QM/MM Simulations on NVIDIA and AMD GPUs

Madushanka Manathunga,† Hasan Metin Aktulga,‡ Andreas W. Götz,*,¶ and Kenneth M. Merz, Jr.*,†

†*Department of Chemistry and Department of Biochemistry and Molecular Biology, Michigan State University, East Lansing, Michigan 48824-1322, United States*

‡*Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan 48824-1322, United States*

¶*San Diego Supercomputer Center, University of California San Diego, La Jolla, California 92093-0505, United States*

E-mail: agoetz@sdsc.edu; merz@chemistry.msu.edu

Abstract

We have ported and optimized the GPU accelerated QUICK and AMBER based *ab initio* QM/MM implementation on AMD GPUs. This encompasses the entire Fock matrix build and force calculation in QUICK including one-electron integrals, two-electron repulsion integrals, exchange-correlation quadrature, and linear algebra operations. General performance improvements to the QUICK GPU code are also presented. Benchmarks carried out on NVIDIA V100 and AMD MI100 cards display similar performance on both hardware for standalone HF/DFT calculations with QUICK and QM/MM molecular dynamics simulations with QUICK/AMBER. Furthermore, with respect to the QUICK/AMBER release version 21, significant speedups are observed for QM/MM molecular dynamics simulations. This significantly increases the range of scientific problems that can be addressed with open-source QM/MM software on state-of-the-art computer hardware.

Introduction

The use of hybrid Quantum Mechanics/Molecular Mechanics (QM/MM) simulations has become increasingly popular in different domains of computational chemistry and biology. Such domains include computer aided drug discovery, investigating enzymatic reaction mechanisms, predicting spectroscopic properties, etc.[1–7] In QM/MM, the molecular system under investigation is partitioned into two regions and one region is described by an accurate quantum chemistry method while the other using a molecular mechanics force field. This allows modeling and simulation of chemical reactions with sufficient accuracy at an affordable computational cost. Often, the limiting factor to use large QM regions or carry out longer time scale simulations is the cost of QM methods. Typically, computing QM forces takes more than 95% of the total QM/MM time. In the recent past, various efforts have been undertaken to develop computationally affordable novel QM methods[8] or reimplement traditional QM methods to harness the power of massively parallel CPU and GPU hardware platforms.[9–40] Most notably, a number of leading quantum chemistry software packages have been empowered with GPU acceleration allowing users to achieve unprecedented simulation speeds and model larger molecular systems efficiently. For instance, our own GPU accelerated QUICK *ab initio* quantum chemistry and density functional theory package is highly efficient on NVIDIA hardware.[39,40] QM/MM simulations with QUICK/AMBER have displayed respectable speedups of up to 53x for a single GPU with respect to a CPU core for a moderate sized QM region size that was benchmarked at the time.[41] However, the GPU hardware landscape has started to significantly expand with new devices flowing in from other vendors such as AMD and Intel. There is a growing demand for such devices due to their attractive price to performance ratios. Therefore, in addition to supporting existing NVIDIA hardware, enabling support for new GPU hardware has become important for traditional computational chemistry software packages. In the present work, we attempt to achieve this task by further improving the performance of QUICK on NVIDIA GPUs, and

porting and optimizing it on AMD cards. By making some necessary changes to AMBER, we also enable more efficient QM/MM calculations on NVIDIA cards and add the ability to utilize AMD GPUs for QM/MM simulations. The next sections of this manuscript are organized as follows. First, we briefly present the theory of QM/MM and concepts of GPU computing. This is followed by details of new improvements to QUICK that enable better performance on NVIDIA cards. The porting of this QUICK version to AMD cards, optimization and changes to AMBER are then discussed. Finally, we benchmark the performance of the latest QM/MM implementation on NVIDIA cards against QUICK/AMBER v21,[41] and on AMD cards.

Computational Methods

In QM/MM, the total energy of a system is given by[42]

$$E_{\text{total}} = E_{\text{MM}} + E_{\text{QM}} + E_{\text{QM-MM}} \tag{1}$$

where E_{MM} is the standard MM energy for all atoms in the MM region and is a sum of bonded and non-bonded energy terms. The term E_{QM} represents the standard QM energy for all atoms in the QM region. If we consider the Kohn-Sham (KS) formalism and the generalized gradient approximation (GGA), the QM energy of a closed shell system is as follows:[43]

$$E_{\text{QM}} = \frac{1}{2}\sum_{A}^{N_{\text{QM}}}\sum_{B}^{N_{\text{QM}}} Z_A Z_B \left|R_A - R_B\right|^{-1} + 2\sum_{i}^{n}\left(\psi_i\right| - \frac{1}{2}\nabla^2\left|\psi_i\right) - \sum_{A}^{N_{\text{QM}}} Z_A \int \rho\left(r\right)\left|r - r_A\right|^{-1} dr + \frac{1}{2}\iint \rho\left(r_1\right)\left|r_1 - r_2\right|^{-1}\rho\left(r_2\right) dr_1 dr_2 + \int f(\rho(r), \nabla\rho(r)) dr. \tag{2}$$

The terms in Eq. 2 account for nuclear-nuclear repulsion, the kinetic energy of electrons, nuclear-electron attraction, electron-electron repulsion and the exchange correlation potential, respectively. In the above equation, N_{QM} is the number of QM atoms, Z_A, Z_B are nuclear charges, R_A and R_B are positions of nuclei A and B, r represent the electronic

coordinates, ψ_i are spatial molecular orbitals, n is the number of occupied orbitals, and ρ is the electron density. Hybrid GGA functionals include an appropriately scaled exchange term that depends on the occupied orbitals. The equations are straight forward to extend for open-shell systems using spin-polarized density functionals within an unrestricted KS formalism.

The third term of Eq. 1, $E_{\text{QM-MM}}$, is the QM/MM interaction energy between the atoms in the QM region and the atoms in the MM region. For a non-polarizable pairwise additive force field like AMBER,[44] it consists of a non-bonded Lennard-Jones potential and the electrostatic interaction between the QM charge density and the surrounding MM point charges:

$$\begin{aligned} E_{\text{QM-MM}} &= \sum_{A}^{N_{\text{QM}}} \sum_{k}^{N_{\text{MM}}} \varepsilon_{Ak} \left[\left(\sigma_{Ak}|R_A - R_k|^{-1}\right)^{12} - \left(\sigma_{Ak}|R_A - R_k|^{-1}\right)^{12} \right] \\ &+ \sum_{A}^{N_{\text{QM}}} \sum_{k}^{N_{\text{MM}}} Z_A Q_k |R_A - R_k|^{-1} + \sum_{k}^{N_{\text{MM}}} Q_k \int \rho(r)|r - R_k|^{-1} dr. \end{aligned} \tag{3}$$

In Eq. 3, N_{MM} is the number of MM atoms, ε_{Ak} and σ_{Ak} are Lennard-Jones parameters for types of QM atoms A and MM atoms k, R_k are are positions of MM atoms, and Q_k are point charges of MM atoms. In mechanical embedding, the last term in Eq. 3 is omitted and the nuclear charges Z_A in the second term are replaced with an effective fixed point charge.

If the QM/MM boundary crosses a covalent bond, care has to be taken to saturate the dangling bond of the QM region. In AMBER, this is achieved with an automated link atom scheme that adds a hydrogen atom at a suitable position of the QM region and retains bonded force field terms with at least one atom in the MM region.[42,45]

In QUICK v21.03, the two-electron repulsion integrals (ERI), exchange correlation (XC) energy, XC potential and their nuclear gradients are computed on the GPU. One-electron integrals (OEI) and their gradients are computed asynchronously on the CPU. This is the version that is distributed with free and open-source AmberTools v21. The underlying algorithm for computing OEI, ERI and their gradients is based on the recurrence relations

scheme developed by Obara-Saika, Head-Gordon and Pople (OSHGP).[46,47] The MM terms are handled by the Sander MD engine in AmberTools.

In the present work, we made significant performance improvements to code paths computing some of the above terms. These include offloading of OEI and OEI gradient calculations to the GPU, in addition to reimplementation of the ERI and ERI gradient device kernels (functions executed on the GPU).

Before proceeding with the associated implementation details, it is important to visit the fundamentals of GPU computing. GPUs use a single instruction multiple data paradigm for executing code and performing work.[48] The graphics processing chip of a GPU comprises a set of execution units (streaming multiprocessors in NVIDIA terminology and compute units in AMD terminology). When programming for GPUs, the work should be organized for and mapped to threads. During code execution, the threads are assigned to execution units in batches (warps in NVIDIA terminology and wave fronts in AMD terminology). The batch sizes for NVIDIA and AMD hardware are 32 and 64, respectively. The execution units execute batches of threads by issuing the same instruction to each thread. There are different memory spaces on the GPU which are physically distinct from the CPU (or host) memory. Global memory is the largest memory space available on a GPU. It is usually several GBs or several dozens of GBs and is accessible by all threads located on all execution units. However, global memory transactions carry a high latency in comparison to other memory operations. A second type of memory space, shared memory, is available on each execution unit. Shared memory can be accessed by threads executed on a given execution unit and the transactions are faster than global memory transactions. A third type of memory space, registers, are available for each thread. Register accesses are the fastest type of memory access. However, only a limited number of registers are available for each thread. If the code being executed requires more than the available number of registers, the additional memory requirement is satisfied by spilling registers into a scratch space (local memory in NVIDIA terminology and

scratch space in AMD terminology) that has the same memory latency as global memory. Additionally, two other read-only memory types, constant and texture memory, are available on the GPU. The associated memory transactions are faster than global memory, but slower than the register file. When porting a code to GPUs, care must be taken to 1) avoid thread divergence, 2) minimize global memory transactions and hide memory latency using appropriate strategies, 3) avoid register spillage. Adhering to such principles will ensure that the code utilizes a high percentage of the peak hardware performance.

General performance improvements to QM/MM implementation

Computing OEIs on the GPU requires similar steps to computing ERIs. These involve prescreening and presorting of OEIs on the host (i.e., CPU), uploading the data to the GPU, computing primitive integrals, computing contracted integrals, and the KS matrix update. OEI gradients can also be computed in an analogous manner. Prescreening of OEIs can be achieved by considering the value of the overlap prefactor, that is $2.0\pi/(\alpha+\beta)exp(-\alpha\beta/(\alpha+\beta)(A-B)^2)$ where α and β are Gaussian exponents centered on atoms A and B. All OEIs that have an overlap prefactor value less than a threshold are excluded.

The existing CPU based OEI code uses this procedure and the implementation is already in place. For presorting, necessary code paths were implemented. Here, OEIs are sorted based on the angular momentum and number of primitives. The sorted integral indices, the product of overlap prefactor and contraction coefficients are then uploaded to the GPU. Source code necessary for these tasks was implemented. OEIs are parallelized based on the number of shells and atoms. To compute primitive integrals, it was necessary to implement device kernels performing the vertical recurrence relation (VRR) algorithm.[46,47] To achieve this task, a Python based CUDA code generator and optimizer (QUICK-GenInt) was developed. The device kernels required to contract primitive integrals and update the KS matrix were written manually. The performance bottleneck of the existing ERI kernels is the higher

register utilization of the associated VRR code. To address this problem, QUICK-GenInt was extended to generate four center integral code paths. The generated VRR device kernels were included in QUICK, compiled and profiled using the NVIDIA Nsight profiler on different NVIDIA GPUs. This allowed us to systematically study the impact of the number of intermediate variables on register usage and to select optimal code paths. Furthermore, larger horizontal recurrence relation (HRR) device kernels were split and organized where appropriate.

Support for AMD GPUs

For porting the QUICK CUDA code to AMD GPUs, translators provided in the HIP toolkit[49] were used. After source to source translation, manual fixing of some code paths was required. This is due to the fact that certain features available in the CUDA toolkit and NVIDIA hardware were not yet available in AMD ROCm. Necessary changes to CMake and the Make based build systems were also introduced. According to initial tests, the performance on MI100 was ~2x slower than on a V100 GPU. This was due to the higher register utilization of kernels on AMD hardware. At this stage, we implemented different kernel versions and profiled them. Based on the register utilization and run times, the most suitable code paths were chosen. The performance of the kernels was systematically studied while varying kernel launch parameters. Based on this study, optimal kernel launch parameters were selected for the AMD version. For BLAS operations and matrix diagonalization, necessary interfaces were written to rocBLAS and rocSolver libraries. Unfortunately, as of ROCm/5.3.0, the symmetric matrix diagonalization routines of rocSolver were not optimized. Due to this reason, the DSYEVD diagonalizer from the Magma library[50,51] was integrated. For this task, required functions were written and changes were made to the QUICK build system. Finally, the CMake build system in AmberTools was updated with appropriate changes to compile the Sander program with AMD HIP support.

Benchmarks

We now present the results of benchmark QM/MM MD simulations of a protein system, the photoactive yellow protein (PYP) in bulk water (see Figure 1). This system has been used in multiple QM/MM studies in the past.[41,52] We considered three different QM regions of PYP. The first, R1, contains a total of 22 atoms and includes the *p*-coumaric acid chromophore and the S-C bond from the CYS69 residue. The second region, R2, contains all atoms in R1 and additionally, the GLH46 and TYR42 residues. The total number of QM atoms in R2 is 49.

The third region, R4, has 159 atoms and contains THR50, ARG52, PHE62, VAL66, ALA67, PRO68, THR70, PHE96, TYR98 in addition to all the atoms in R2. These QM regions contain hydrogen link atoms whenever the QM/MM interface crosses a covalent bond. Note that the nomenclature of QM regions is consistent with previous studies.[41,52] The MM region of the protein was represented by the ff99SB forcefield and the water molecules in the system by SPC/Fw. A QM/MM electrostatics cutoff of 8 ˚A, 0.5 fs time step, two different level of theories (B3LYP/6-31G* and B3LYP/def2-SVP), electrostatic and mechanical embedding (EE and ME) were employed in the simulations. Furthermore, conservative SCF convergence thresholds and integral cutoffs (10^{-6} RMS threshold for density matrix convergence in the SCF, 10^{-8} integral cutoff) were used when computing forces of the QM region. The suitability of the cutoffs were checked by carrying out energy conservation tests (see section S1).

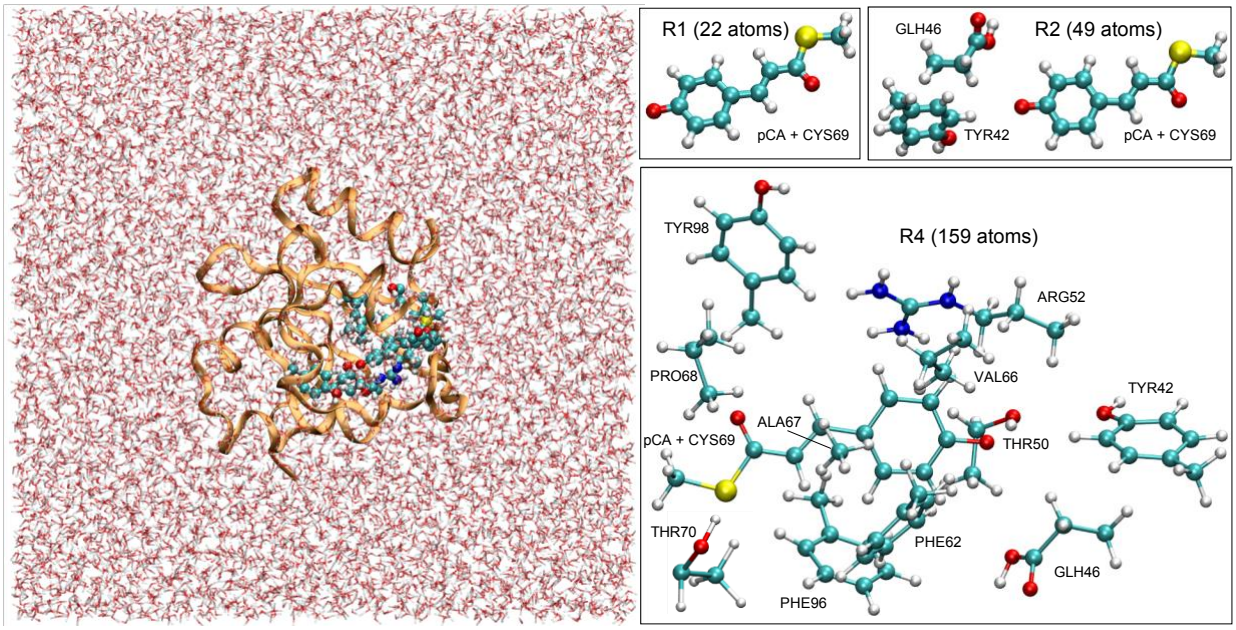

Figure 1: QM/MM setup used for benchmarks. Water box (left) and the protein highlighted in orange are treated at MM level. The chromophore and selected residues in ball and stick representation are treated at the QM level (R4 is shown as an example). Panels on right show different QM regions.

In Table 1, we present the results of all simulations. Information on running simulations and input files are provided in the SI. V100 runs were carried out on a single node containing 4 NVIDIA Volta V100-PCIe (32 GB) type GPUs, Intel Xeon(R) Platinum 8260 (2.40GHz) CPUs and 178 GB memory. For A100 runs, a node containing 4 NVIDIA A100-SXM4 (80 GB) GPUs, AMD EPYC 7713 (2.0 GHz) CPUs and 512 GB memory was used. Both nodes were located in the high performance computing center (iCER HPCC) at Michigan State University. Simulations on AMD GPUs were performed on the AMD cloud platform, specifically, on a node containing 8 AMD Instinct MI100 (32 GB) type GPUs, AMD EPYC 7742 64-Core (2.25 GHz) CPUs and 512 GB memory. The code was compiled on NVIDIA platforms with CUDA/11.4.2, GCC/9.3.0 and OpenMPI/4.0.3. The AMD version was compiled using ROCM/5.3.0 with the same GCC and OpenMPI versions. Magma/2.6.2 was also used in the AMD version. For each simulation, an equal number of CPU cores and GPUs was used. For

comparison, the V100/A100/MI100 GPUs have a peak FP64 performance of 7.8/9.7/11.5 TFLOPs and a global memory bandwidth of 0.9/1.6/1.2 TB/s.

Comparison of simulations carried out using QUICK/AMBER v21[41] and v23 on V100 platforms shows that the latter version is significantly faster than the former. The realized speedup is between 1-2x for all QM regions. We attribute this performance enhancement to improvements made to QUICK. Furthermore, comparison of speedups obtained for EE and ME suggests that former is higher than the latter. This is due to the efficient computation of OEI and OEI gradients on the GPU as a large number of nuclear attraction type integrals and integral derivatives need to be computed with electrostatic embedding. Higher ps/day can be obtained by employing A100 GPUs. The speedups observed for v23 runs on the A100 with respect to the V100 varies between 1 to 2x, depending on the number of atoms and basis functions in the system. This demonstrates that our code can efficiently make use of the increased number of compute units and floating point performance of the A100 GPUs.

Performance comparisons between the V100 and MI100 runs lead to following observations. In the smallest example, V100 runs are up to 2.3x faster with respect to MI100 runs. However, as the system size gets bigger, the performances become similar. The reasons for this behaviour include the use of different linear solvers for matrix diagonalization and differences in hardware platforms. To understand the performance of GPU kernels better, we report kernel run times of all the important device kernels on the V100 and MI100 in Table 2. Note that reported times are average kernel times of the first MD step of each simulation, as reported by the Nsight (on NVIDIA) and the rocprof (on AMD) profiling software. The results suggest that the performance of kernels on the two types of hardware are similar except for XC gradient kernel (see section S2). V100 GPU has a peak FP64 capability up to 7.8 TFLOPS, 0.9 TB/s global memory bandwidth whereas for the MI100, these are 11.5 TFLOPS and 1.2 TB/s respectively.[53,54] Both types of hardware allows a maximum of 255 registers per thread, and currently, this limits the performance of most of

our kernels. Interestingly, the kernels compiled for V100 utilize a lower number of registers per thread in Table 1: Performance comparison (ps/day) of QM/MM MD simulations using QUICK/AMBER v21.03 and v23.03 on different NVIDIA and AMD GPUs. The systems R1, R2 and R4 contain 22, 49 and 159 atoms respectively. The number of basis functions with 6-31G* are 217, 440, 1206 and 244, 509 and 1479 with def2-SVP.

			R1		R2		R4	
GPU	QM/MM type	# GPUs	6-31G*	def2-SVP	6-31G*	def2-SVP	6-31G*	def2-SVP
v21.03								
V100	EE	1	5.36	4.92	1.86	1.49	0.20	0.12
		2	8.74	7.94	3.30	2.60	0.37	0.23
		4	12.17	11.45	5.24	4.12	0.66	0.42
	ME	1	7.57	7.47	2.59	2.25	0.21	0.13
		2	11.01	11.19	4.40	3.87	0.40	0.26
		4	14.60	14.62	6.60	6.00	0.72	0.47
v23.03								
V100	EE	1	7.09	7.69	3.04	2.82	0.37	0.25
		2	10.33	10.87	4.94	4.51	0.65	0.45
		4	13.60	14.00	6.96	6.52	1.04	0.73
	ME	1	7.73	7.76	3.14	2.88	0.38	0.26
		2	10.83	11.09	5.09	4.69	0.68	0.47
		4	14.15	14.20	7.26	6.78	1.12	0.78
A100	EE	1	9.83	9.83	4.17	4.03	0.60	0.42
		2	14.05	14.44	6.55	6.19	1.01	0.72
		4	16.57	17.15	9.32	8.79	1.55	1.13

	ME	1	9.95	9.99	4.25	4.07	0.62	0.44
		2	14.34	14.86	6.73	6.39	1.05	0.76
		4	17.03	17.46	9.59	9.14	1.64	1.19
MI100	EE	1	4.51	4.95	2.38	2.30	0.35	0.25
		2	5.57	6.15	3.43	3.40	0.58	0.42
		4	6.11	6.60	4.34	4.20	0.91	0.67
	ME	1	4.65	5.02	2.43	2.34	0.36	0.26
		2	5.67	6.21	3.56	3.50	0.63	0.44
		4	6.23	6.68	4.46	4.32	0.98	0.70

comparison to the MI100. This is most likely due to the maturity of the CUDA compiler.

Table 2: Average kernel times (s) of a single step of QM/MM MD simulations using QUICK/AMBER v23.03 on NVIDIA V100 and AMD MI100 GPUs. The level of theory used is B3LYP/def2-SVP.

	R1		R2		R4	
Kernel name	V100	MI100	V100	MI100	V100	MI100
OEI	0.02	0.02	0.10	0.09	1.10	1.08
OEI gradient	0.07	0.09	0.33	0.44	3.35	4.45
ERI	0.09	0.12	0.35	0.42	5.89	6.09
ERI gradient	1.47	2.00	5.43	6.24	58.53	56.03
XC	0.10	0.12	0.19	0.24	0.90	1.14
XC gradient	0.34	0.09	0.68	0.20	4.07	2.08

Conclusions and future directions

In the presented work, we made significant improvements to the performance of the QUICK/AMBER QM/MM implementation and enabled support for AMD GPUs. The benchmark results show that realized speedups on NVIDIA GPUs are multi-fold with respect to the previous release version. Performance on the MI100 GPU is similar to that of the V100 for moderate sized QM regions. Using our latest implementation, AMBER users will be able

to carry out more efficient QM/MM simulations on a wider range of hardware choices. As for the future directions of this project, room for performance optimization should be explored further. This will be mostly on the QUICK side. ERI gradient calculations are currently the most expensive task in QM/MM, and the bottleneck of these kernels is the higher register utilization on both NVIDIA and AMD GPUs. Re-implementation of such kernels will help reducing kernel run times on both hardware. Furthermore, the Fock matrix build can be further accelerated by implementing dedicated algorithms for Coulomb and exchange contributions and density fitting algorithms. Currently, the diagonalizer routines in the rocSolver library are not well optimized and due to this reason, the AMD version makes use of matrix diagonalizer routines from the Magma library. The performance of these routines in future ROCm releases should be tested and support for rocSolver diagonalization should be enabled. Another avenue worth exploring is implementing a mixed precision DFT version in QUICK. In principle, such a version should deliver ~2x performance boost with sufficient accuracy for MD simulations.

Data and Software Availability

The features described here will be available in QUICK-23.03 and AmberTools23 official release versions. The latest version of QUICK is available on GitHub (https://github.com/merzlab/QUICK). AmberTools can be downloaded from the AMBER webpage (http://ambermd.org/AmberTools.php). Input files for QM/MM simulations reported in this work are available in supporting information.

Acknowledgement

We thank Gina Sitaraman, Mahdieh Ghazimirsaeed, Leopold Grinberg, Trinayan Baruah from AMD and Kyle Jacobs, Kristopher Keipert from NVIDIA for their useful comments on

technical aspects of our GPU code. We also thank Chip Freitag, Nicholas Curtis, Bryce Mackin, Pak Nin Lui and the AMD corporation for providing access to computer resources. M.M. and K.M.M., Jr. are grateful to the Department of Chemistry and Biochemistry and high-performance computer center (iCER HPCC) at the Michigan State University. This research was supported by the National Science Foundation Grant OAC-1835144 and CSSI Frameworks Grant 2209717. This work also used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation (Grant No. ACI-1053575, resources at the San Diego Supercomputer Center through award TG-CHE130010).

Supporting Information Available

Details on energy conservation tests, performance analysis of XC gradient kernel, running QM/MM simulations on NVIDIA and AMD GPUs, input files for QM/MM simulations reported in this work (qmmm inputs.zip).

References

(1) Lipparini, F.; Mennucci, B. Hybrid QM/classical models: Methodological advances and new applications. *Chem. Phys. Rev.* **2021**, *2*, 041303.

(2) Tzeliou, C. E.; Mermigki, M. A.; Tzeli, D. Review on the QM/MM Methodologies and Their Application to Metalloproteins. *Molecules* **2022**, *27*.

(3) Manathunga, M.; Götz, A. W.; Merz Jr., K. M. Computer-aided drug design, quantummechanical methods for biological problems. *Curr. Opin. Struct. Biol.* **2022**, *75*, 102417.

(4) Morzan, U. N.; Alonso de Arminõ, D. J.; Foglia, N. O.; Ram´ırez, F.; González Lebrero, M. C.; Scherlis, D. A.; Estrin, D. A. Spectroscopy in Complex Environments from QM-MM Simulations. *Chem. Rev.* **2018**, *118*, 4071–4113.

(5) Quesne, M. G.; Borowski, T.; de Visser, S. P. Quantum Mechanics/Molecular Mechanics Modeling of Enzymatic Processes: Caveats and Breakthroughs. *Chem. Eur. J.* **2016**, *22*, 2562–2581.

(6) van der Kamp, M. W.; Mulholland, A. J. Combined Quantum Mechanics/Molecular Mechanics (QM/MM) Methods in Computational Enzymology. *Biochemistry* **2013**, *52*, 2708–2728.

(7) Senn, H. M.; Thiel, W. QM/MM Methods for Biomolecular Systems. *Angew. Chem., Int. Ed.* **2009**, *48*, 1198–1229.

(8) Bannwarth, C.; Caldeweyher, E.; Ehlert, S.; Hansen, A.; Pracht, P.; Seibert, J.; Spicher, S.; Grimme, S. Extended tight-binding quantum chemistry methods. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2021**, *11*, e1493.

(9) Walker, R. C., Götz, A. W., Eds. *Electronic Structure Calculations on Graphics Processing Units: From Quantum Chemistry to Condensed Matter Physics*; Wiley: West Sussex, England, 2016.

(10) Ufimtsev, I. S.; Martinez, T. J. Quantum Chemistry on Graphical Processing Units. 2. Direct Self-Consistent-Field Implementation. *J. Chem. Theory Comput.* **2009**, *5*, 1004–1015.

(11) Ufimtsev, I. S.; Martinez, T. J. Quantum Chemistry on Graphical Processing Units. 3. Analytical Energy Gradients, Geometry Optimization, and First Principles Molecular Dynamics. *J. Chem. Theory Comput.* **2009**, *5*, 2619–2628.

(12) Seritan, S.; Bannwarth, C.; Fales, B. S.; Hohenstein, E. G.; Kokkila-Schumacher, S. I. L.; Luehr, N.; Snyder, J. W.; Song, C.; Titov, A. V.; Ufimtsev, I. S.; Martinez, T. J. TeraChem: Accelerating electronic structure and ab initio molecular dynamics with graphical processing units. *J. Chem. Phys.* **2020**, *152*, 224110.

(13) Seritan, S.; Bannwarth, C.; Fales, B. S.; Hohenstein, E. G.; Isborn, C. M.; Kokkila-Schumacher, S. I. L.; Li, X.; Liu, F.; Luehr, N.; Snyder Jr., J. W.; Song, C.; Titov, A. V.; Ufimtsev, I. S.; Wang, L.-P.; Martinez, T. J. TeraChem: A graphical processing unit accelerated electronic structure package for large-scale ab initio molecular dynamics. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2021**, *11*, e1494.

(14) Andrade, X.; Aspuru-Guzik, A. Real-Space Density Functional Theory on Graphical Processing Units: Computational Approach and Comparison to Gaussian Basis Set Methods. *J. Chem. Theory Comput.* **2013**, *9*, 4360–4373.

(15) Ku¨hne, T. D.; Iannuzzi, M.; Del Ben, M.; Rybkin, V. V.; Seewald, P.; Stein, F.; Laino, T.; Khaliullin, R. Z.; Schu¨tt, O.; Schiffmann, F.; Golze, D.; Wilhelm, J.; Chulkov, S.; Bani-Hashemian, M. H.; Weber, V.; Borˇstnik, U.; Taillefumier, M.; Jakobovits, A. S.; Lazzaro, A.; Pabst, H.; Mu¨ller, T.; Schade, R.; Guidon, M.; Andermatt, S.; Holmberg, N.; Schenter, G. K.; Hehn, A.; Bussy, A.; Belleflamme, F.; Tabacchi, G.; Glöß, A.; Lass, M.; Bethune, I.; Mundy, C. J.; Plessl, C.; Watkins, M.; VandeVondele, J.; Krack, M.; Hutter, J. CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations. *J. Chem. Phys.* **2020**, *152*, 194103.

(16) Eriksen, J. J. Efficient and portable acceleration of quantum chemical many-body methods in mixed floating point precision using OpenACC compiler directives. *Mol. Phys.* **2017**, *115*, 2086–2101.

(17) Barca, G. M. J.; Bertoni, C.; Carrington, L.; Datta, D.; De Silva, N.; Deustua, J. E.;

Fedorov, D. G.; Gour, J. R.; Gunina, A. O.; Guidez, E.; Harville, T.; Irle, S.; Ivanic, J.; Kowalski, K.; Leang, S. S.; Li, H.; Li, W.; Lutz, J. J.; Magoulas, I.; Mato, J.; Mironov, V.; Nakata, H.; Pham, B. Q.; Piecuch, P.; Poole, D.; Pruitt, S. R.; Rendell, A. P.; Roskop, L. B.; Ruedenberg, K.; Sattasathuchana, T.; Schmidt, M. W.; Shen, J.; Slipchenko, L.; Sosonkina, M.; Sundriyal, V.; Tiwari, A.; Galvez Vallejo, J. L.; Westheimer, B.; W loch, M.; Xu, P.; Zahariev, F.; Gordon, M. S. Recent developments in the general atomic and molecular electronic structure system. *J. Chem. Phys.* **2020**, *152*, 154102.

(18) DePrince, A. E. I.; Hammond, J. R. Coupled Cluster Theory on Graphics Processing Units I. The Coupled Cluster Doubles Method. *J. Chem. Theory Comput.* **2011**, *7*, 1287–1295.

(19) A. Eugene DePrince, I.; Kennedy, M. R.; Sumpter, B. G.; Sherrill, C. D. Density-fitted singles and doubles coupled cluster on graphics processing units. *Mol. Phys.* **2014**, *112*, 844–852.

(20) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K. GPU-Based Implementations of the Noniterative Regularized-CCSD(T) Corrections: Applications to Strongly Correlated Systems. *J. Chem. Theory Comput.* **2011**, *7*, 1316–1327.

(21) Bhaskaran-Nair, K.; Ma, W.; Krishnamoorthy, S.; Villa, O.; van Dam, H. J. J.; Apra, E.; Kowalski, K. Noniterative Multireference Coupled Cluster Methods on Heterogeneous CPU–GPU Systems. *J. Chem. Theory Comput.* **2013**, *9*, 1949–1957.

(22) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K.; Agrawal, G. Optimizing tensor contraction expressions for hybrid CPU-GPU execution. *Cluster Comput.* **2013**, *16*, 131–155.

(23) Fales, B. S.; Curtis, E. R.; Johnson, K. G.; Lahana, D.; Seritan, S.; Wang, Y.; Weir, H.; Martinez, T. J.; Hohenstein, E. G. Performance of Coupled-Cluster Singles and Doubles

on Modern Stream Processing Architectures. *J. Chem. Theory Comput.* **2020**, *16*, 4021–4028.

(24) Peng, C.; Calvin, J. A.; Valeev, E. F. Coupled-cluster singles, doubles and perturbative triples with density fitting approximation for massively parallel heterogeneous platforms. *Int. J. Quantum Chem.* **2019**, *119*, e25894.

(25) Peng, C.; Lewis, C. A.; Wang, X.; Clement, M. C.; Pierce, K.; Rishi, V.; Pavosevic, F.; Slattery, S.; Zhang, J.; Teke, N.; Kumar, A.; Masteran, C.; Asadchev, A.; Calvin, J. A.; Valeev, E. F. Massively Parallel Quantum Chemistry: A high-performance research platform for electronic structure. *J. Chem. Phys.* **2020**, *153*, 044120.

(26) Perera, A.; Bartlett, R. J.; Sanders, B. A.; Lotrich, V. F.; Byrd, J. N. Advanced concepts in electronic structure (ACES) software programs. *J. Chem. Phys.* **2020**, *152*, 184105.

(27) Apra, E.; Bylaska, E. J.; de Jong, W. A.; Govind, N.; Kowalski, K.; Straatsma, T. P.; Valiev, M.; van Dam, H. J. J.; Alexeev, Y.; Anchell, J.; Anisimov, V.; Aquino, F. W.; Atta-Fynn, R.; Autschbach, J.; Bauman, N. P.; Becca, J. C.; Bernholdt, D. E.; Bhaskaran-Nair, K.; Bogatko, S.; Borowski, P.; Boschen, J.; Brabec, J.; Bruner, A.; Cauët, E.; Chen, Y.; Chuev, G. N.; Cramer, C. J.; Daily, J.; Deegan, M. J. O.; Dunning, T. H.; Dupuis, M.; Dyall, K. G.; Fann, G. I.; Fischer, S. A.; Fonari, A.; Fruchtl, H.; Gagliardi, L.; Garza, J.; Gawande, N.; Ghosh, S.; Glaesemann, K.; Götz, A. W.; Hammond, J.; Helms, V.; Hermes, E. D.; Hirao, K.; Hirata, S.; Jacquelin, M.; Jensen, L.; Johnson, B. G.; Jonsson, H.; Kendall, R. A.; Klemm, M.; Kobayashi, R.; Konkov, V.; Krishnamoorthy, S.; Krishnan, M.; Lin, Z.; Lins, R. D.; Littlefield, R. J.; Logsdail, A. J.; Lopata, K.; Ma, W.; Marenich, A. V.; Martin del Campo, J.; Mejia-Rodriguez, D.; Moore, J. E.; Mullin, J. M.; Nakajima, T.; Nascimento, D. R.; Nichols, J. A.; Nichols, P. J.; Nieplocha, J.; Otero-de-la Roza, A.; Palmer, B.; Panyala, A.; Pirojsirikul, T.; Peng, B.; Peverati, R.; Pittner, J.; Pollack, L.; Richard, R. M.; Sadayappan, P.; Schatz, G. C.; Shelton,

W. A.; Silverstein, D. W.; Smith, D. M. A.; Soares, T. A.; Song, D.; Swart, M.; Taylor, H. L.; Thomas, G. S.; Tipparaju, V.; Truhlar, D. G.; Tsemekhman, K.; Van Voorhis, T.; Vazquez-Mayagoitia, A.; Verma, P.;
Villa, O.; Vishnu, A.; Vogiatzis, K. D.; Wang, D.; Weare, J. H.; Williamson, M. J.;
Windus, T. L.; Wolinski, K.; Wong, A. T.; Wu, Q.; Yang, C.; Yu, Q.; Zacharias, M.;
Zhang, Z.; Zhao, Y.; Harrison, R. J. NWChem: Past, present, and future. *J. Chem. Phys.* **2020**, *152*, 184102.

(28) Williams-Young, D. B.; de Jong, W. A.; van Dam, H. J. J.; Yang, C. On the Efficient Evaluation of the Exchange Correlation Potential on Graphics Processing Unit Clusters. *Front. Chem.* **2020**, *8*.

(29) Kowalski, K.; Bair, R.; Bauman, N. P.; Boschen, J. S.; Bylaska, E. J.; Daily, J.; de Jong, W. A.; Dunning, T. J.; Govind, N.; Harrison, R. J.; Ke¸celi, M.; Keipert, K.; Krishnamoorthy, S.; Kumar, S.; Mutlu, E.; Palmer, B.; Panyala, A.; Peng, B.; Richard, R. M.; Straatsma, T. P.; Sushko, P.; Valeev, E. F.; Valiev, M.; van Dam, H. J. J.; Waldrop, J. M.; Williams-Young, D. B.; Yang, C.; Zalewski, M.; Windus, T. L. From NWChem to NWChemEx: Evolving with the Computational Chemistry Landscape. *Chem. Rev.* **2021**, *121*, 4962–4998.

(30) Kussmann, J.; Ochsenfeld, C. Hybrid CPU/GPU Integral Engine for Strong-Scaling Ab Initio Methods. *J. Chem. Theory Comput.* **2017**, *13*, 3153–3159.

(31) Laqua, H.; Thompson, T. H.; Kussmann, J.; Ochsenfeld, C. Highly Efficient, Linear-Scaling Seminumerical Exact-Exchange Method for Graphic Processing Units. *J. Chem. Theory Comput.* **2020**, *16*, 1456–1468.

(32) Kussmann, J.; Laqua, H.; Ochsenfeld, C. Highly Efficient Resolution-of-Identity Density Functional Theory Calculations on Central and Graphics Processing Units. *J. Chem. Theory Comput.* **2021**, *17*, 1512–1521.

(33) Miao, Y.; Merz Jr, K. M. Acceleration of Electron Repulsion Integral Evaluation on Graphics Processing Units via Use of Recurrence Relations. *J. Chem. Theory Comput.* **2013**, *9*, 965–976.

(34) Miao, Y.; Merz Jr, K. M. Acceleration of High Angular Momentum Electron Repulsion Integrals and Integral Derivatives on Graphics Processing Units. *J. Chem. Theory Comput.* **2015**, *11*, 1449–1462.

(35) Johnson, K. G.; Mirchandaney, S.; Hoag, E.; Heirich, A.; Aiken, A.; Martinez, T. J. Multinode Multi-GPU Two-Electron Integrals: Code Generation Using the Regent Language. *J. Chem. Theory Comput.* **0**, *0*, null.

(36) Barca, G. M. J.; Alkan, M.; Galvez-Vallejo, J. L.; Poole, D. L.; Rendell, A. P.; Gordon, M. S. Faster Self-Consistent Field (SCF) Calculations on GPU Clusters. *J. Chem. Theory Comput.* **2021**, *17*, 7486–7503.

(37) Seidl, C.; Barca, G. M. J. Q-Next: A Fast, Parallel, and Diagonalization-Free Alternative to Direct Inversion of the Iterative Subspace. *J. Chem. Theory Comput.* **2022**, *18*, 4164–4176.

(38) Vallejo, J. L. G.; Barca, G. M.; Gordon, M. S. High-performance GPU-accelerated evaluation of electron repulsion integrals. *Mol. Phys.* **2022**, *0*, e2112987.

(39) Manathunga, M.; Miao, Y.; Mu, D.; Götz, A. W.; Merz Jr., K. M. Parallel Implementation of Density Functional Theory Methods in the Quantum Interaction Computational Kernel Program. *J. Chem. Theory Comput.* **2020**, *16*, 4315–4326.

(40) Manathunga, M.; Jin, C.; Cruzeiro, V. W. D.; Miao, Y.; Mu, D.; Arumugam, K.; Keipert, K.; Aktulga, H. M.; Merz Jr., K. M.; Götz, A. W. Harnessing the Power of Multi-GPU Acceleration into the Quantum Interaction Computational Kernel Program. *J. Chem. Theory Comput.* **2021**, *17*, 3955–3966.

(41) Cruzeiro, V. W. D.; Manathunga, M.; Merz Jr., K. M.; Götz, A. W. Open-Source Multi-GPU-Accelerated QM/MM Simulations with AMBER and QUICK. *J. Chem. Inf. Model.* **2021**, *61*, 2109–2115.

(42) Götz, A. W.; Clark, M. A.; Walker, R. C. An extensible interface for QM/MM molecular dynamics simulations with AMBER. *J. Comput. Chem.* **2014**, *35*, 95–108.

(43) Pople, J. A.; Gill, P. M.; Johnson, B. G. Kohn—Sham density-functional theory within a finite basis set. *Chem. Phys. Lett.* **1992**, *199*, 557–560.

(44) Ponder, J.; Case, D. A. Force fields for protein simulations. *Adv. Prot. Chem.* **2003**, *66*, 27–85.

(45) Walker, R. C.; Crowley, M. F.; Case, D. A. The implementation of a fast and accurate QM/MM potential method in Amber. *J. Comput. Chem.* **2008**, *29*, 1019–1031.

(46) Obara, S.; Saika, A. Efficient recursive computation of molecular integrals over Cartesian Gaussian functions. *J. Chem. Phys.* **1986**, *84*, 3963–3974.

(47) Head-Gordon, M.; Pople, J. A. A method for two-electron Gaussian integral and integral derivative evaluation using recurrence relations. *J. Chem. Phys.* **1988**, *89*, 5777–5786.

(48) *Professional CUDA C Programming*, 1st ed.; Wrox Press Ltd.: GBR, 2014.

(49) AMD, HIP Programming Guide v4.5. https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-GUIDE.html.

(50) Tomov, S.; Dongarra, J.; Baboulin, M. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Comput.* **2010**, *36*, 232–240, Parallel Matrix Algorithms and Applications.

(51) Brown, C.; Abdelfattah, A.; Tomov, S.; Dongarra, J. J. Design, Optimization, and

Benchmarking of Dense Linear Algebra Algorithms on AMD GPUs. 2020 IEEE High Performance Extreme Computing Conference, HPEC 2020, Waltham, MA, USA, September 22-24, 2020. 2020; pp 1–7.

(52) Isborn, C. M.; Götz, A. W.; Clark, M. A.; Walker, R. C.; Martinez, T. J. Electronic Absorption Spectra from MM and ab Initio QM/MM Molecular Dynamics: Environmental Effects on the Absorption Spectrum of Photoactive Yellow Protein. *J. Chem. Theory Comput.* **2012**, *8*, 5092–5106.

(53) NVIDIA, NVIDIA Tesla V100 GPU Architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, accessed 11/08/2022.

(54) AMD, AMD Instinct MI100 Accelerator. https://www.amd.com/system/files/documents/instinct-mi100-brochure.pdf, accessed 11/08/2022.

TOC Graphic