GPU acceleration of many-body perturbation theory methods in MOLGW with OpenACC

Young-Moo Byun¹ and Jejoong Yoo¹⃣

¹Department of Physics, Sungkyunkwan University, Suwon-si, Gyeonggi-do 16419, Korea
(Dated: December 5, 2023)

Quasiparticle self-consistent many-body perturbation theory (MBPT) methods that update both eigenvalues and eigenvectors can calculate the excited-state properties of molecular systems without depending on the choice of starting points. However, those methods are computationally intensive even on modern multi-core central processing units (CPUs) and thus typically limited to small systems. Many-core accelerators such as graphics processing units (GPUs) may be able to boost the performance of those methods without losing accuracy, making starting-point-independent MBPT methods applicable to large systems. Here, we GPU accelerate MOLGW, a Gaussian-based MBPT method with \( G_0W_0 \) calculations without compromising accuracy.\(^{12}\) GPUs were accelerators (special-purpose hardware) designed for computer graphics at first, but later general-purpose computing on graphics processing units (GPGPU) made it possible for GPUs to perform non-specialized computations that have typically been conducted by CPUs. On the hardware side, GPUs consisting of many lightweight cores can accelerate numerically intensive and massively parallel computations in a single instruction multiple threads (SIMT) fashion. Some modern GPUs supporting high-performance double-precision floating-point (FP64) computations are widely used for HPC scientific applications that require high precision and accuracy for reliability and stability, such as \( \text{ab initio} \) quantum chemistry applications.\(^{19,20}\)

On the software side, there are two popular and mature GPU programming models: compute unified device architecture (CUDA) and open accelerators (OpenACC). While CUDA is a low-level model and is an extension to C/C++/Fortran, \(^{21,22}\) like open multi-processing (OpenMP), OpenACC is a high-level model based on compiler directives.\(^{23}\) CUDA enables to harness the parallel computing power of GPUs, but CUDA programming could be complex, because it requires an understanding of the GPU hardware architecture and a significant modification of the original CPU source code. Also, the CUDA code runs only on NVIDIA GPUs. OpenACC is an alternative to CUDA to simplify parallel programming and make the parallel code portable across various kinds of platforms such as operating systems, compilers, CPUs, and GPUs.

As the creator of the Python programming language said “Maintainable code is more important than clever code,” maintainability is as important as performance in software development, including the development of HPC scientific applications. OpenACC achieves a balance between productivity and performance by enabling scientists and researchers to accelerate existing CPU codes on GPUs quickly with minimal programming effort, and thus is becoming increasingly popular.\(^{24}\) For example, the GPU port of VASP has lately transitioned

I. INTRODUCTION

A theoretical method capable of accurately and efficiently describing the excited-state properties of molecular systems is important for the rational design of molecules with desired properties. Many-body perturbation theory (MBPT) methods based on the one-body Green’s function \( G \) might be a good choice, because the \( GW \) method with \( W \) being the dynamically screened Coulomb interaction has been successfully used to calculate electronic excitations in solids for a few decades.\(^{25}\) For example, the non-self-consistent (one-shot) \( GW \) method \( (G_0W_0) \) starting from the Perdew–Burke–Ernzerhof (PBE) generalized gradient approximation to density-functional theory \( (G_0W_0@PBE) \) can accurately calculate the bandgap of semiconductors at a reasonably low computational cost.\(^{26}\)

However, \( G_0W_0@PBE \) fails for molecules, significantly underestimating the ionization energy (IE) of small \( sp \) and \( 3d \) molecules by \( \sim 0.5 \) and \( \sim 1.0 \) eV, respectively.\(^{27,28}\) A simple workaround to make the \( G_0W_0 \) method work for finite systems is to change its starting point from the PBE exchange-correlation functional to the Hartree–Fock (HF) exchange or hybrid functionals, which admix a fraction of non-local HF (exact) exchange with semilocal exchange.\(^{29}\) However, this workaround makes the \( G_0W_0 \) method empirical; \( G_0W_0 \) is system dependent and its results depend strongly on the choice of starting points.\(^{30,31}\) The quasiparticle self-consistent \( GW \) (qsGW) method can address the system and starting-point dependency issues of the \( G_0W_0 \) method,\(^{32,33}\) giving good results for both the bandgap of solids and the IE of atoms and molecules.\(^{34,35}\) However, the qsGW method is computationally expensive even for small systems, making qsGW calculations of large systems not feasible even on high-performance computing (HPC) supercomputers powered by modern multi-core central processing units (CPUs).

Graphics processing units (GPUs) may solve the efficiency problem of the qsGW method, enabling large-scale qsGW calculations without compromising accuracy.\(^{12,15}\)
from CUDA to OpenACC, because “OpenACC dramatically decreases GPU development and maintenance efforts.”

In recent years, MBPT methods for molecules with local orbitals have been implemented in a variety of electronic structure codes, including FIESTA,47,48 TURBOMOLE,49 CP2K,50 and MOLGW.51 Among those codes, MOLGW is a double-precision Fortran/C++/Python CPU code for MBPT excited-state calculations of molecular systems based on Gaussian basis sets. Although one of us has recently parallelized MOLGW with OpenMP, starting-point-independent MBPT methods, such as ρscGW, in OpenMP-parallelized MOLGW still are too time-consuming to be applied to large systems.52

In this work, we port MOLGW to the GPU using OpenACC to extend the range of its applicability while keeping the original CPU source code intact. We benchmark the performance of GPU-enabled MOLGW and find that the OpenACC version of MOLGW on desktop GPUs can outperform the OpenMP version on workstation and supercomputer CPUs.

The rest of this paper is structured as follows: First, we give a mathematical introduction to MBPT. Second, we overview our OpenACC implementation of MBPT methods in MOLGW. Third, we analyze and discuss our benchmark results for the performance of our GPU-accelerated MOLGW. Last, we summarize and conclude.

II. THEORETICAL BACKGROUND

In this section, we give a minimal and simplified introduction to MBPT related to OpenMP and OpenACC implementations in MOLGW. More details about MBPT can be found elsewhere.37,42,43 We use a simple and consistent notation: (i) we follow the notations in MOLGW implementation and application papers,37,41 (ii) we use Hartree atomic units, and (iii) we omit the complex conjugate notation, because we consider real wavefunctions.

A. Many-body perturbation theory (MBPT)

MBPT can be used to calculate electron addition and removal energies. In MBPT, the central quantity is the one-body Green’s function:

\[ G^\sigma (\mathbf{r}, \mathbf{r}', \omega) = \sum_i \frac{\varphi_i^\sigma (\mathbf{r}) \varphi_i^{\sigma^*} (\mathbf{r}')}{\omega - E_i^\sigma - i\eta} + \sum_a \frac{\varphi_a^\sigma (\mathbf{r}) \varphi_a^{\sigma^*} (\mathbf{r}')}{\omega - E_a^\sigma + i\eta}, \]  

(1)

where \( \sigma \) is the spin direction, \( \omega \) is frequency (energy), \( \eta \) is a positive infinitesimal, \( E_i^\sigma \) and \( \varphi_i^\sigma \) are one-electron energies and wavefunctions, respectively, and \( i \) and \( a \) run over occupied and empty (unoccupied or virtual) states, respectively. In the following, we omit space and frequency variables \((\mathbf{r}, \mathbf{r}', \omega)\) for notational simplicity whenever needed.

Once the non-interacting (bare) Green’s function \( G_0^\sigma \) is known, the interacting (dressed) Green’s function \( G^\sigma \) can be obtained by solving the Dyson equation:

\[ G^\sigma = G_0^\sigma + G_0^\sigma \Delta \Sigma^\sigma G^\sigma, \]  

(2)

where \( \Sigma^\sigma \) is the non-local, frequency-dependent (dynamical), and non-Hermitian self-energy, which accounts for many-body effects. Two popular approximations to the self-energy are the GW approximation and second-order Møller–Plesset perturbation theory (MP2), which are based on \( W \) and the bare (unscreened) Coulomb interaction \( v \), respectively.38,43

B. Self-consistent field (SCF)

In MBPT for molecules using localized orbitals, the molecular orbitals (MOs) and MO energies are used as one-electron wavefunctions and energies, respectively, to construct the Green’s function in Eq. (1). MOs can be expressed as a linear combination of atomic orbitals \( \phi_\mu \) (LCAO):

\[ \varphi_m^\sigma (\mathbf{r}) = \sum_\mu C_{\mu m}^\sigma \phi_\mu (\mathbf{r}), \]  

(3)

where \( C_{\mu m}^\sigma \) are the LCAO-MO coefficients. MOLGW uses Gaussian-type orbitals (GTOs) as AOs.

The LCAO-MO coefficients in Eq. (3) can be obtained from a self-consistent solution to a generalized eigenvalue problem:

\[ \mathbf{H}^\sigma \mathbf{C}^\sigma = \mathbf{S}^\sigma \mathbf{C}^\sigma \mathbf{c}^\sigma, \]  

(4)

where \( \mathbf{C}^\sigma \) is a matrix of LCAO-MO coefficients, \( \mathbf{c}^\sigma \) is a diagonal matrix of MO energies, \( \mathbf{S} \) is the AO overlap matrix:

\[ S_{\mu \nu} = \int d \mathbf{r} \phi_\mu (\mathbf{r}) \phi_\nu (\mathbf{r}), \]  

(5)

and \( \mathbf{H}^\sigma \) is the Hamiltonian matrix:

\[ H_{\mu \nu}^\sigma = T_{\mu \nu} + V_{\text{ext}, \mu \nu} + J_{\mu \nu} - K_{\mu \nu}^\sigma + \Sigma_{\mu \nu}^\sigma, \]  

(6)

where \( T \), \( V_{\text{ext}} \), \( J \), and \( K^\sigma \) are the kinetic energy, external potential energy, Hartree, and Fock exchange terms, respectively, and \( \Sigma^\sigma \) is the correlation part of the self-energy. When \( \Sigma^\sigma = 0 \) (neglecting electron correlation effects) in Eq. (6), Eq. (4) becomes the HF equation.

We parallelize the last three terms on the right side of Eq. (6) – \( J \), \( K^\sigma \), and \( \Sigma^\sigma \) – using OpenACC in this work. The Hartree term in Eq. (6) is given by

\[ J_{\mu \nu} = \sum_{\lambda \tau} (\mu \nu | \lambda \tau) D_{\lambda \tau}^\sigma, \]  

(7)

where \( (\mu \nu | \lambda \tau) \) are the 4-center two-electron Coulomb repulsion integrals:

\[ (\mu \nu | \lambda \tau) = \iint d \mathbf{r} d \mathbf{r}' \phi_\mu (\mathbf{r}) \phi_\nu (\mathbf{r}) \frac{1}{| \mathbf{r} - \mathbf{r}' |} \phi_\lambda (\mathbf{r}') \phi_\tau (\mathbf{r}'), \]  

(8)
Start

Calculate S, T and V_{ext}: Eqs. (5) and (6)

Calculate J and K: Eqs. (7), (8), (9) and (10)

HF? (Σ_{c} = 0)

Yes

Construct H: Eq. (6)

No

Solve the HF/qsGW/qsMP2 equation: Eq. (4)

SCF converged?

Yes

Solve the Casida equation: Eq. (12)

No

End

Construct A and B: Eq. (12)

Calculate Σ_{c,qsGW}

Eqs. (17), (18), and (19)

Calculate Σ_{qsMP2}

Eqs. (20), (21), and (16)

FIG. 1. (Color online) Flowchart of qsGW and qsMP2 implementations in MOLGW. Boxes with a dashed line represent computational bottlenecks in this work. Light blue boxes represent major computational bottlenecks in this work. OpenMP- and OpenACC-parallelized equations are highlighted in red.

TABLE I. OpenACC directives and clauses used in this work. N_{kernel} represents the number of kernels in a computational bottleneck. AO-MO represents the atomic orbital-to-molecular orbital integral transformation.

| Bottleneck | Equations | Function/subroutine and file | N_{kernel} | OpenACC directives and clauses |
|------------|-----------|-----------------------------|------------|---------------------------------|
| N/A | Eq. (8) | See footnote | N/A | declare/create/update/device & routine/seq |
| Hartree J | Eq. (7) | setup_hartree(), m_hartree | 1 | data/copyin/copy & m_hartree.f90 parallel/loop/collapse/seq |
| Exchange K | Eq. (10) | setup_exchange(), m_hartree | 1 | data/copyin/copy & m_hartree.f90 parallel/loop/collapse/seq |
| AO-MO | Eq. (16) | calculate_eri_4center_eigen() | 4 | data/copyin/copyout & m_eri_ao_mo.f90 parallel/loop/collapse/seq |
| Σ_{qsGW} excl. AO-MO | Eqs. (17), (18), and (19) | chi_to_vchi() | 1 | data/enter data/exit data/copyin/copyout & linear_response.f90 parallel/loop/collapse |
| Σ_{qsMP2} excl. AO-MO | Eqs. (20) and (21) | pt2_selfenergy_qg() | 1 | data/copyin/copy & pt2_selfenergy_qg.f90 parallel/loop/collapse/seq |

\(^a\) Functions and subroutines are in the first line, and files are in the second line.

\(^b\) For data movement (first line) and computation (second line)

\(^c\) Not available because what OpenACC does in this Fortran module is not parallelizing computational bottlenecks, but transferring global data from CPU memory to GPU memory.

\(^d\) Functions: negligible_basispair(), index_eri(), index_pair(), and eri(). Subroutines: prepare_eri(), setup_shell_index(), setup_basispair(), and identify_negligible_shellpair().

\(^e\) These three are major computational bottlenecks in this work.

\(^f\) CUDA unified (managed) memory can optimize data transfer in this kernel.
and $\mathbf{D}^\sigma$ is the density matrix:

$$D^\sigma_{\mu\nu} = \sum_m f^\sigma_m C^\sigma_{\mu m} C^\sigma_{\nu m},$$

where $f^\sigma_m$ are occupation numbers (0 or 1). The Fock exchange term in Eq. (8) is given by

$$K^\sigma_{\mu\nu} = \sum_{\lambda\tau} D^\sigma_\lambda\tau (\mu\lambda|\tau\nu).$$

It should be noted that the resolution-of-identity (RI) approximation (the density-fitting approximation) significantly reduces the computational and memory costs of the 4-center integrals in Eq. (8) at the cost of a slight accuracy loss. In MOLGW, the RI approximation is parallelized using message passing interface (MPI). In this work, the MPI-parallelized RI approximation in MOLGW is not ported to the GPU, because the RI-MP2 implementation in MOLGW suffers from the MPI communication overhead (the communication time is much greater than the computation time) and fixing the issue is beyond the scope of this work.

C. GW approximation

The GW approximation neglecting vertex corrections (electron-hole interactions) is the first-order expansion of the self-energy in $W$. The interacting (reducible) polarizability $\chi$ is needed to obtain the correlation part of the GW self-energy $\Sigma_{c}^{GW,\sigma}$:

$$\Sigma_{c}^{GW,\sigma} = iG^\sigma (W - v) = iG^\sigma v\chi v,$$

and $\chi$ within the random-phase approximation (RPA) can be obtained by solving the Casida equation:

$$\begin{pmatrix} A & B \\ -B & -A \end{pmatrix} \begin{pmatrix} X^s \\ Y^s \end{pmatrix} = \begin{pmatrix} X^s \\ Y^s \end{pmatrix} \Omega_s,$$

where $A$ and $B$ are the resonant and coupling matrices, respectively, and $(X^s, Y^s)$ and $\Omega_s$ are the eigenvectors and corresponding eigenvalues, respectively. $A$ and $B$ are given by

$$A_{ia\sigma} = (\epsilon^a_i - \epsilon^\sigma_i)\delta_{ij}\delta_{a\sigma}, -(ia\sigma|jba'),$$

$$B_{ia\sigma} = -(ia\sigma|jba'),$$

where $i$ and $j$ are for occupied states, $a$ and $b$ are for empty states, and $(ia\sigma|jba')$ are the 4-orbital two-electron Coulomb repulsion integrals:

$$(ia\sigma|jba') = \iiint d\mathbf{r}d\mathbf{r'}\varphi^*_i(\mathbf{r})\varphi^*_j(\mathbf{r})\frac{1}{|\mathbf{r} - \mathbf{r'}|}\varphi^a_j(\mathbf{r'})\varphi^b_a(\mathbf{r'}),$$

which is obtained from the AO-MO integral transformation:

$$(ia\sigma|jba') = \sum_{\mu\nu\lambda\tau} C^\sigma_{\mu i} C^\sigma_{\mu a} C^\sigma_{\nu a} C^\sigma_{\nu j} (\mu\nu|\lambda\tau).$$

Once $(X^s, Y^s)$ and $\Omega_s$ in Eq. (12) are found, one can obtain the spectral (Lehmann) representation of $\chi(\omega)$, $W(\omega)$, and $\Sigma^{GW,\sigma}_c(\omega)$ successively, as shown in Eq. (11). $\Sigma^{GW,\sigma}_c(\omega)$ is given by

$$\langle \varphi^a_{m} | \Sigma_c^{GW,\sigma}(\omega) | \varphi^b_{n} \rangle = \sum_{is} \frac{w^s_{min} w^s_{nis}}{\omega - \epsilon^a_i + \Omega_s - i\eta} + \sum_{as} \frac{w^s_{max} w^s_{inas}}{\omega - \epsilon^b_a - \Omega_s + i\eta},$$

where $i$ runs over occupied states, $a$ runs over empty states, $s$ runs over all excitations, and $w^s_{mn\sigma}$ are given by

$$w^s_{mn\sigma} = \sum_{ia\sigma} (mn\sigma|ia\sigma')(X^s_{ia\sigma'} + Y^s_{ia\sigma'}).$$

The conventional $G_0W_0$ method uses only diagonal elements $(m = n)$ of Eq. (17) for efficiency, giving only quasiparticle (QP) energies.

In order to remove the starting point dependency in the $G_0W_0$ method, Faleev and co-workers proposed the $q$-GW method using a static and Hermitian approximation to the GW self-energy. The correlation part of the “mode A” $q$-GW self-energy $\Sigma^{qGW,\sigma}_c$ is given by

$$\langle \varphi^a_{m} | \Sigma^{qGW,\sigma}_c | \varphi^b_{n} \rangle = \frac{1}{2} \left[ \langle \varphi^a_{m} | \Sigma^{GW,\sigma}_c (\epsilon^a_{m}) | \varphi^b_{n} \rangle + \langle \varphi^a_{n} | \Sigma^{GW,\sigma}_c (\epsilon^b_{n}) | \varphi^b_{m} \rangle \right],$$

where $\epsilon^a_{m}$ and $\epsilon^b_{n}$ are $q$-GW QP energies and wavefunctions, respectively. Eq. (19) is referred to as the quasiparticle self-consistent approximation in this work. When $\Sigma_{c}^{\sigma} = \Sigma_{c}^{qGW,\sigma}$ in Eq. (8), Eq. (1) becomes the $q$-GW QP equation, which updates both eigenvalues and eigenvectors and thus gives both QP energies and wavefunctions.

It should be noted that for the construction of the $\Sigma^{qGW,\sigma}_c$ matrix in Eq. (19), plane wave-based MBPT codes use truncated basis sets for efficiency, whereas Gaussian-based MOLGW uses all basis sets, because Gaussian basis sets are much more compact than plane wave ones.

D. Second-order Møller–Plesset perturbation theory (MP2)

MP2 is the second-order expansion of the self-energy in $v$ and is the simplest post-HF method. For molecules with weak screening, $v$ is close to $W$, so MP2 and GW methods give similar results for the IE of atoms and molecules.
energy $\Sigma^{\text{MP2}}_{c\sigma}(\omega)$ is given by

$$\langle \varphi'_m | \Sigma_{c\sigma}^{\text{MP2}}(\omega) | \varphi'_n \rangle = \sum_{iap\sigma} \left( m\sigma | a\alpha' \rangle (pm\sigma | a\alpha' \rangle \right) \times \left[ \frac{f_p^\sigma}{\omega + \epsilon_a^\sigma - \epsilon_i^\sigma - \epsilon_p^\sigma - i\eta} + \frac{1 - f_p^\sigma}{\omega + \epsilon_i^\sigma - \epsilon_a^\sigma - \epsilon_p^\sigma + i\eta} \right] - \sum_{iap} \left( m\sigma | a\alpha \rangle (pm\sigma | a\alpha \rangle \right) \times \left[ \frac{f_p^\sigma}{\omega + \epsilon_a^\sigma - \epsilon_i^\sigma - \epsilon_p^\sigma - i\eta} + \frac{1 - f_p^\sigma}{\omega + \epsilon_i^\sigma - \epsilon_a^\sigma - \epsilon_p^\sigma + i\eta} \right],$$

(20)

where $i$ runs over occupied states, $a$ runs over empty states, $p$ runs over both occupied and empty states, and $\epsilon_a^\sigma$ and $\varphi'_m^\sigma$ are HF energies and wavefunctions, respectively. On the right side of Eq. (20), the first summation is the second-order Coulomb (direct) interaction, and the second summation is the second-order exchange (SOX) interaction. Like the $G_0W_0$ method, the one-shot MP2 method starting from HF (MP2@HF) uses only diagonal $\epsilon$ and $\varphi$ elements ($\Sigma$ and MP2@HF) uses only diagonal elements ($m = n$) of Eq. (20), giving only QP energies.

The quasiparticle self-consistent MP2 (qsMP2) method is a MP2 counterpart of the qsGW method. For molecules with strong correlation, qsMP2 gives more stable results than MP2@HF. Applying the quasiparticle self-consistent approximation in Eq. (19) to $\Sigma^{\text{MP2}}_{c\sigma}(\omega)$ in Eq. (20) gives the correlation part of the qsMP2 self-energy $\Sigma^{\text{qsMP2}}_{c\sigma}$:

$$\langle \varphi'_m | \Sigma_{c\sigma}^{\text{qsMP2}}(\epsilon'_m) | \varphi'_n \rangle = \frac{1}{2} \left( \langle \varphi'_m | \Sigma_{c\sigma}^{\text{MP2}}(\epsilon'_m) | \varphi'_n \rangle + \langle \varphi'_n | \Sigma_{c\sigma}^{\text{MP2}}(\epsilon'_n) | \varphi'_m \rangle \right),$$

(21)

where $\epsilon'_m$ and $\varphi'_m$ are qsMP2 QP energies and wavefunctions, respectively. When $\Sigma_{c\sigma}^{\text{qsMP2}} = \Sigma_{c\sigma}^{\text{qsMP2}}$, in Eq. (20), Eq. (4) becomes the qsMP2 QP equation.

It should be noted that the correlation part of the MP2 self-energy in Eq. (20) is different from the MP2 correlation energy. The MP2 self-energy is for excited-state properties, such as electron attachment and detachment energies, whereas the MP2 correlation energy is for ground-state properties, such as total energy. For atoms and molecules, the so-called $\Delta$SCF method based on the MP2 total energy gives similar vertical IEs to the MP2 self-energy method.

### E. Computational costs of qsGW and qsMP2

In terms of compute time, qsGW and qsMP2 methods are computationally much more demanding than one-shot $G_0W_0$ and MP2@HF methods because of a couple of reasons. First, one-shot $G_0W_0$ and MP2@HF methods need no SCF iteration, but qsGW and qsMP2 methods require dozens of SCF iterations. Second, while one-shot $G_0W_0$ and MP2@HF methods need to calculate only diagonal ($m = n$) elements of the $\langle \varphi'_m | \Sigma_{c\sigma}(\omega) | \varphi'_n \rangle$ matrix in Eqs. (17) and (20), qsGW and qsMP2 methods have to calculate all elements. Last, whereas one-shot $G_0W_0$ and MP2@HF methods need to solve QP equations for a couple of MOs of interest, such as frontier MOs, qsGW and qsMP2 methods have to solve QP equations for all MOs.

However, qsGW and qsMP2 methods use almost the same amount of memory as one-shot $G_0W_0$ and MP2@HF methods, and thus are ideal for GPU acceleration. Memory-hungry electronic-structure methods that demand a large amount of memory are not suitable for GPU computing, because they cannot run on the GPU with a small memory.

### F. Computational bottlenecks

Figure 1 depicts a flowchart of qsGW and qsMP2 implementations in MOLGW. Figure 4 shows the computational bottlenecks, such as $J$ in Eq. (7), $K$ in Eq. (10), the AO-MO integral transformation in Eq. (16), $\Sigma^{\text{qsGW}}$ in Eqs. (17), (18), and (19), and $\Sigma^{\text{qsMP2}}$ in Eqs. (20) and (21). For the AO-MO integral transformation in Eq. (16), the compute time scales as $O(N^5)$ with $N$ being the system size, because MOLGW uses a traditional transformation that reduces the $O(N^4)$ algorithm to $O(N^5)$. The AO-MO integral transformation is a major computational bottleneck common in qsGW and qsMP2 calculations, as shown in Eqs. (16), (17), (18), and (20). For the Casida matrix in Eqs. (12), (13), and (14), the construction and diagonalization time and memory scale as $O(N^3)$ and $O(N^4)$, respectively, and the diagonalization time scales as $O(N^6)$. In this work, the construction and diagonalization of the Casida matrix are minor computational bottlenecks, because small Casida matrices are used.

### III. IMPLEMENTATION DETAILS

#### A. OpenACC implementation

OpenMP and OpenACC are multi-platform shared-memory (thread-based) parallel programming models, which can target both CPUs and GPUs and are free from the inter-process communication overhead in distributed-memory (process-based) models such as MPI, as discussed in Section II B. Although OpenMP supports GPU offloading, we used OpenMP for CPUs and OpenACC for GPUs in this work, because for now OpenACC is a more mature model for GPU offloading than OpenMP.

Table I summarizes OpenACC directives and clauses used in this work and presents where and why they are
Listing 1. Simplified Fortran source code of the OpenMP- and OpenACC-parallelized subroutine for the calculation of the correlation part of the $\Sigma_{qsMP2}^c$.

Listing 1 presents an example of OpenMP (CPU) and OpenACC (GPU) implementations in MOLGW. Listing 1 shows that the calculation of $\Sigma_{qsMP2}^c$ consists of two computational bottlenecks: the AO-MO integral transformation (a line 19) and $\Sigma_{qsMP2}^c$ excluding the transformation (lines 31–35) (see Table I). This is the case for the
B. Similarities between OpenMP and OpenACC implementations

OpenMP (CPU) and OpenACC (GPU) implementations in MOLGW are similar in a few ways\textsuperscript{56}. First, both annotate the original source code with pragmas without data layout restructuring\textsuperscript{58}, allowing us to preserve the loop order in a serial CPU version, as shown at lines 14, 15, 31, and 32 in Listing 1. Second, both require code refactoring\textsuperscript{61}. For example, we fixed a race condition, which we encountered when parallelizing the implementation of $\Sigma_c^{\text{qsGW}}$. Also, we changed the index type of some arrays from a 4-byte integer to an 8-byte integer to enable MOLGW to use a large amount of memory. Also, we implemented our own timing routines into MOLGW (for Section V.B), because built-in timing routines in MOLGW give wrong results for long calculations.

Third, both parallelize the computational bottlenecks using similar compiler directives. For example, Listing 1 shows that OpenMP and OpenACC parallelized a loop using `omp parallel do` (lines 24, 25, 41, and 42) and `acc parallel loop` (lines 28 and 38) directives, respectively.

Last, both collapse (merge or unroll) nested loops into a single loop to increase parallelism. For example, Listing 1 shows that OpenMP and OpenACC merged `pstate` and `qstate` loops (lines 31, 32, 34, and 35) using the `collapse` clause (lines 25 and 28, respectively).

C. Differences between OpenMP and OpenACC implementations

However, there are a few differences between OpenMP (CPU) and OpenACC (GPU) implementations in MOLGW. First, unlike OpenMP, OpenACC decouples data movement from computation, as shown at lines 11 and 49 in Listing 1 requiring to optimize the data transfer between host (CPU) and device (GPU) memories after offloading the computation to the GPU. We used CUDA unified (managed) memory, a single memory space for both host and device memories\textsuperscript{62}, to manage dynamically allocated arrays used in the implementation of $\Sigma_c^{\text{qsGW}}$ (see Table I). We used a manual deep copy to handle dynamic data structures, as is done for the OpenACC version of VASP\textsuperscript{36} because OpenACC 2.6 does not support a true deep copy (NVIDIA compilers used in this work support the OpenACC 2.6 specification). We transferred global data from CPU memory to GPU memory after the completion of the calculation of the 4-center two-electron Coulomb repulsion integrals in Eq. (6) (see Table I). We used NVIDIA profilers to confirm that we fully optimized the CPU–GPU communication overhead in our OpenACC implementation.

Second, unlike the OpenMP (CPU) implementation, our OpenACC (GPU) implementation does not parallelize the 4-center integrals in Eq. (8), because MOLGW uses Libint, a single instruction multiple data (SIMD)-vectorized CPU library for computing Gaussian integrals\textsuperscript{63}. The 4-center integrals are calculated only once at the beginning of MOLGW execution, and thus are a major computational bottleneck in non-self-consistent $G_0W_0$ and MP2@HF calculations, but a minor computational bottleneck in quasiparticle self-consistent GW and MP2 calculations (see Section V.E).

Last, the OpenMP (CPU) implementation diagonalizes matrices, such as HF, $\text{qsGW}$, and $\text{qsMP2}$ matrices in Eqs. (4) and (6) with $\Sigma_c^\alpha = 0$, $\Sigma_c^{\text{qsGW}}$, and $\Sigma_c^{\text{qsMP2}}$, respectively, and the Casida matrix in Eq. (12) (see Fig. 1), using OpenMP-parallelized Intel math kernel library (MKL), but our OpenACC (GPU) implementation performs the matrix diagonalization serially. In this work, the matrix diagonalization is a minor computational bottleneck, because small matrices are used, as discussed in Section V.E.

D. Regression testing
In order to ensure that our parallel GPU version of MOLGW gives the same results as the original serial CPU version, we performed two kinds of regression testing. First, we used the built-in automated regression test suite in MOLGW and verified that our GPU implementation produces correct results. Second, we compared QP energies obtained from CPU and GPU calculations. Figure 2 depicts HF and qsGW energies for the highest occupied molecular orbital (HOMO) of a butane molecule (C₄H₁₀) from serial CPU, parallel OpenMP CPU, and parallel OpenACC GPU calculations during the SCF iteration, which are presented in Table II in Supporting Information. Figure 2 shows that parallel OpenMP CPU and OpenACC GPU calculations give identical results to the serial CPU calculation, as is the case for qsMP2 as well (see Fig. 1 in Supporting Information). Table II in Supporting Information shows 5 and 10 matching significant digits for HOMO and total energies, respectively, during 10 SCF iterations.

IV. BENCHMARK CONFIGURATIONS

In order to assess the performance of our OpenACC implementation in MOLGW for GPU computing, we conducted a few benchmarks using various software and hardware configurations. We used the following software configurations: First, we used two different starting-point-independent MBPT methods, qsGW and qsMP2, which are computationally the most demanding electronic structure methods in MOLGW. Second, we used the Dunning’s correlation-consistent basis set, cc-pVTZ. Third, we used the first seven linear alkanes (CₙH₂ₙ₊₂, where n = 1, 2, ..., 7) to systematically increase the molecular size and the GPU memory usage. We did not use alkanes larger than heptane (C₇H₁₆) due to our GPU memory limit of 24 GB. To calculate large alkanes, one should reduce the memory usage by using small basis sets, such as cc-pVDZ and Pople’s basis sets, at the cost of accuracy loss or by implementing molecular symmetry into MOLGW. Fourth, for the performance analysis using compute times, we used total of 8 SCF iterations, consisting of first 5 HF iterations [Σₓ = 0 in Eq. (6)] and subsequent 3 qsGW or qsMP2 iterations [Σₓ = ΣₓqsGW or ΣₓqsMP2, respectively, in Eq. (6)] (see Fig. 1). It should be noted that dozens of qsGW or qsMP2 iterations are needed to reach the convergence. Fifth, for the roofline performance analysis, we used total of 6 SCF iterations (1 qsGW or qsMP2 iteration), the Intel Advisor (OpenMP), and the NVIDIA Nsight Compute (OpenACC). Last, we used either Intel or NVIDIA compilers (formerly known as PGI compilers) for OpenMP, but only NVIDIA compilers for OpenACC.

We used the following hardware configurations: For OpenACC, we used two different GPUs, NVIDIA GeForce RTX 3090 and 4090 for a desktop computer (RTX 3090 and RTX 4090, respectively, in the following), whose specifications are summarized in Table II. The RTX 4090 has a newer architecture and higher FP64 performance than the RTX 3090. For OpenMP, we used two different CPUs, AMD Ryzen Threadripper PRO 3975WX 32-core CPU with SMT disabled are used. GPUS are not used. The black dotted line represents the ideal speedup. Data is taken from Table III in Supporting Information.

V. RESULTS AND DISCUSSION

A. Performance analysis using total timings

In order to obtain the GPU speedups over CPU, we performed an OpenMP (CPU) and OpenACC (GPU) benchmark using total timings and summarized the benchmark results in Table III. We see a possibility that GPUs can accelerate large-scale quasiparticle self-consistent MBPT calculations. For example, the qsGW/cc-pVTZ calculation of the heptane molecule (C₇H₁₆) on the Threadripper and Xeon Phi GPUs using only 3 qsGW iterations takes 15.69 and 23.71 hours of compute time, respectively. The RTX 4090 GPU reduces the compute time to 4.98 hours (by 3.2 and 4.8 times, respectively). As noted in Section IV about 10 times more iterations and compute times are needed for real calculations. For example, the above calculation on the Xeon Phi CPU using 30 iterations takes about...
### TABLE II. Specifications for GPUs and CPUs used in this work.

|                  | GPU          |                | CPU          |                |
|------------------|--------------|----------------|--------------|----------------|
|                  | RTX 3090 †   | RTX 4090 ‡  | Xeon Phi †   | Threadripper †|
| Platform         | Desktop      | Desktop        | Supercomputer| Workstation    |
| Release year     | 2020         | 2022           | 2016         | 2020           |
| Architecture     | Ampere       | Ada Lovelace   | Knights Landing (KNL) | Zen 2 (Castle Peak) |
| Manufacturing process | 8 nm        | 5 nm           | 14 nm        | 7 nm           |
| Number of cores  | 10494 †      | 16384 ‡       | 68           | 32             |
| Boost clock      | 1.70 GHz     | 2.52 GHz       | 1.60 GHz     | 4.20 GHz       |
| Base clock       | 1.40 GHz     | 2.23 GHz       | 1.40 GHz     | 3.50 GHz       |
| Memory size      | 24 GB        | 24 GB          | 96 GB        | 128 GB         |
| Memory type      | GDDR6X       | GDDR6X         | DDR4         | DDR4           |
| Memory bandwidth | 935.8 GB/s   | 1008 GB/s      | 115.2 GB/s   | 102.4 GB/s     |
| FP64             | 556 GFLOP/s  | 1290 GFLOP/s   | N/A          | 268 GFLOP/s    |

- † NVIDIA GeForce RTX 3090
- ‡ NVIDIA GeForce RTX 4090
- Intel Xeon Phi 7250
- AMD Ryzen Threadripper PRO 3975WX
- Number of compute unified device architecture (CUDA) cores
- Maximum memory bandwidth for the quad-channel configuration used in this work. The Intel Advisor gives a measured value of 61.2 GB/s.
- Not available. We can no longer access the Xeon Phi to measure its FP64 using the Intel Advisor, because our supercomputing time grant at the KISTI National Supercomputing Center ended.
- Measured by the Intel Advisor

---

**FIG. 4.** (Color online) OpenMP and OpenACC speedups of qsGW and qsMP2 methods (left and right, respectively). GB represents gigabyte. Data is taken from Table III.

---

30 days, which can exceed the walltime limit (typically, about 2 days) in HPC supercomputing centers. In the following, we will visualize and analyze our benchmark results in Table III to find trends.

We begin by briefly checking the OpenMP (CPU) parallel efficiency. Figure 3 shows the OpenMP speedup as a function of number of CPU threads using different MBPT methods. We used the propane molecule ($C_3H_8$) and the Threadripper CPU. We see that the qsMP2 method gives a higher parallel efficiency than the qsGW one ($\sim 0.66$ and $\sim 0.53$, respectively). This is because the implementation of $\Sigma_{MP2}$ in Eq. (20) takes less steps and thus is simpler than that of $\Sigma_{GW}$ in Eqs. (12), (13), (14), (17), and (18), as shown in Fig. 1. We also see that for both qsGW and qsMP2 methods, the OpenMP parallel efficiency is almost independent of the number of CPU threads (the OpenMP speedup scales nearly linearly with the number of CPU threads).

Next, we compare OpenMP (CPU) and OpenACC (GPU) performances using a single alkane molecule. Fig-
TABLE III. OpenMP and OpenACC compute times using different alkane molecules (C$_n$H$_{2n+2}$), MBPT methods, CPUs, and GPUs. GB represents gigabyte.

| Molecule | Method | NBF | Memory (GB) | Threadripper (hours) | Xeon Phi (hours) | OpenACC (hours) | Speedup |
|----------|--------|-----|-------------|----------------------|------------------|-----------------|---------|
|          |        |     |             | 1 thread | 32 threads | 68 threads | RTX 3090 | RTX 4090 | RTX 4090 |
| CH$_4$   | qsGW   | 86  | 0.1         | 0.017    | 0.0014  | 0.0047    | 0.0061  | 0.0031  | 0.5x    |
| C$_2$H$_6$ | qsGW   | 144 | 0.8         | 0.033    | 0.018   | 0.038     | 0.043   | 0.018   | 1.0x    |
| C$_3$H$_8$ | qsGW   | 202 | 2.8         | 2.47     | 0.15    | 0.27      | 0.17    | 0.077   | 1.9x    |
| C$_4$H$_{10}$ | qsGW | 260 | 5.0         | 11.38    | 0.65    | 1.00      | 0.55    | 0.29    | 2.2x    |
| C$_5$H$_{12}$ | qsGW | 318 | 9.5         | N/A      | 2.65    | 3.31      | 1.39    | 0.87    | 3.0x    |
| C$_6$H$_{14}$ | qsGW | 376 | 14.9        | N/A      | 6.93    | 9.44      | 3.03    | 2.17    | 3.2x    |
| C$_7$H$_{16}$ | qsGW | 434 | 21.6        | N/A      | 15.69   | 23.71     | 6.80    | 4.98    | 3.2x    |
| CH$_4$   | qsMP2  | 86  | 0.1         | 0.11     | 0.0056  | 0.012     | 0.0078  | 0.0039  | 1.4x    |
| C$_2$H$_6$ | qsMP2  | 144 | 0.5         | 1.69     | 0.081   | 0.16      | 0.056   | 0.029   | 2.8x    |
| C$_3$H$_8$ | qsMP2  | 202 | 1.8         | 9.42     | 0.44    | 0.92      | 0.26    | 0.11    | 4.0x    |
| C$_4$H$_{10}$ | qsMP2 | 260 | 4.4         | 31.48    | 1.69    | 3.46      | 0.82    | 0.34    | 5.0x    |
| C$_5$H$_{12}$ | qsMP2 | 318 | 8.6         | N/A      | 6.06    | 9.09      | 1.94    | 0.81    | 7.5x    |
| C$_6$H$_{14}$ | qsMP2 | 376 | 14.5        | N/A      | 14.96   | 21.94     | 4.12    | 1.68    | 8.9x    |
| C$_7$H$_{16}$ | qsMP2 | 434 | 21.4        | N/A      | 29.88   | 45.40     | 7.89    | 3.08    | 9.7x    |

* AMD Ryzen Threadripper PRO 3975WX 32-core CPU  
* Intel Xeon Phi 7250 68-core CPU  
* Simultaneous multithreading (SMT) is disabled.  
* Single-thread calculations are not performed on the Xeon Phi CPU, which is made up of lightweight cores with a base clock of 1.40 GHz, as shown in Table II.  
* Only 3 qsGW and qsMP2 iterations are used for benchmark purposes. For real calculations, one should use about 10 times more iterations, leading to about 10 times larger OpenMP and OpenACC compute times.  
* NBF represents the number of basis functions. The cc-pVTZ basis set is used.  
* GPU peak memory usage  
* NVIDIA GeForce RTX 3090 GPU on a desktop computer with the AMD Ryzen 9 3950X 16-core CPU  
* NVIDIA GeForce RTX 4090 GPU on a desktop computer with the AMD Ryzen 9 5950X 16-core CPU  
* NVIDIA compilers (formerly known as PGI compilers) are used.  
* Intel compilers are used.  
* OpenMP time from the Threadripper CPU using 32 threads over OpenACC time from the RTX 4090 GPU  
* OpenMP time from the Xeon Phi CPU using 68 threads over OpenACC time from the RTX 4090 GPU  

Section V B. shows OpenMP and OpenACC speedups of different MBPT methods. We used the propane molecule (C$_3$H$_8$), the Threadripper CPU, and the RTX 4090 GPU. We see a couple of trends. First, for both qsGW and qsMP2 methods, OpenACC achieves higher speedups than 32 OpenMP threads (by 1.9x and 4.0x, respectively), which shows that GPUs can perform high-precision MBPT calculations of molecules faster than CPUs. Second, for both OpenMP and OpenACC, the qsMP2 method gets higher speedups than the qsGW one (by 1.3x and 2.7x, respectively), which shows that the performance boost from parallelization depends on the kind of MBPT methods.

Finally, we analyze the effect of the molecule size on the OpenACC (GPU) performance. Figure 6 shows the OpenACC GPU speedup with respect to OpenMP CPU threads as a function of alkane (C$_n$H$_{2n+2}$) size using different MBPT methods, GPUs, and CPUs. We see a few trends. First, the OpenACC speedup depends on the kind of MBPT methods: for all alkane sizes and GPUs, the qsMP2 method gets higher OpenACC speedups than the qsGW one. This trend will be discussed in detail in Section V B. Second, the OpenACC speedup increases with the alkane size (the basis size or the GPU memory footprint) for all MBPT methods and GPUs, larger alkane molecules get higher OpenACC speedups, because large arrays utilize more CUDA cores than small arrays. For example, qsGW calculations of butane and heptane molecules (C$_4$H$_{10}$ and C$_7$H$_{16}$, respectively) on the RTX 4090 GPU use 5.0 and 21.6 GB of GPU memory, respectively, and get OpenACC speedups of 2.2x and 3.2x, respectively, compared to those on the Threadripper CPU, as shown in Table III. Also, small-scale MBPT calculations using small basis sets might not benefit from GPU acceleration, because they do not have a sufficient number of threads to saturate CUDA cores. For example, the qsGW calculation of the methane molecule (CH$_4$),...
We analyzed the asymptotic time complexity of the OpenACC speedup with respect to alkane \( (C_nH_{2n+2}) \) size using a fitting function:

\[
    f(n) = \alpha \times n^\beta,
\]

where \( \alpha \) and \( \beta \) are fitting constants, and summarized the analysis results in Table VI in Supporting Information. For example, we obtained \( f(n) = 1.30 \times n^{1.05} \) for the \( \text{qsMP2} \) method on the RTX 4090 GPU and the Threadripper CPU, as shown in Fig. 5. The scaling factor \( \beta \) ranges from 0.56 (the \( \text{qsGW} \) method on the RTX 4090 GPU and the Xeon Phi CPU) to 1.21 (the \( \text{qsGW} \) method on the RTX 3090 GPU and the Threadripper CPU).

Third, the OpenACC speedup depends on the kind of CPUs: for all MBPT methods and alkane sizes, the Xeon Phi CPU underperforms the Threadripper CPU, taking longer OpenMP compute times and thus giving higher OpenACC speedups. For example, for the \( \text{qsMP2} \) calculation of \( C_4H_{10} \), the Xeon Phi CPU gives a 2.0x higher OpenACC speedup than the Threadripper CPU, as shown in Table III.

Last, the OpenACC speedup increases with the GPU size for the smallest alkane, on the RTX 4090 GPU uses only 86 basis functions and gets an OpenACC speedup of 0.5x relative to that on the Threadripper CPU (the calculation runs faster on the CPU than on the GPU), as shown in Table III. This trend will be discussed in detail in Section V C.
TABLE IV. Dynamic random access memory (DRAM) arithmetic intensity (AI) and double-precision floating-point performance (FP64) of major computational bottlenecks in qsGW and qsMP2 calculations of alkane molecules \((C_nH_{2n+2})\), obtained from the NVIDIA Nsight Compute. (G)FLOP represents (giga) floating-point operations. AO-MO represents the atomic orbital-to-molecular orbital integral transformation. \(\Sigma_{qsGW}^c\) and \(\Sigma_{qsMP2}^c\) represent the correlation part of qsGW and qsMP2 self-energies, respectively.

| Molecule | AO-MO | 1st kernel | 2nd kernel | 3rd kernel | 4th kernel | \(\Sigma_{qsGW}^c\) excl. AO-MO | \(\Sigma_{qsMP2}^c\) excl. AO-MO |
|----------|-------|------------|------------|------------|------------|----------------|-----------------|
|          | AI FP64 | AI FP64 | AI FP64 | AI FP64 | AI FP64 | AI FP64 | AI FP64 |
| CH\(_4\) | 1.78 22.45 | 35.76 135.3 | 36.84 120.2 | 36.59 129.8 | 0.25 12.36 | 4230 317.5 |
| C\(_2\)H\(_6\) | 0.18 32.24 | 51.44 162.5 | 50.56 147.6 | 50.99 162.1 | 0.13 28.28 | 9840 466.6 |
| C\(_3\)H\(_8\) | 0.13 33.44 | 42.04 121.0 | 41.34 148.0 | 32.91 144.7 | 0.07 12.54 | 13714 613.0 |
| C\(_4\)H\(_10\) | 0.11 31.51 | 44.59 134.8 | 44.40 211.3 | 0.70 103.1 | 6.56 21569 651.8 |
| C\(_5\)H\(_12\) | 0.12 33.11 | 49.41 237.4 | 49.89 239.2 | 0.32 70.3 | 24942 705.3 |
| C\(_6\)H\(_14\) | 0.14 37.42 | 56.69 319.5 | 56.29 243.2 | 0.32 70.3 | 24942 705.3 |
| C\(_7\)H\(_16\) | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

\(\text{a}^{*}\) See Table III for the number of basis functions and the GPU peak memory usage

\(\text{b}^{*}\) Not available because profiling calculations are prohibitively expensive. Profiling calculations take more than 10 times longer than normal calculations.

TABLE V. GPU utilization of major computational bottlenecks in qsGW and qsMP2 calculations of alkane molecules \((C_nH_{2n+2})\), obtained from the NVIDIA Nsight Compute. AO-MO represents the atomic orbital-to-molecular orbital integral transformation. \(\Sigma_{qsGW}^c\) and \(\Sigma_{qsMP2}^c\) represent the correlation part of qsGW and qsMP2 self-energies, respectively.

| Molecule | AO-MO | 1st kernel | 2nd kernel | 3rd kernel | 4th kernel | \(\Sigma_{qsGW}^c\) excl. AO-MO | \(\Sigma_{qsMP2}^c\) excl. AO-MO |
|----------|-------|------------|------------|------------|------------|----------------|-----------------|
|          | Memory Compute | Memory Compute | Memory Compute | Memory Compute | Memory Compute | Memory Compute |
| CH\(_4\) | 19.2 7.3 | 33.0 81.1 | 28.7 69.8 | 31.4 76.4 | 33.1 76.4 | 1.6 1.4 |
| C\(_2\)H\(_6\) | 29.8 10.5 | 30.9 79.4 | 38.9 66.2 | 62.6 72.9 | 75.9 2.0 | 25.9 |
| C\(_3\)H\(_8\) | 35.6 10.8 | 31.0 79.8 | 28.9 69.3 | 74.4 53.3 | 33.6 73.4 |
| C\(_4\)H\(_10\) | 37.2 9.9 | 30.4 81.2 | 64.0 67.6 | 57.1 33.0 | 19.8 72.4 |
| C\(_5\)H\(_12\) | 41.1 10.1 | 30.5 81.4 | 70.7 66.6 | 48.7 22.5 | 20.0 77.9 |
| C\(_6\)H\(_14\) | 42.8 11.1 | 30.5 81.4 | 93.5 61.9 | 42.1 17.9 | 20.1 84.3 |
| C\(_7\)H\(_16\) | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

\(\text{a}^{*}\) See Table III for the number of basis functions and the GPU peak memory usage

\(\text{b}^{*}\) Not available because profiling calculations are prohibitively expensive. Profiling calculations take more than 10 times longer than normal calculations.

generation: for all MBPT methods and alkane sizes, the RTX 4090 GPU takes shorter OpenACC compute times and thus gives higher OpenACC speedups than the RTX 3090 GPU (see Table III for their specifications). For example, for qsGW and qsMP2 calculations of C\(_4\)H\(_10\), the RTX 4090 GPU gives higher OpenACC speedups than the RTX 3090 GPU by 1.9 and 2.4 times, respectively, as shown in Table III, which are close to the ratio of the RTX 4090 FP64 performance to the RTX 3090 one (1290 GFLOP/s ÷ 556 GFLOP/s ÷ 2.3), as shown in Table III.

Overall, the qsMP2 calculation of C\(_7\)H\(_16\) on the RTX 4090 GPU using 21.4 GB of GPU memory achieves the highest OpenACC speedup of 9.7x and 14.7x with respect to that on the Threadripper and Xeon Phi CPUs, respectively, as shown in Table III. Based on the trends above, we expect that larger-memory-footprint MBPT calculations on high-end GPUs, such as the NVIDIA H100 with a high FP64 performance and 80 GB of an extension of the second generation of high bandwidth memory (HBM2e), would achieve higher OpenACC speedups than 9.7x and 14.7x.
B. Performance analysis using partial timings

In order to find why the qsMP2 method gets higher OpenACC speedups than the qsGW method, as seen in Section VA, we performed an OpenMP (CPU) and OpenACC (GPU) benchmark using partial timings and summarized the benchmark results in Tables IV and V in Supporting Information. As explained in Section III A the \( \Sigma_{\text{qsMP2}} \) calculation is composed of two computational bottlenecks: the AO-MO integral transformation (AO-MO in the following) and the AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) (see Table I and Listing I), which is the case for the \( \Sigma_{\text{qsGW}} \) calculation as well. Figure 6 shows decomposed OpenMP (CPU) and OpenACC (GPU) compute times of \( \Sigma_{\text{qsGW}} \) and \( \Sigma_{\text{qsMP2}} \) calculations of the pentane molecule (C\(_5\)H\(_{12}\)) on the Threadripper CPU and the RTX 4090 GPU. We see that in the case of OpenMP (CPU) calculations, the AO-MO bottleneck does not dominate the \( \Sigma_{\text{qsGW}} \) calculation, but does the \( \Sigma_{\text{qsMP2}} \) calculation. For example, the AO-MO bottleneck takes 4.7 times less OpenMP compute time than the AO-MO-excluded \( \Sigma_{\text{qsGW}} \) bottleneck in the \( \Sigma_{\text{qsGW}} \) calculation, but 11.8 times more OpenMP compute time than the AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) bottleneck in the \( \Sigma_{\text{qsMP2}} \) calculation (see Table IV in Supporting Information). We also see that the AO-MO bottleneck benefits the most from GPU acceleration among three major computational bottlenecks. For example, OpenACC (GPU) reduces OpenMP (CPU) compute times of AO-MO, AO-MO-excluded \( \Sigma_{\text{qsGW}} \), and AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) bottlenecks by 94%, 71%, and 46%. As a result, GPU acceleration has a stronger effect on the qsMP2 method than the qsGW method (GPU speedups of 8.9x and 3.2x, respectively, as shown in Table III).

C. Roofline performance analysis

In order to find what is limiting the performance of our OpenACC implementation, we conducted the roofline performance analysis\(^\text{\ref{footnote1}}\) of three major computational bottlenecks (a total of 6 kernels) – AO-MO (consisting of 4 kernels), AO-MO-excluded \( \Sigma_{\text{qsGW}} \), and AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) (see Table I) – in OpenACC calculations on the RTX 4090 GPU, and summarized the analysis results in Table IV. Figure 7 depicts the roofline analysis results for the hexane molecule (C\(_6\)H\(_{14}\)). We see that the AO-MO bottleneck, which is common in qsGW and qsMP2 calculations, has both compute- and memory-bound kernels. For example, the first and fourth kernels in the AO-MO bottleneck are under the peak bandwidth ceiling (diagonal line) and thus are memory-bound, while the second and third kernels are under the peak performance ceiling (horizontal line) and thus are compute-bound. We also see that the AO-MO-excluded \( \Sigma_{\text{qsGW}} \) bottleneck is highly memory-bound (arithmetic intensity of 0.03 FLOP/byte), whereas the AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) bottleneck is highly compute-bound (FP64 performance of 705.3 GFLOP/sec) (see Table IV). This shows that the qsGW method is more memory-bound and less compute-bound than the qsMP2 method. We summarized the roofline analysis results for OpenMP calculations on the Threadripper CPU in Table VII in Supporting Information, and depicted the results for C\(_6\)H\(_{14}\) in Fig. 2 in Supporting Information.  

In order to understand the optimization level of our OpenACC implementation, we analyzed the GPU utilization of three major computational bottlenecks (a total of 6 kernels) in OpenACC calculations on the RTX 4090 GPU with the FP64 performance of 1290 GFLOP/s and the memory bandwidth of 1008 GB/s (see Table I), and summarized the analysis results in Table V. We see that the GPU utilization depends on the molecule size. For example, the GPU compute utilization of the AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) bottleneck increases with the alkane size from 38.2% to 84.3% (CH\(_4\) and C\(_7\)H\(_{16}\), respectively), while that of the second kernel in the AO-MO bottleneck remains nearly constant at ~80% across all alkane sizes. Also, the GPU memory utilization of the first kernel in the AO-MO bottleneck increases with the molecule size from 19.2% to 42.8% (CH\(_4\) and C\(_7\)H\(_{16}\), respectively), whereas that of the second kernel remains almost unchanged at ~30% across all alkane sizes. We also see that the GPU utilization depends on the kind of kernels. For example, in the case of the hexane molecule (C\(_6\)H\(_{14}\)), the GPU compute utilization ranges from 0.6% to 84.3% (AO-MO-excluded \( \Sigma_{\text{qsGW}} \) and \( \Sigma_{\text{qsMP2}} \) kernels, respectively), and the GPU memory utilization ranges from 4.4% to 93.5% (the AO-MO-excluded \( \Sigma_{\text{qsMP2}} \) kernel and the third kernel in the AO-MO bottleneck, respectively). We summarized the analysis results for the CPU...
utilization of three major computational bottlenecks in OpenMP calculations on the Threadripper CPU with the measured FP64 performance of 268 GFLOP/s and the measured quad-channel memory bandwidth of 61.2 GB/s (see Table II) in Table VIII in Supporting Information.

D. Discussion

We have a few points to discuss. First, OpenMP and OpenACC implementations in MOLGW “added parallelism into existing source code without significantly modifying it,” enabling us to maintain a single version of source code for both CPUs and GPUs. The clean and maintainable MOLGW source code makes it possible for domain scientists, such as electronic structure method developers, with no or little parallel programming background to do “more science and less programming.”

Second, our current OpenACC version of MOLGW runs only on a single GPU. We will enable MOLGW to run on a multi-GPU node using the hybrid OpenMP/OpenACC parallelization in the future. The multi-GPU implementation can enhance scalability of MOLGW but will reduce readability, understandability, and backward compatibility of source code, because it requires to change the serial CPU code.

Third, the matrix diagonalization in MOLGW is not ported to the GPU, as discussed in Section [IIIC]. Although the diagonalization of the Casida matrix scales as \(O(N^6)\), it is a minor computational bottleneck in this work due to a small prefactor, as discussed in Section [IIIF]. However, it will become a major computational bottleneck in \(G_0W_0\) and \(qsGW\) calculations of very large systems. Hence, we will port the matrix diagonalization in MOLGW to the GPU in the future possibly using a GPU-accelerated library for linear algebra.

Last, not all MBPT methods and molecular systems would benefit from GPU acceleration. For example, computationally cheap one-shot \(G_0W_0\) and MP2@HF methods should attain lower GPU speedups than computationally expensive \(qsGW\) and \(qsMP2\) ones, as discussed in Section [IIIE]. Also, MBPT calculations of small systems could run faster on the CPU than on the GPU, as seen in Section [VIB] and those of large systems might require more memory than available on the GPU (typically, the CPU has more memory than the GPU, as shown in Table [I]). Overall, the GPU version of a MBPT code cannot fully replace the CPU version, because CPUs are still needed for certain kinds of MBPT methods and molecular systems.

VI. SUMMARY AND CONCLUSIONS

In summary, we have ported MOLGW to the GPU using OpenACC without sacrificing accuracy, and evaluated the performance of GPU-accelerated MOLGW using different starting-point-independent MBPT methods, system sizes, and GPUs. We found that the GPU-accelerated version of MOLGW can run faster than the CPU version using 32 OpenMP threads by up to 9.7 times, and the speedup depends on the kind of MBPT methods and increases with the GPU generation and the system size (the basis size or the GPU memory footprint). We identified both compute- and memory-bound kernels in GPU-accelerated MOLGW using the roofline performance analysis. Our choice of OpenACC over CUDA allows us to maintain a single version of the MOLGW source code for both CPUs and GPUs, which significantly reduces programming and maintenance efforts and enhances code and performance portability. Our GPU acceleration of quasiparticle self-consistent MBPT methods in MOLGW paves the way for the application of \(\textit{ab initio}\) MBPT calculations without empirical parameters, such as starting points, to complex real systems.

VII. AUTHORSHIP CONTRIBUTION STATEMENT

Young-Moo Byun conceived and designed the project, implemented and validated the computer code, performed the computations, collected the data, analyzed and interpreted the results, and wrote the draft manuscript. Jejoong Yoo supervised the project, provided the computing resources, and reviewed and revised the manuscript. All authors reviewed the results and approved the final version of the manuscript.

VIII. DATA AVAILABILITY STATEMENT

Basis sets and the MOLGW source code with our OpenACC implementation for a single GPU are publicly available via GitHub at https://github.com/ymbyun/molgw-1.F-openmp-openacc-single-gpu.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1A2C1101424). This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-02068, Artificial Intelligence Innovation Hub). This work was supported by the National Supercomputing Center with supercomputing resources including technical support (KSC-2021-CRE-0212).

\[ G. \quad \text{Onida, L. \quad Reining, \quad and \quad A. \quad Rubio. \quad Rev. \quad Mod. \quad Phys. \quad 74, \quad 601 \quad (2002)} \]
208, 149 (2016)
209. D. Golze, M. Dvorak, and P. Rinke, Frontiers in Chemistry 7, 377 (2019)
210. F. Bruneval, N. Dattani, and M. J. van Setten, Frontiers in Chemistry 9 (2021), 10.3389/fchem.2021.749779
211. F. Bruneval, N. Vast, and L. Reining, Phys. Rev. B 74, 045102 (2006)
212. F. Bruneval, The Journal of Chemical Physics 136, 194107 (2012) https://doi.org/10.1063/1.4718428
213. CrawfordGroup, “C++ programming tutorial in chemistry,” https://github.com/CrawfordGroup/ProgrammingProjects/, accessed November 19, 2023.
214. Openmp homepage,” https://www.openmp.org/(), accessed June 8, 2023.
215. B. Q. Pham, M. Allan, and M. S. Gordon, Journal of Chemical Theory and Computation 19, 2213 (2023), pMID: 37011288, https://doi.org/10.1021/acs.jctc.2c01137.
216. Y.-M. Byun, “Molgw 1.1 with openmp and openacc (single gpu),” https://github.com/ymbyun/molgw-1.
217. F-openmp-openacc-single-gpu, accessed June 8, 2023.
218. C. Bonati, E. Calore, S. Coscetti, M. D’Elia, M. Mesiti, F. Negro, S. F. Schifano, and R. Tripiccione, in Proceedings of the 2015 International Workshop on Software Engineering for High Performance Computing in Science, SE4HPCS ‘15 (IEEE Press, 2015) p. 9–15.
219. “Code refactoring,” https://en.wikipedia.org/wiki/Code_refactoring, accessed November 28, 2023.
220. M. Wolfe, “Openacc and cuda unified memory,” https://www.pgroup.com/blogs/posts/openacc-unified-memory.htm, accessed June 8, 2023.
221. E. F. Valeev, “Libint: A library for the evaluation of molecular integrals of many-body operators over gaussian functions,” http://libint.valeyev.net/ (2016), version 2.2.0.
222. “Regression testing,” https://en.wikipedia.org/wiki/Regression_testing, accessed November 28, 2023.
223. S. Williams, A. Waterman, and D. Patterson, Commun. ACM 52, 65–76 (2009)
224. J. Yliluoma, “Guide into openmp,” https://bisqwit.iki.fi/story/howto/openmp/, accessed June 8, 2023.