate excitation energies of defect states in semiconductors are critical for manipulating them with light for quantum computing [4]. Here we use semiconductor with divacancy defects as the application case of the GW method. These systems usually require a large supercell consisting of up to thousands of atoms due to the extended nature of the defect states (see Fig. 1), posing great challenges in obtaining GW results.

BerkeleyGW [5] is a massively parallel package for studying exited-state properties of electrons in materials using the GW method and beyond. Previously, it has shown excellent scalability on CPUs [6], but with increasing demands for larger scale GW calculations, acceleration is exceedingly needed. In this paper, we present several significant algorithm and implementation advancements made in BerkeleyGW to scale these calculations efficiently on leadership supercomputers, and to an unprecedented level of 2,742 atoms and 10,968 electrons. The parallelization and optimization techniques presented here are widely applicable to other GW or HPC codes which share the same computational characteristics, hence being largely beneficial to the community.

To summarize, we have made contributions as follows:

- We developed a multi-tier parallelization scheme across

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

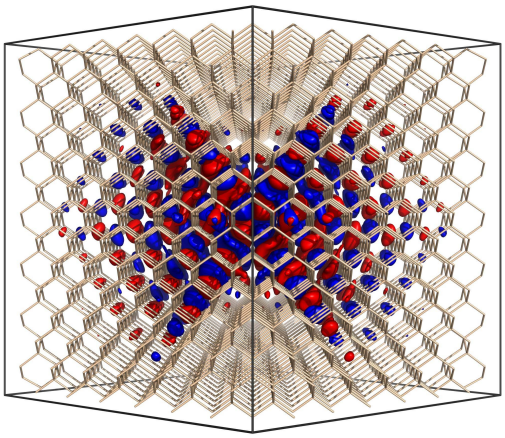

Fig. 1. Isosurface enclosing 90% of the wavefunction for an in-gap state from a divacancy defect in Si. Calculation cell contains 2,742 atoms, 10,968 electrons.

valence/conduction bands, quasiparticle states, and planewave basis elements to allow for an efficient and flexible multi-GPU, multi-node implementation. Techniques employed include non-blocking cyclic communication, workload batching, asynchronous data copies, and the utilization of shared memory on GPUs (Sec. V).

- We achieved near linear strong and weak scaling up to the full scale of Summit with 27,648 GPUs. This makes large-scale GW calculations possible and, unprecedentedly, a Si divacancy structure with 2,742 atoms and 10,968 electrons is solved within 10 minutes.
- We achieved 105.9 PFLOP/s double-precision performance with the 2,742-atom structure, running on the full size of Summit, reaching 52.7% of the peak (200.79 PFLOP/s). We also observed an 86 times runtime speedup and 16 times energy saving when comparing Summit at OLCF and Cori Haswell at NERSC node-to-node.
- We implemented a workflow for pre-staging large input files to node-local SSDs on Summit to improve I/O.

IV. Current State of the Art

The first-principles GW method has been fast developing since its inception as a practical approach by Hybertsen and Louie [2] in the 1980s, with G representing the Green's function and W the screened Coulomb interaction. Standard GW formalism involves summation over many unoccupied states and scales as $O(N^4)$. (As a comparison, DFT scales as $O(N)$ to $O(N^3)$ depending on different strategies.) New GW approaches have been proposed to eliminate the summation over unoccupied states, as demonstrated by Umari et al. [7], by Giustino et al. [8], and by Govoni et. al [9]–[11]. The formal scaling of these implementations remains $O(N^4)$, and they maintain similar characteristics to the standard GW approach which can be exploited for optimization and parallelization. Planewave basis sets are commonly adopted for accurate treatment in the implementations of the GW method, such as the popular software packages including BerkeleyGW [5] (this work), Yambo [12], WEST [9], Quantum ESPRESSO [13], Abinit [14], VASP [15], [16], and SterheimerGW [17].

Recently, cubic-scaling GW approaches have been developed [18]–[20], which typically rely on an alternative representation of the self-energy – for example, in the real-space and imaginary-time domains. The cubic scaling is then obtained at the cost of an increased overhead with a large prefactor, usually related to a series of numerical integration. Therefore, the problem size at which the cubic-scaling approach outperforms the standard $O(N^4)$ approach is strongly system-dependent. Another emerging avenue for reduced-scaling GW is the use of stochastic sampling techniques [21], which enables linear scaling for some properties of materials (not all properties are available with stochastic approaches) by sacrificing the deterministic nature of the calculation and introducing uncorrelated stochastic errors.

Besides the above-mentioned planewave GW packages, there are other implementations using localized basis functions, such as Gaussians (Fiesta [22], MolGW [23]), numerical atomic orbitals (FHI-aims [24], SIESTA [25]), mixed Gaussian and planewaves (CP2K [26]) and linearized augmented-planewave with local orbitals (ELK [27], Exciting [28]). Localized bases require a smaller basis size, hence reducing the computational cost; however these calculations are more difficult to converge systematically compared with planewave basis sets, especially for systems with diffuse wavefunctions as in small-gap semiconductors and metals. In general, localized basis sets are particularly efficient for isolated systems (molecules and cluster), while planewaves are more convenient and natural for solid-state systems.

As most GW studies still focus on systems with tens to hundreds of atoms because of the high computational complexity, the community is pushing towards much larger systems to access new phenomena. To the best of our knowledge, the largest GW calculation to date is for a twisted bilayer phosphorene structure containing ∼2,700 atoms (∼13,500 electrons) using linear-scaling stochastic GW [29]. For the deterministic approaches (which provide a more complete picture of the excited-state properties), one of the largest and recently reported calculation involves a graphene nanoribbon consisting of ∼1,700 atoms using cubic-scaling GW combined with localized Gaussian basis set [19]. For the standard $O(N^4)$ scaling GW using the accurate planewave basis set, the system size is still quite limited. Here, we focus on optimizing the standard $O(N^4)$ GW approach with planewave basis sets as implemented in BerkeleyGW [5] and achieve unprecedented scalability and applicability to describe complex materials.

V. Innovations Realized

A. GW Method Complexity

The many-body problem of electronic excitations in a material is described by the Dyson's equation (in atomic units),

$$h_0(\mathbf{r})\psi_i(\mathbf{r}) + \int \Sigma(\mathbf{r},\mathbf{r}';E_i^{\mathrm{QP}})\psi_i(\mathbf{r}')d\mathbf{r}' = E_i^{\mathrm{QP}}\psi_i(\mathbf{r}), \quad (1)$$

where $\mathbf{r}$ is the electron position, ψ_i is the quasiparticle (QP) wavefunction for state i, E_i^{QP} is the quasiparticle excitation energy, h_0 is a single-particle (or mean-field) operator, and Σ is a non-Hermitian, non-local, and frequency-dependent operator named the self-energy.

In the GW method, we approximate the self-energy Σ to the first order in the Green's function $G(\mathbf{r}, \mathbf{r}'; \omega)$ and the screened Coulomb interaction $W(\mathbf{r}, \mathbf{r}'; \omega)$ as a frequency convolution,

$$\Sigma(\mathbf{r}, \mathbf{r}'; \omega) = i \int \frac{d\omega'}{2\pi} e^{-i\delta\omega'} G(\mathbf{r}, \mathbf{r}'; \omega - \omega') W(\mathbf{r}, \mathbf{r}'; \omega'), \quad (2)$$

where ω and ω' represent frequencies and $\delta \to 0^+$. The core GW computation is the evaluation of matrix elements Σ_{lm} of the self-energy operator between orbital wavefunctions l and m. Σ_{lm} is expressed as a sum over orbitals n [2], [3],

$$\Sigma_{lm} = \frac{i}{2\pi} \sum_{nGG'} M_{ln}^{-G^*} M_{mn}^{-G'} \int_0^\infty d\omega \frac{\epsilon_{GG'}^{-1}(\omega)\, v_{G'}}{E_l - E_n - \omega}, \quad (3)$$

where G corresponds to planewave basis elements, $\epsilon_{GG'}^{-1}$ represents a square matrix (inverse matrix of the dielectric matrix $\boldsymbol{\epsilon}$) representing the dielectric screening in the system, E is the orbital energy, $v_{G'}$ is the Coulomb interaction in reciprocal space and M are the matrix elements between orbitals, $M_{mn}^G = \int d\mathbf{r}\, \psi_m^*(\mathbf{r}) e^{i\mathbf{G}\cdot\mathbf{r}} \psi_n(\mathbf{r})$.

Besides the large complexity introduced by the non-local nature (GG' dependence) and summation over states (n), Eq. 3 involves a frequency convolution. The frequency dependence of $\boldsymbol{\epsilon}$ can be efficiently captured using the generalized plasmon-pole (GPP) approximation [2]. For the first-principles GW calculations using the GPP model, the static dielectric matrix $\boldsymbol{\epsilon}(\omega{=}0)$ is explicitly computed and thereafter extended to nonzero frequencies according to the model.

In BerkeleyGW [5], the GW computational workflow is implemented in two standalone executables, the Epsilon and Sigma modules. Epsilon computes the dielectric matrix $\boldsymbol{\epsilon}$ and its inverse $\boldsymbol{\epsilon}^{-1}$; subsequently, Sigma computes the self-energy $\Sigma(\mathbf{r}, \mathbf{r}', \omega)$ and its corresponding matrix elements Σ_{ml}, using $\boldsymbol{\epsilon}^{-1}$ as an input. The workflow is represented as Epsilon $\xrightarrow{\epsilon^{-1}}$ Sigma $\xrightarrow{\Sigma_{ml}}$ output. These two modules combined give the solution to Dyson's equation (Eq. 1), i.e. a set of quasiparticle excitation energies E^{QP}. An overview of the computational cost and memory requirement for the most time-consuming components of Epsilon and Sigma is shown in Tab. I.

TABLE I
COMPUTATIONAL AND MEMORY COMPLEXITY FOR EPSILON AND SIGMA

	Kernel	Computation	Memory
Epsilon	MTXEL	$O(N_v N_c N_G^\psi \log N_G^\psi)$	$O(N_v N_c N_G)$
	CHI-0	$O(N_v N_c N_G^2)$	$O(N_v N_c N_G + N_G^2)$
	Inversion	$O(N_G^3)$	$O(N_G^2)$
Sigma	MTXEL	$O(N_\Sigma N_b N_G^\psi \log N_G^\psi)$	$O(N_b N_G)$
	GPP	$O(N_\Sigma N_b N_G^2)$	$O(N_G^2 + N_b N_G)$

N_v is the number of valence (occupied) bands, N_c the number of conduction (empty) bands, $N_b = N_v + N_c$ the total number of bands, N_G^ψ and N_G are the planewave basis set size for the wavefunction and the dielectric matrix respectively, N_Σ the total number of quasiparticle energies (i.e. matrix elements of Σ). Each operation takes place once per spin channel, as we discuss later, except for Inversion, which happens once per system.

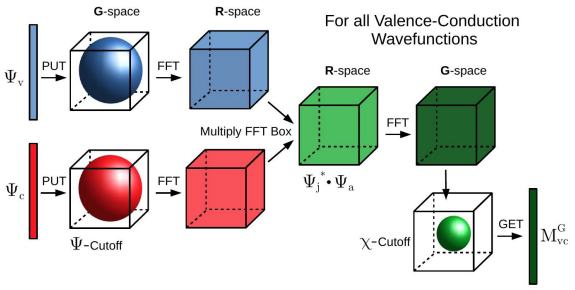

Fig. 2. Schematic representation of operations in Epsilon MTXEL for each pair of valence and conduction wavefunctions.

B. The Epsilon Module

Epsilon computes the inverse static dielectric matrix $\boldsymbol{\epsilon}^{-1}(\omega{=}0)$ employing a planewave basis of size N_G. The computed $\boldsymbol{\epsilon}^{-1}$ is a $N_G \times N_G$ double-complex matrix. Typically N_G is on the order of 100k for large-scale applications. The three major computational kernels in Epsilon are: the matrix elements (MTXEL), the static polarizability (CHI-0) and the dielectric matrix inversion (Inversion) kernels.

1) Matrix Element Kernel (MTXEL): The MTXEL kernel computes transition matrix elements M_{vc}^G between valence (v) and conduction (c) wavefunctions using a planewave basis. The schematic implementation shown in Fig. 2 is based on fast Fourier transforms (FFTs). MTXEL consists of an outer loop over N_v valence wavefunctions (ψ_v) and an inner loop over N_c conduction wavefunctions, and for each (v, c) pair a convolution is performed in real space for N_G planewave functions [5]. The resulting matrix elements M_{vc}^G, scaled by a (v, c) energy dependent factor, are stored as column-vectors in the matrix $\mathbf{M}$, which has N_G rows and $N_v \times N_c$ columns. The construction of MTXEL is embarrassingly parallelelized by distributing the $N_v \times N_c$ independent pairs over MPI tasks.

The MTXEL kernel calls cuFFT [30] for FFT operations on the GPU. As shown in Tab. I, the kernel has an $O(N^3 \log N)$ scaling for FLOPs and $O(N^3)$ scaling for memory. The main implication of this is that the host-device data copies will become a bottleneck if we naively copy data back and forth at every cuFFT call. On the other hand, offloading all data to device beforehand will exhaust the GPU memory very quickly due to the large amount of (v, c) pairs. To help alleviate these problems, we have adopted the following strategies.

- *Precomputing Conduction Wavefunctions FFTs*: The FFTs of conduction wavefunctions (red box in Fig. 2) are reused across all N_v valence wavefunctions (the outer loop), so these FFTs are pre-computed in a separate loop and saved on the GPU for later reuse. This causes a lock in of a certain amount of GPU memory, but it reduces the number of FFTs and memory copies by a factor two.
- *Batching Mechanism for Conduction Wavefunctions*: The

inner loop of N_c wavefunctions are processed in batches to avoid memory overflow on the GPU. Host non page-able (pinned) memory is used to allow for direct host-device data transfers with a higher bandwidth. The pinned buffer is kept under a certain size and is also reused between iterations to keep the allocation cost low.

- *Data Streams*: A number of data streams (or CUDA streams) are created and reused. Each stream is used to schedule the sequence of data copy and cuFFT operations of each cycle of the batch loop. This allows for multiple cycles to run concurrently on GPU and also overlap between GPU computation and device to host memory transfer across cycles.

2) Static Polarizability Kernel (CHI-0): CHI-0 is the most computationally demanding kernel of Epsilon, scaling as $O(N^4)$ with the system size (see Tab. I). The algorithmic motif is a large distributed matrix multiplication in the form of $\mathbf{M}\times\mathbf{M}^{\mathrm{T}^*}$ with $\mathbf{M}$ computed from MTXEL. For large systems, $\mathbf{M}$ has on the order of 100k rows and 10M columns, which poses challenges when computing CHI-0 in a distributed fashion. Since the parallelization pattern in MTXEL is such that different columns of $\mathbf{M}$ are distributed across different processors, the data layout in CHI-0 naturally takes a similar shape as shown in Fig. 3. Each MPI rank holds all the rows and its own subset of columns of $\mathbf{M}$ and calculates its contribution to χ_0. The matrix χ_0 is distributed with a 2D block-cyclic data layout, such that the subsequent linear algebra operations on χ_0 can be performed in parallel using libraries such as ScaLAPACK [31] and ELPA [32]. Due to the peculiar data layout in CHI-0, the efficient use of external libraries for such matrix multiplication (i.e. COSMA [33] and SLATE [34]) is not straightforward for multi-GPU, multi-node applications.

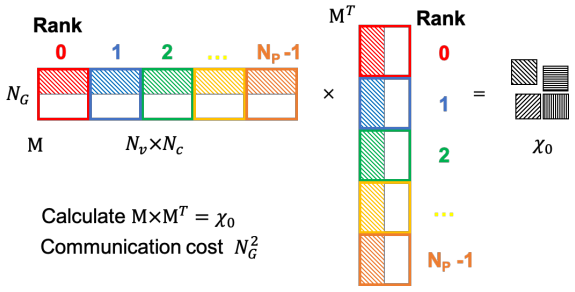

Fig. 3. Data layout and distributed calculation in Epsilon CHI-0. For all N_P processes, the total data communicated is N_G^2 (it does not depend on N_P).

To achieve optimal performance, we use the cuBLAS [35] library for the local GPU matrix multiplication (ZGEMM) on each MPI rank, and the following strategies:

- *Offloading Data Preparation Kernels*: Local buffers need to be prepared according to the χ_0's 2D block-cyclic layout before the GEMM operations. Such preparation can be performed on host (Host-Prep. algorithm in Tab. IV) or accelerated on device after offloading $\mathbf{M}$ over batches to avoid out-of-memory on GPU (Full-Offload algorithm). As shown later in Sec. VII, the acceleration of this kernel outperforms the data copy overhead and the recomputation batches even for the smallest test case.
- *Batching Mechanism*: To avoid hitting the memory limit on GPUs, we devise a batching mechanism to load columns of $\mathbf{M}$ based on device's maximum memory, the results then being accumulated to χ_0 from all batches.
- *Non-Blocking Cyclic Communication Scheme*: Distributed GEMM can be implemented via MPI collectives where each rank calculates its contribution to the ith quarter of χ_0 at the ith iteration of the process, $1 \leq i \leq N_P$, and the blocking MPI_Reduce call sums all the contributions together to obtain the final results for that χ_0 quarter (each quarter in Fig. 3). In our scheme, instead of blocking MPI collectives, we use point-to-point non-blocking MPI_Isend and MPI_Irecv. The communication pattern is a ring, and each MPI rank R_{my} only communicates with its neighbouring ranks, $R_{\mathrm{my}+1}$ and $R_{\mathrm{my}-1}$ (cyclically). As shown in Fig. 4, at each iteration i, $1 \leq i \leq N_P$, rank R_{my} calculates its contribution for the $(\mathrm{my}-i)$th quarter of χ_0, and at the same time receives a partial sum about that same quarter from rank $R_{\mathrm{my}-1}$ (during the first iteration, only zeros are received). The local ZGEMM results from the GPU are then added on top of this partial sum (represented by the black bands, and done on the host), before being sent on to rank $R_{\mathrm{my}+1}$ in the next iteration. The MPI communication (gray bands) can largely overlap with the GPU computation (colored bands with pattern shading), hence providing a significant performance boost.

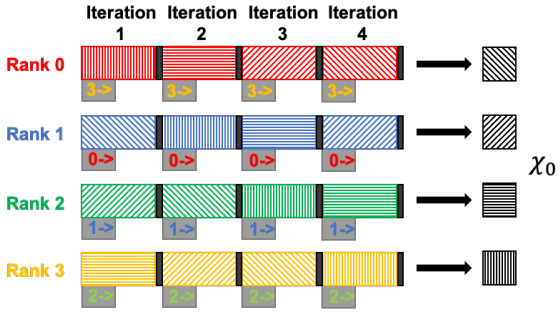

Fig. 4. Non-blocking cyclic MPI communication based on point-to-point calls in Epsilon CHI-0 exemplify with 4 MPI tasks. Colored bands represent GPU computation with the same shading patterns for quarters of χ_0 as in Fig. 3. Gray bands represent MPI communication (MPI_Wait) synchronizing the non-blocking send and receive. Black bands represent copying local ZGEMM results from GPU to CPU, adding them to the received partial sum from $R_{\mathrm{my}-1}$, and posting MPI_Isend/MPI_Irecv calls for the next iteration.

Overall, the parallel algorithms developed in CHI-0 display a FLOPs scaling of $O(N_v N_c N_G^2/N_p)$, memory scaling of $O\left((N_G^2 + N_v N_c N_G)/N_p\right)$ and a communication volume per MPI task of $O(N_G^2)$ when employing N_p MPI ranks.

3) Static Inverse Dielectric Kernel (Inversion): The final step of Epsilon is an algebraic matrix inversion, usually taking a very small fraction of the execution time. In this step, χ_0 is multiplied by the Coulomb potential to give the

dielectric matrix ϵ, which is then algebraically inverted to give the inverse dielectric matrix ϵ^{-1}. This step is currently performed using LU decomposition and triangular inversion in ScaLAPACK on the CPU, and will be ported to the GPU when accelerated distributed libraries will be available.

C. The Sigma Module

Sigma calculates the quasiparticle energies of electrons E_i^{QP} by solving the Dyson's equation (Eq. 1), using ϵ^{-1} computed in Epsilon. The most expensive computational part in Sigma is the evaluation of N_Σ matrix elements of the self-energy operator Σ_{lm}, each displaying a computational cost of $O(N^3)$ with the system size, where N_Σ can vary from tens for simpler systems, to thousands for complex applications. Since the evaluation of each self-energy matrix element is independent, a two-level parallelization strategy is implemented:

1) Inter-pool parallelization: the total number of MPI tasks N_p is divided into N_{pools} pools of equal size $N_p^{\text{pool}} = N_p/N_{\text{pools}}$. The N_Σ self-energy matrix elements are then distributed over pools, each pool working independently on $N_\Sigma^{\text{pool}} = N_\Sigma/N_{\text{pools}}$ quasiparticles.
2) Intra-pool parallelization: parallelization of the work required for the evaluation of each self-energy matrix element across multiple processors within a pool.

The first level of parallelization is straightforward, while the second level involves the implementation of Eq. 3 using the GPP model. For each pair of (l, m) wavefunctions, the working equation can be rewritten in the compact form as:

$$\Sigma_{lm} = \sum_{n}^{N_b} \sum_{GG'}^{N_G} M_{Gn}^{l*} [P(E_n)]_{GG'} M_{G'n}^{m}, \tag{4}$$

where $\mathbf{P}(E_n)$ is an $N_G \times N_G$ matrix dependent on index n (via E_n argument), and $\mathbf{M}^l$ and $\mathbf{M}^m$ are $N_G \times N_b$ matrices grouping the matrix elements for the l and m indices, respectively. $\mathbf{P}(E_n)$ depends on the n^{th} column-vector of $\mathbf{M}^l$ and $\mathbf{M}^m$, so that Eq. 4 represents a data reduction across these matrices with a complex matrix-vector interdependence.

These operations are performed by the (Sigma) GPP kernel, and the computation of the elements in $\mathbf{M}$ is performed by the (Sigma) MTXEL kernel. Sigma processes one self-energy matrix element at a time, i.e. an (l, m) pair, through an outer loop over N_Σ^{pool}. Within each iteration, Sigma then executes sequentially the MTXEL kernel and the GPP kernel. Tab. I summarizes the computational cost and memory requirement for Sigma, and Fig. 5 gives a schematic representation of the data distribution and basic operations for the intra-pool parallelization. The columns of the $\mathbf{M}$ matrix are distributed across different processors, and they are communicated in an all-to-all fashion. Similar to Epsilon, there are also two communication schemes, the MPI collectives (MPI_Bcast) based, and the non-blocking cyclic scheme [6], similar to the one previously described for the Epsilon module. Overall, for a parallel execution employing N_p processors, Sigma displays a computational complexity of $O(N_\Sigma N_b N_G^2/N_p)$ (FLOPs), a memory complexity of $O\left((N_G^2 + N_b N_G)/N_p^{\text{pool}}\right)$, and a $O(N_\Sigma^{\text{pool}} N_b N_G)$ communication volume per MPI task.

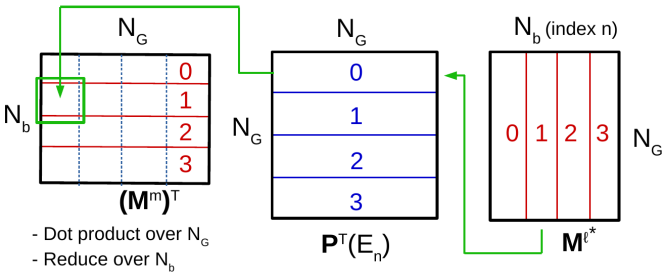

Fig. 5. Schematic representation of the intra-pool parallelization in Sigma, with 4 MPI processes as an example.

The Sigma kernels are detailed below.

1) Sigma MTXEL Kernel: The MTXEL kernel is largely similar to that in Epsilon, but the outer loop is restricted solely to the ψ_l and ψ_m wavefunctions for the currently calculated matrix element Σ_{lm}, and the inner loop runs over all bands $N_b = N_v + N_c$. Each MPI task computes its N_b^{distr} subset of N_b bands and stores the results in the local portion of $\mathbf{M}^l$ and $\mathbf{M}^m$ matrices. The loop is performed over batches as well (see Sec. V-B1) with a typical size of 10 to 40.

2) Sigma GPP Kernel: The entire computation of Sigma GPP is performed on the device, including the many matrix-vector operations in Eq. 4, with $\mathbf{P}(E_n)$ computed on the fly for each index n. The algorithm consists of three nested loops as shown in Fig. 6. The innermost loop iterates over bands (index n), and each processor is responsible for N_b^{distr} columns of $\mathbf{M}$ for the current communication cycle. The two outer loops run across N_G^{distr} columns and all N_G rows of $\mathbf{P}(E_n)$. The relation between these parameters is $N_b^{\text{dist}} < N_G^{\text{dist}} \ll N_G$, and this dictates a lot of the design choices in our implementation:

a) The first two loops together provide sufficient parallelism for GPUs and are hence collapsed, while the N_b^{dist} loop is unrolled on each thread to gain arithmetic intensity and leverage data locality for G' and G-related arrays.
b) The n loop is further divided into band blocks (Fig. 6), with each block containing $N_{b-\text{block}}^{\text{distr}}$ iterations. The choice for $N_{b-\text{block}}^{\text{distr}}$ is so that it is small enough for n-related arrays to fit into L1 and L2 caches (and be kept there).
c) To further reduce data movement, the combined G' and G iteration space is tiled with small blocks of G' and G iterations executed one at a time. Since G varies more frequently than G', the loop tiling is designed so that different SMs share some (not all) G-related arrays, while some (not all) G'-related arrays are kept in L2 (L1 is too small to be used this way).
d) Shared memory is used for reduction across G', G and n iterations on each thread and for reduction across threads within a thread block. Some small, frequently used arrays are kept in shared memory wherever possible. For warp-wide reductions, the primitive `__shfl_down_sync` is used for register-level data exchange, which is more efficient than utilizing shared memory.

e) Long-latency instructions such as divides and square roots for complex numbers are replaced with reciprocals and absolute values wherever possible, which helps reduce the 'math pipe throttle' type of warp stalls [36] and significantly improves warp issue rates.
f) Parameters such as the number of threads per thread block and the number of thread blocks are more optimized to gain maximum occupancy, given the constraints posed by high register and high shared memory usage.
g) The band blocks in b) are placed on different CUDA streams and the results (partial sums of self-energy) are transferred back from each kernel on each stream to the host asynchronously. The results are then further reduced on the CPU, across all MPI processes.
h) Similar to Epsilon, the non-blocking cyclic communication scheme (see Sec.V-B2) is employed in Sigma, and maximum asynchronicity is available to leverage overlap between GPU/CPU computation, host-device transfers, and MPI communication.

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 σ over GPU thread blocks, CUDA streams, and MPI ranks

Fig. 6. Loop structure of the Sigma GPP kernel on the GPU. $\sigma_{b-\text{block}}$ is the partial contribution to Σ_{lm} for the current band block.

To achieve optimal performance on a particular architecture and for a problem size, fine tuning is required for a large number of parameters, such as for the loop tiling and shared memory usage. Hence, we provide both a standard version of the GPP kernel, which runs out of the box and can achieve decent performance for all common use cases of BerkeleyGW, and a more optimized version which incorporates all the optimizations discussed above and is specifically tuned to the V100 GPUs on Summit. The latter is denoted with a asterisk, e.g., Si-2742* in Tab. VI, and most results presented in Sec. VII are for the standard version, unless specifically stated.

VI. How Performance Was Measured

A. Benchmarks and Performance Metrics

The benchmarks employed to assess performance in this paper are divacancy defect systems in bulk semiconductors, namely silicon (Si) and silicon carbide (SiC), with a range of cell sizes from 214 atoms to 2,742 atoms. These structures have distinct computational complexities and memory footprints and Tab. II illustrates some of the most important parameters and computational requirements. In terms of HPC performance, we report the time-to-solution (for both CPU and GPU implementations), strong and weak scaling, double-precision performance (PFLOP/s) and energy efficiency (kJ/FLOP).

TABLE II
Minimum computational requirements for benchmarks

Parameters	Si-214	Si-510	Si-998	SiC-998	Si-2742
N_{spin}	1	1	1	2 (↑/↓)	1
N_G^{ψ}	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_Σ	Variable, up to 256 per spin				
Epsilon PFLOPs	2.5	80.5	1164	10,091	66,070
Epsilon Memory (TB)	0.45	6.07	45.1	135	934
Epsilon Comm.Vol. (GB)	3.92	22.5	85.3	1428	640
Sigma PFLOPs	0.127	1.71	12.6	58.2	260.7
Sigma Memory (GB)	6.19	34.3	133.8	791.4	1006
Sigma Comm.Vol. (GB)	2.27	12.8	48.5	77.2	365.4

We use the same notation as in Tab. I, and ↑ and ↓ represent spin up and spin down configurations respectively, for calculations with two components (i.e., when $N_{\text{spin}} = 2$). Com.Vol. is the total communication volume per MPI task. Note that the Sigma PFLOPs and Comm.Vol. per MPI task, are measured per self-energy matrix element, and that the Sigma Memory is per pool, accounting for device memory only.

B. Estimation of Operation Counts

To gauge the amount of operations performed in Epsilon and Sigma, we use the canonical FLOP count from the most compute-intensive components of the code as a lower bound, and use tools such as NVIDIA Nsight Compute [37] to validate. For Epsilon, the most compute-intensive part is CHI-0, especially for large calculations, and within CHI-0, complex matrix multiplications (ZGEMM) dominate the calculations. The FLOP estimation for ZGEMMs is

$$\text{FLOPs count (ZGEMM)} = 8 \times N \times K \times M, \qquad (5)$$

which has been validated in [6]. For our benchmarks, $N = M = N_G$ and $K = N_v \times N_c$ for each spin.

TABLE III
Validation of operation count for Sigma (TFLOPs)
(Meas. - Measured, Est. - Estimated)

System	N_Σ	N_b	N_G	Meas.	Est.	% Est./Meas.
Si-510	2	3223	9315	90.49	90.25	99.7
Si-510	2	4261	9315	119.54	119.32	99.8
SiC-214	4	2309	2945	12.96	12.93	99.7
SiC-214	4	3409	6979	107.23	107.17	99.9
Si-510*	2	4261	9315	112.18	112.05	99.9
SiC-214*	4	3409	6979	100.63	100.64	100.0

For Sigma, the GPP kernel takes up 95% of the FLOPs for large simulations and the remaining FLOPs come mostly from a few ZGEMMs (performing a partial summation) and FFTs. The computational complexity for GPP is $O(N_\Sigma N_b N_G^2)$ (see Tab. I) and, by performing a series of tests using the Si-214 system, we have determined a linear relationship between the FLOPs and several parameters as $\alpha N_\Sigma N_b N_G^2$, with α being a constant prefactor. This relationship was validated with Nsight Compute measurements for other systems in Tab. III, with

less than 1% discrepancy between estimated and measured. The prefactor for the standard GPP kernel (see Sec. V-C2) is $\alpha = 153.36$ and for the more optimized version is $\alpha = 151.53$. Thus, the total FLOPs count for GPP is

$$\text{FLOPs count (GPP)} = (153.36 + 8) \times N_\Sigma N_b N_G^2, \quad (6)$$

where the additional 8 FLOPs comes from ZGEMM when static remainder is employed. In both Epsilon and Sigma, we are underestimating the operation counts as we only include the GPU operations from the most compute-intensive part of the code in the FLOPs estimation. We note that the more optimized kernel does not employ the ZGEMMs for the partial sum, therefore the additional 8 FLOPs are not included in its total FLOPs count.

C. HPC Systems

Two leadership HPC systems are used for performance assessment in this work, namely Summit at OLCF [38] and Cori at NERSC [39]. Cori (Haswell and GPU partitions) is used for some of the CPU benchmarks and GPU tests while Summit is employed for the large-scale full-system runs.

Summit consists of 4,608 nodes, each with two IBM POWER9 CPUs and six NVIDIA V100 GPUs. Each POWER9 processor has 21 cores and is connected to 3 GPUs via NVLink. Each GPU has 80 SMs (Streaming Multiprocessors) and 16GB HBM2 memory on device. There is 512 GB of DDR4 memory on the host and 1.6TB of node-local non-volatile memory that can be used as a burst buffer.

Cori has two main partitions, Haswell and KNL, and a 18-node GPU chassis for code development activities. There are 2,688 Haswell nodes, each with two Intel Xeon E5-2698v3 CPUs and 128 GB of DDR4 memory. The GPU chassis contains eight NVIDIA V100 GPUs and two Intel Xeon 6148 Skylake CPUs on each node. Most of Cori-related results in Sec. VII are from the Haswell partition and the GPU chassis.

VII. Performance Results

In this section, we report performance results such as the time-to-solution, FLOP/s performance, percentage of peak, and energy efficiency. For the GPU implementation, all runs are configured with one GPU per MPI process and all CPUs on the host are equally divided among the MPI processes and run as OpenMP threads. The full-system runs take place on Summit and we have based our percentage of peak calculation on 43.57 TFLOP/s per node, i.e. 200.79 PFLOP/s for 4608 nodes as reported in [40].

A. CPU versus GPU Implementation

Thanks to the innovations in Sec. V-B, Epsilon exhibits an overall 18.6x speedup in runtime when moving from the CPU implementation to GPUs for a moderate sized system, Si-214. As reported in Tab. IV, the two dominant kernels MTXEL and CHI-0 (Full-Offload) have gained 25 times and 23 times speedup in runtime. The Full-Offload algorithm (see Sec.V-B2) has also contributed to the performance improvement by reducing host-device data transfers and executing data preparation routines on GPUs – a 2x speedup over the Host-Prep version for CHI-0.

TABLE IV
Runtime comparison of CPU and GPU implementations of Epsilon on Cori GPU (2 nodes) at NERSC for Si-214

	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

For Sigma, the GPU implementation is 4.3x faster on one Summit node than its CPU implementation on 20 Cori Haswell nodes – a 86 times node-to-node speedup (Fig. 7). This is much higher than the peak throughput ratio of 36 between two machines (43.6 TFLOP/s to 1.2 TFLOP/s), demonstrating better resource usage on the GPUs than on the CPUs.

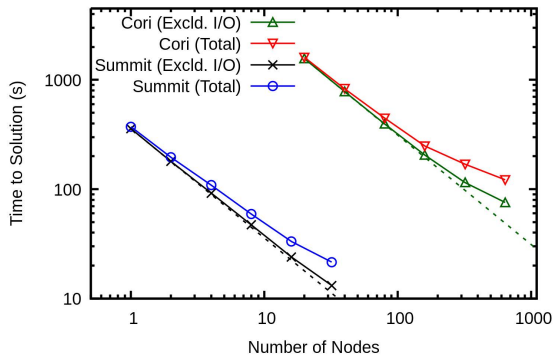

Fig. 7. Runtime comparison of Sigma on Cori Haswell (CPU) and Summit (GPU) for the Si-510 system with the evaluation of 4 quasiparticles states. The total time to solution is roughly 6 minutes on 1 Summit node (6 GPUs), and 27 minutes on 20 Cori nodes (640 CPUs).

With modern HPC design, energy consumption is emerging as a fundamental metric to assess the efficiency of current implementations. Fig. 8 shows the energy analysis we have performed for a range of GW calculations on Summit and on Cori Haswell. The average power per node is taken from the TOP500 list [40], [41] for Summit (2,200 W) and the latest study at NERSC [42] for Cori Haswell (400 W). These numbers include power consumption not only from the computing units (CPUs and GPUs), but also from cooling, network, file systems and other operation-related units, and hence they give a more realistic picture of the energy consumption. With energy consumption formulated as the average power per node times runtime, Fig. 8 shows that for the same calculation, there is about 7x energy saving for CHI-0 and 16x for Sigma when switching from Cori Haswell to Summit.

With performance portability in mind, we have also developed an OpenACC version in tandem with the CUDA version. The OpenACC version is compiled with PGI compiler, and PGI Fortran interfaces are used for the cuBLAS and cuFFT library calls. Tab. V shows that for a medium-sized problem Si-510, the OpenACC version presents a less than 10% performance difference compared to the CUDA

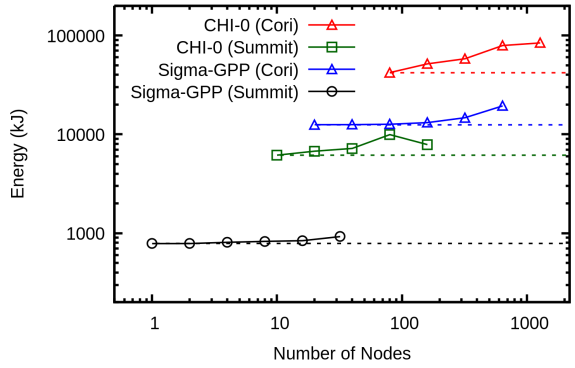

Fig. 8. Energy efficiency comparison of CHI-0 and Sigma GPP kernels on Cori Haswell and Summit for Si-510 with 4 quasiparticles.

TABLE V
PERFORMANCE COMPARISON OF CUDA VS OPENACC (SIGMA GPP)

Runtime	6 GPUs	12 GPUs	24 GPUs
CUDA	358 s	182 s	91 s
OpenACC	378 s	192 s	100 s

Results for the Si-510 system with 4 quasiparticles on Summit excluding I/O.

version. This provides a proof of concept that a directive-based programming model can be nearly as performant as CUDA for accelerating BerkeleyGW. Additionally, this version can be translated to OpenMP, with a similar syntax, which is planned to be supported on all major DOE pre-exascale and exascale supercomputers such as Perlmutter, Frontier, El Capitan and Aurora.

B. Weak Scaling

Fig. 9(a) shows the weak scaling pattern of Epsilon, where the number of GPUs utilized is scaled exactly to the problem size, i.e. $1824/126 \approx 15$ and $(998/510)^4 \approx 15$. The most computationally intensive kernel in Epsilon is CHI-0, which scales at $O(N^4)$ with the number of atoms in the system. The memory complexity for CHI-0 is $O(N^3)$ (see Tab. I), and the MPI communication is $O(N^2)$ (see Fig. 3). As the number of atoms increases from 510 to 998, the computational demand (FLOPs) per GPU remains almost constant, but the amount of memory required per MPI task decreases (i.e. $O(N^3)$ *vs* $O(N^4)$). With the reduced amount of data to process on each processor, there is less batching required (see Sec. V-B2). However, this benefit may not be enough to overcome the extra communication required by the quartically increasing number of processors, thus leading to 5% runtime increase for CHI-0.

For Sigma, the GPP kernel is the most computationally demanding component, and it scales at $O(N^3)$ with the system size, and linearly with the number of quasiparticle energies (N_Σ). To assess the weak scaling of GPP, we can either increase the system size for a fixed N_Σ, or increase N_Σ for a fixed system size. Fig. 9(b) shows the results for both scenarios, with Cases A, B and C for the first and C, D and E for the second. Excellent weak scaling is observed

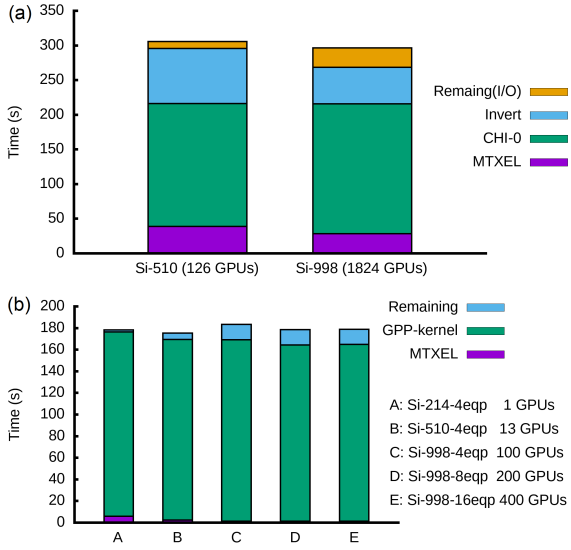

Fig. 9. Weak scaling performance of (a) Epsilon and (b) Sigma on Summit. For Epsilon the number of GPUs is scaled according to the $O(N^4)$ computational complexity. For Sigma the number of GPUs is scaled according to the $O(N^3)$ computational complexity in Cases A, B and C, and to the number of quasiparticles in Cases C, D and E.

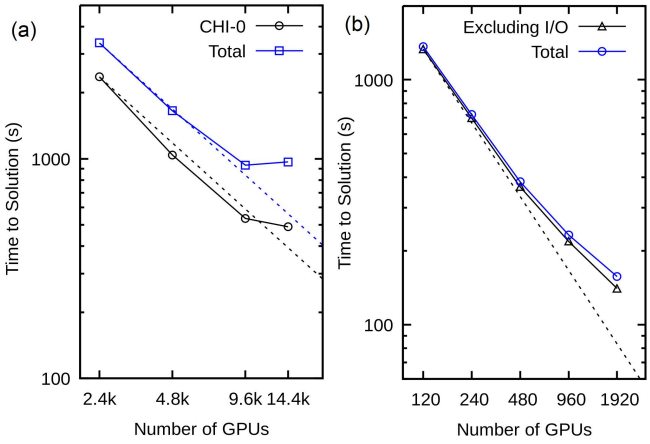

Fig. 10. Strong scaling performance for the SiC-998 system for (a) Epsilon and (b) Sigma (pool scaling with intra-pool parallelization).

in both scenarios, and the runtime variation between all five cases is less than 3%, thanks to the two-level parallelization strategy (see Sec. V-C). The 3% may have come from load imbalance issues caused by non-uniform matrix distribution (i.e. indivisible GPU numbers).

C. Strong Scaling

For the strong scaling, we focus on the two largest systems, SiC-998 and Si-2742. They each require tens of EFLOPs for Epsilon and Sigma calculation (see Tab. II).

Fig. 10(a) shows the strong scaling performance of Epsilon, both CHI-0 alone and the total runtime. The non-CHI-0 parts include MTXEL (GPU accelerated), Inversion (CPU only, using ScaLAPACK) and file I/O (for SiC-998, the input size is 230 GB and output 333 GB). Excellent scaling is observed up

to 9,600 GPUs, thanks to Epsilon's batching mechanism (see Sec. V-B2). At 9,600 GPUs, the entire CHI-0 kernel can be run within one batch, i.e. $135\times1024/9600/N_{\text{spin}} = 7.2$ GB (please see Tab. II for 135 TB, and there is 16 GB of HBM memory on the GPU). After this, the performance efficiency drops mainly due to (a) the loss of effective overlapping between GPU computation and MPI communication (computation time has become too short), and (b) load imbalance between nodes, caused by indivisible matrix dimensions and non-uniform distribution of the workload over the ring patterned communication.

For Sigma, as described in Sec.V-C, there are two levels of parallelization. One is over the independent quasiparticles (or pools) and the other is within the pool, across multiple processors (intra-pool). Fig. 10(b) shows the strong scaling behavior of the intra-pool parallelization for SiC-998, where 8 quasiparticle energies are calculated using 2 pools, with the pool size ranging from 60 GPUs to 960 GPUs. Excellent scaling performance is observed up to 480 GPUs per pool and 960 GPUs in total, and the deviation from linearity is mostly due to load imbalance between processors and loss of effectiveness of the overlap between GPU computaion and MPI communication.

With good strong scaling performance at 120 GPUs per pool, we investigate the inter-pool parallel efficiency for SiC-998. As shown in Fig. 11, the first five data points (green and violet) sweep through different numbers of pools from 5, 10, 20, 40 and to 80 pools. Almost linear efficiency is observed. Since 80 is the maximum possible number of pools (1 quasiparticle for each of the 2 spin channels), to scale further, we increase the pool size from 120 GPUs to 180, 240 and 342 GPUs (the last three black and blue markers), and end at the full system of Summit and 27,360 GPUs in total. Again, almost linear scaling is observed, even though I/O becomes an issue at extreme scale and drives the entire curve off linearly slightly. For Si-2742, a similar trend is shown in Fig. 11 from 30 pools, 60 pools, and 120 pools with 120 GPUs per pool (first three blue and yellow data points), to 120 pools with 180, 204 and 228 GPUs per pool (last three pairs of data points).

To alleviate from I/O issues, we have implemented a workflow that pre-stages input data into the node-local solid-state storage devices (SSDs) on Summit before the compute job runs. Before the pre-staging, the input data needs to be prepared in a suitable format for later parallel executions, and this preparation only needs to be done once for a given pool size (i.e. number of MPI tasks per pool). Fig. 11 shows that the Si-2742* system has significantly less I/O time, and more specifically, the two Si-2742 entries in Tab. VI show the exact timings, where the SSD utilization has reduced the I/O time from 226s to 23s. Note that the time spent in the pre-staging step (which is a separate job step from the compute job), is about 52s for the Si-2742* benchmark and it is not included in the 23s in Tab. VI. With 20.4 GB of input data being copied to each of the 4608 nodes on Summit, this results in an average aggregated bandwidth of 1.8 TB/s.

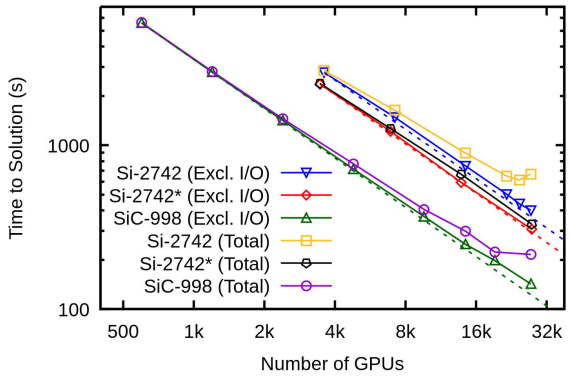

Fig. 11. Strong scaling performance of Sigma on Summit for the SiC-998 with 160 quasiparticles, Si-2742 with 120 quasiparticles, and Si-2742* with 128 quasiparticles, respectively. Labels denoted with an asterisk (*) are for the more optimized GPP kernel described in Sec. V-C2 and with the SSD utilization on Summit.

D. Full Summit Runs and Peak Performance

Fig. 12 presents the throughput performance of the two most computationally demanding kernels from Epsilon and Sigma, CHI-0 and GPP, in terms of FLOP/s rate versus the number of GPUs, and the highest values come from Sigma and are tabulated in Tab. VI.

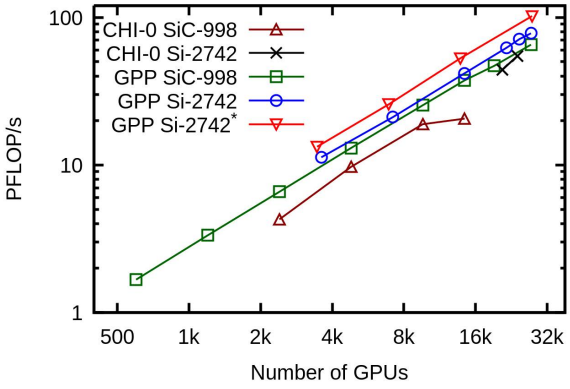

Fig. 12. Throughput (PFLOP/s) of Epsilon CHI-0 and Sigma GPP for SiC-998 and Si-2742 on Summit.

For the SiC-998 system, the overall throughput for CHI-0 is 18.9 PFLOP/s with 9,600 GPUs (at 27% of peak), and 20.6 PFLOP/s with 14,400 GPUs. For the Si-2742 system, the 4,000-node run reports a 54.8 PFLOP/s throughput and a 31.4% utilization of the 4,000-node peak performance of 172 PFLOP/s.

A series of Sigma calculations are reported in Tab. VI, where the highest throughput comes from the Si-2742 system with the more optimized GPP kernel (denoted by *). On 4,608 Summit nodes, we have achieved 105.9 PFLOP/s double-precision performance and 52.7% of the machine peak, as highlighted in gray.

TABLE VI
BEST PERFORMANCE FROM SIGMA

	# of GPUs	# of Pools	GPUs per Pool	Compute (s)	IO (s)	Throughput (PFLOP/s)	% of Peak
SiC-998	27,360	80	342	142	71	65.3	32.9
Si-2742	27,360	120	228	401	226	78.0	39.2
Si-2742*	27,648	128	216	307	23	102.1	50.9
Si-2742*	27,648	256	108	592	39	105.9	52.7

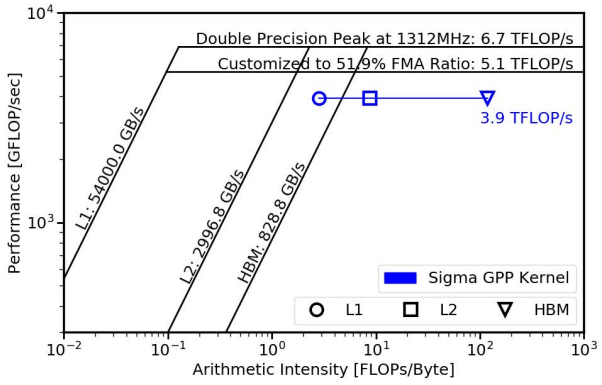

Fig. 13. Roofline analysis of the most optimized Sigma GPP kernel on Summit during the Si-2742 calculation. Note that there are hundreds of thousands of kernel invocations of GPP and this is only a random sample. This kernel has 51.9% of FMA instructions and at 3.92 TFLOP/s, it achieves 58.4% of the per-GPU peak, 6.7 TFLOP/s, and 76.9% of the more customized Roofline ceiling, 5.1 TFLOP/s, at the default clock frequency 1312 MHz.

This achievement is extraordinary considering the following challenges.

1) GPP performs a BLAS2 like operation rather than BLAS3 (matrix-matrix multiply like), so it is inherently bandwidth bound. However, via cache blocking, loop unrolling and other optimizations, we have moved the kernel to the compute bound regime (see Fig.13), with an Arithmetic Intensity from HBM of 132 FLOP/byte, which is well over the Roofline machine balance point [43], [44].
2) The complex data access pattern for multiple arrays presents a challenge in ensuring memory coalescence, without which we can easily suffer from low warp issue rates caused by 'long scoreboard' (global memory) or 'short scoreboard' (shared memory) stalls [36].
3) The existence of long-latency instructions in the kernel, such as divides and square roots causes a certain level of low warp issue and warp execution rates.
4) Due to the complex data type and the inter-dependency of the variables, the kernel uses a lot of registers (80-90 per thread) and shared memory (24-36KB per thread block); this limits how many warps can be scheduled, reduces occupancy and the kernel's latency-hiding capability.
5) There is a 51.9% ratio of FMAs (fused multiply adds) out of all FP64 instructions measured with Nsight Compute. With the estimation of $(2 \times 0.519 + 0.481)/2 = 75.9\%$ [44], the best performance achievable is 5.1 TFLOP/s for V100 at the standard clock frequency 1312 MHz (see Fig.13). GPP achieves 3.92 TFLOP/s, 76.9% of that peak, despite the above challenges.

These limiters are commonly seen in large scale HPC applications. Therefore, the implementation and optimization techniques presented in this paper can be widely generalized to other codes and other domains.

VIII. IMPLICATIONS

We have detailed the innovations we implemented in the GW calculations in BerkeleyGW and the performance results from simulating several defect complexes in silicon and silicon carbide, semiconductors that continue to be broadly important in electronics technology and quantum information. The advances described here enable the well-founded GW method to be applied to challenging, fundamental questions related to defects in established and emerging materials, such as two-dimensional transition metal dichalcogenides and materials projected for use in quantum information systems, replacing semi-empirical methods.

This work demonstrates the possibility for large-scale GW calculations on current HPC systems, and has profound scientific and computational implications. First, a supercell of over 2,700 atoms and 10,000 electrons is simulated in minutes, pushing the limit of GW calculations to the ten-thousand electron regime and allowing for an unprecedented accuracy for excited-state properties of complex materials. Second, the GW workflow scales to the full size of Summit with 27,648 GPUs and achieves 105.9 PFLOP/s performance and 52.7% of the peak. Excellent strong and weak scaling is shown and nearly 20 times of energy savings is observed, with both highly performant and portable approaches implemented. This work sets a new milestone for large-scale GW electronic-structure calculations and opens up new possibilities for high-fidelity complex materials science studies in the exascale timeframe.

IX. 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.

REFERENCES

[1] R. O. Jones and O. Gunnarsson, "The density functional formalism, its applications and prospects," *Rev. Mod. Phys.*, vol. 61, no. 3, p. 689, 1989.

[2] M. S. Hybertsen and S. G. Louie, "Electron correlation in semiconductors and insulators: Band gaps and quasiparticle energies," *Phys. Rev. B*, vol. 34, no. 8, p. 5390, 1986.

[3] M. Hybertsen and S. G. Louie, "First-principles theory of quasiparticles: calculation of band gaps in semiconductors and insulators," *Phys. Rev. Lett.*, vol. 55, no. 13, p. 1418, 1985.

[4] J. Weber, W. Koehl, J. Varley, A. Janotti, B. Buckley, C. Van de Walle, and D. D. Awschalom, "Quantum computing with defects," *PNAS*, vol. 107, no. 19, pp. 8513–8518, 2010.

[5] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "Berkeleygw: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," *Comput. Phys. Commun.*, vol. 183, no. 6, pp. 1269–1289, 2012.

[6] M. Del Ben, F. H. da Jornada, A. Canning, N. Wichmann, K. Raman, R. Sasanka, C. Yang, S. G. Louie, and J. Deslippe, "Large-scale gw calculations on pre-exascale hpc systems," *Comput. Phys. Commun.*, vol. 235, pp. 187 – 195, 2019.

[7] P. Umari, G. Stenuit, and S. Baroni, "Gw quasiparticle spectra from occupied states only," *Phys. Rev. B*, vol. 81, no. 11, p. 115104, 2010.

[8] F. Giustino, M. L. Cohen, and S. G. Louie, "Gw method with the self-consistent sternheimer equation," *Phys. Rev. B*, vol. 81, no. 11, p. 115105, 2010.

[9] M. Govoni and G. Galli, "Large scale gw calculations," *J. Chem. Theory Comput.*, vol. 11, no. 6, pp. 2680–2696, 2015.

[10] H.-V. Nguyen, T. A. Pham, D. Rocca, and G. Galli, "Improving accuracy and efficiency of calculations of photoemission spectra within the many-body perturbation theory," *Phys. Rev. B*, vol. 85, no. 8, p. 081101, 2012.

[11] T. A. Pham, H.-V. Nguyen, D. Rocca, and G. Galli, "Gw calculations using the spectral decomposition of the dielectric matrix: Verification, validation, and comparison of methods," *Phys. Rev. B*, vol. 87, no. 15, p. 155148, 2013.

[12] A. Marini, C. Hogan, M. Grüning, and D. Varsano, "Yambo: an ab initio tool for excited state calculations," *Comput. Phys. Commun.*, vol. 180, no. 8, pp. 1392–1403, 2009.

[13] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *J. Phys. Cond. Matter*, vol. 21, no. 39, p. 395502, 2009.

[14] X. Gonze, B. Amadon, P.-M. Anglade, J.-M. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Côté *et al.*, "Abinit: First-principles approach to material and nanosystem properties," *Comput. Phys. Commun.*, vol. 180, no. 12, pp. 2582–2615, 2009.

[15] G. Kresse and J. Hafner, "Ab initio molecular dynamics for liquid metals," *Phys. Rev. B*, vol. 47, no. 1, p. 558, 1993.

[16] G. Kresse and J. Hafner, "Ab initio molecular-dynamics simulation of the liquid-metal–amorphous-semiconductor transition in germanium," *Phys. Rev. B*, vol. 49, no. 20, p. 14251, 1994.

[17] M. Schlipf, H. Lambert, N. Zibouche, and F. Giustino, "Sternheimergw: a program for calculating gw quasiparticle band structures and spectral functions without unoccupied states," *Comput. Phys. Commun.*, vol. 247, p. 106856, 2020.

[18] P. Liu, M. Kaltak, J. c. v. Klimeš, and G. Kresse, "Cubic scaling GW: Towards fast quasiparticle calculations," *Phys. Rev. B*, vol. 94, p. 165109, Oct 2016.

[19] J. Wilhelm, D. Golze, L. Talirz, J. Hutter, and C. A. Pignedoli, "Toward GW calculations on thousands of atoms," *J. Phys. Chem. Lett.*, vol. 9, no. 2, pp. 306–312, 2018, pMID: 29280376.

[20] M. Kim, G. J. Martyna, and S. Ismail-Beigi, "Complex-time shredded propagator method for large-scale GW calculations," *Phys. Rev. B*, vol. 101, p. 035139, Jan 2020.

[21] D. Neuhauser, Y. Gao, C. Arntsen, C. Karshenas, E. Rabani, and R. Baer, "Breaking the theoretical scaling limit for predicting quasiparticle energies: The stochastic gw approach," *Phys. Rev. Lett.*, vol. 113, p. 076402, Aug 2014.

[22] D. Jacquemin, I. Duchemin, and X. Blase, "Benchmarking the bethe–salpeter formalism on a standard organic molecular set," *J. Chem. Theory Comput.*, vol. 11, no. 7, pp. 3290–3304, 2015, pMID: 26207104.

[23] F. Bruneval, T. Rangel, S. M. Hamed, M. Shao, C. Yang, and J. B. Neaton, "molgw 1: Many-body perturbation theory software for atoms, molecules, and clusters," *Comput. Phys. Commun.*, vol. 208, pp. 149 – 161, 2016.

[24] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, and K. Reuter, "Ab initio molecular simulations with numeric atom-centered orbitals," *Comput. Phys. Commun.*, vol. 180, no. 11, pp. 2175 – 2196, 2009.

[25] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, and D. Sánchez-Portal, "The siesta method for ab initio order-n materials simulation," *J. Phys. Condens. Matter*, vol. 14, no. 11, p. 2745, 2002.

[26] T. D. Kühne, M. Iannuzzi, M. Del Ben, V. V. Rybkin *et al.*, "CP2K: An electronic structure and molecular dynamics software package - quickstep: Efficient and accurate electronic structure calculations," *J. Chem. Phys.*, vol. 152, no. 19, p. 194103, 2020.

[27] "Elk: an all-electron full-potential linearised augmented-plane wave (lapw) code," http://elk.sourceforge.net/.

[28] A. Gulans, S. Kontur, C. Meisenbichler, D. Nabok, P. Pavone, S. Rigamonti, S. Sagmeister, U. Werner, and C. Draxl, "Exciting: a full-potential all-electron package implementing density-functional theory and many-body perturbation theory," *J. Phys. Condens. Matter*, vol. 26, no. 36, p. 363202, 2014.

[29] J. Brooks, G. Weng, S. Taylor, and V. Vlcek, "Stochastic many-body perturbation theory for moiré states in twisted bilayer phosphorene," *J. Phys. Condens. Matter*, vol. 32, no. 23, p. 234001, 2020.

[30] "cuFFT: NVIDIA CUDA Fast Fourier Transform library," https://developer.nvidia.com/cufft.

[31] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[32] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, "The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science," *J. Phys. Condens. Matter*, vol. 26, no. 21, p. 213201, 2014.

[33] G. Kwasniewski, M. Kabić, M. Besta, J. VandeVondele, R. Solcà, and T. Hoefler, "Red-blue pebbling revisited: Near optimal parallel matrix-matrix multiplication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[34] M. Gates, J. Kurzak, A. Charara, A. YarKhan, and J. Dongarra, "Slate: Design of a modern distributed and accelerated linear algebra library," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[35] "cuBLAS: NVIDIA CUDA GPU-accelerated implementation of the standard basic linear algebra subroutines (BLAS)," https://developer.nvidia.com/cublas.

[36] "NVIDIA Nsight Compute Kernel Profiling Guide," https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html.

[37] "NVIDIA Nsight Compute Profiling Tool," https://docs.nvidia.com/nsight-compute/NsightCompute/index.html.

[38] "The Summit Supercomputer at Oak Ridge National Laboratory," https://www.olcf.ornl.gov/summit/.

[39] "The Cori Supercomputer at NERSC, Lawrence Berkeley National Laboratory," http://www.nersc.gov/systems/cori/.

[40] "Top500 list," https://www.top500.org/.

[41] "TOP500 List - Energy Efficient High Performance Computing Power Measurement Methodology," https://www.top500.org/static/media/uploads/methodology-2.0rc1.pdf.

[42] S. Bhalachandra, B. Austin, and N. Wright, "A Year in the Life of the Power Consumption of a Supercomputer: A study of NERSC's Cori (Submitted)," *Supercomputing Conference 2020*.

[43] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, vol. 52, no. 4, 2009.

[44] C. Yang, T. Kurth, and S. Williams, "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system," *Concurrency and Computation: Practice and Experience*, p. 5547, 2019.