JCTC Journal of Chemical Theory and Computation

pubs.acs.org/JCTC Article

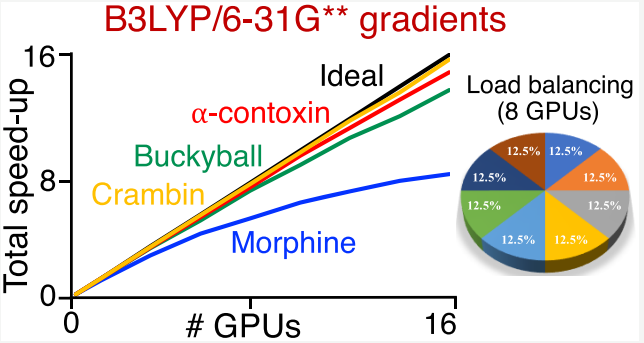
# Harnessing the Power of Multi-GPU Acceleration into the Quantum Interaction Computational Kernel Program

Madushanka Manathunga, Chi Jin, Vinícius Wilian D. Cruzeiro, Yipu Miao, Dawei Mu, Kamesh Arumugam, Kristopher Keipert, Hasan Metin Aktulga, Kenneth M. Merz, Jr.,* and Andreas W. Götz*

Cite This: *J. Chem. Theory Comput.* 2021, 17, 3955−3966 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

**ABSTRACT:** We report a new multi-GPU capable *ab initio* Hartree−Fock/density functional theory implementation integrated into the open source QUantum Interaction Computational Kernel (QUICK) program. Details on the load balancing algorithms for electron repulsion integrals and exchange correlation quadrature across multiple GPUs are described. Benchmarking studies carried out on up to four GPU nodes, each containing four NVIDIA V100-SXM2 type GPUs demonstrate that our implementation is capable of achieving excellent load balancing and high parallel efficiency. For representative medium to large size protein/organic molecular systems, the observed parallel efficiencies remained above 82% for the Kohn−Sham matrix formation and above 90% for nuclear gradient calculations. The accelerations on NVIDIA A100, P100, and K80 platforms also have realized parallel efficiencies higher than 68% in all tested cases, paving the way for large-scale *ab initio* electronic structure calculations with QUICK.




## 1. INTRODUCTION

At the dawn of the exascale computing era, multiple graphics processing unit (multi-GPU) execution has become inevitable for high performance computing applications. Software packages from various fields such as artificial intelligence[1] and numerical weather prediction[2] are already harvesting the power of hundreds and thousands of GPUs. While a single GPU is capable of performing trillions of floating point operations per second, outperforming single or even multiple modern central processing units (CPUs), properly engineered scientific applications are able to exploit an enormous amount of computational power on multi-GPU platforms.

The power of using multiple GPUs has been harnessed into a range of traditional computational chemistry tools,[3−57] including a range of *ab initio* electronic structure software packages.[17−57] However, among these are only a few Gaussian function based quantum chemistry codes for mean-field Hartree−Fock (HF) and density functional theory (DFT) calculations.[18−22,41,55−57] Meanwhile, with multi-GPU nodes increasingly becoming common in contemporary supercomputer centers, open-source quantum chemical codes that can fully exploit their power are in demand. Toward this end, we have further improved our open source quantum chemistry software package called QUantum Interaction Computational Kernel (QUICK)[58−60] by incorporating multi-GPU capabilities. QUICK is capable of performing efficient *ab initio* HF and DFT energy and gradient calculations with Gaussian basis sets. The CPU version of QUICK is parallelized via the message passing interface (MPI),[61] and excellent performance was recently demonstrated both for the CPU and the GPU version.[60] For instance, B3LYP energy and gradient calculations performed on a single NVIDIA V100 were shown to be, respectively, ∼30−90 times and ∼35−60 times faster than the same calculations performed using a single Intel Xeon Skylake CPU core.[60] In the QUICK GPU version, the electron repulsion integrals (ERIs), the exchange correlation (XC) energy and potential, and their derivatives with respect to nuclear coordinates are computed on the GPU. The ERIs are computed using vertical and horizontal recurrence relationships reported by Obara, Saika, Head-Gordon, and Pople (OSHGP algorithm).[62,63] The XC contributions are calculated based on a scheme developed by Pople and co-workers.[64] In addition to computing the above quantities, assembling the Fock matrix and gradient vector are also done on the GPU.

**Received:** February 8, 2021
**Published:** June 1, 2021






https://doi.org/10.1021/acs.jctc.1c00145
*J. Chem. Theory Comput.* 2021, 17, 3955−3966

Among the few publications found in the literature regarding the multi-GPU implementation of ERIs, Ufimtsev and Martinez's work[18] is perhaps the earliest. In this implementation, the Coulomb and exchange ERIs are first organized into two matrices in which the rows and column indices correspond to bra and ket pairs of primitive integrals. The matrices are then sorted based on (1) the angular momentum of each bra and ket pair, (2) each pair's contribution to the Schwarz upper bound. Next, different rows of the matrices are cyclically mapped to available GPUs such that each GPU computes a subset of the Coulomb and exchange integrals. A similar approach is used for parallelizing ERI gradient calculations.[19] The observed speed-ups using this approach were reasonable, for instance 2 to 2.8-fold for computing ERIs on 3 GTX280 cards and 3 to 3.5-fold for computing ERI gradients on 2 GeForce 295GTX cards each having 2 graphics processors. Extending this work further, the authors reported SCF calculations of different molecular systems on up to 128 GPU nodes in a later article.[20] The ERI calculations displayed linear scaling through 128 nodes, however, the linear algebra demonstrated linear scaling only between 4 to 16 nodes. A second multi-GPU capable ERI engine was reported by Kussman and Ochsenfeld,[55] and quite recently, a fragmentation based Fock build algorithm with dynamic load balancing was reported by Gordon and co-workers.[65] In the context of XC parallelization on multi-GPUs, Williams-Young et al.[52] documented a three level parallelization scheme. In such a scheme, the load balancing is achieved by pre-estimating the floating point operations (FLOPs) incurred by batches of grid points.

Our multi-GPU implementation consists of the following features. The message passing interface (MPI) is used to set up the calculation and communicate between compute ranks (*i.e.* different CPU cores) hosting GPUs. The ERI workload is statically distributed among the GPUs. The XC workload parallelization is performed in two stages, with the second being a load rebalancing stage for the XC gradients. The next sections of this manuscript are organized as follows: In section 2, we briefly revisit some of the theoretical concepts essential to describe the implementation. Since the practical computational implementation of HF and DFT methods are not distinct from each other, we focus on the Kohn–Sham formalism to drive the discussion. The details of the multi-GPU parallelization are then presented in section 3. Here we first discuss the important aspects of multi-GPU programming and present an implementation that follows this philosophy. In section 4, benchmarking results are presented and discussed. The tests provide insight into the scaling of the ERI and XC algorithms on several widely used NVIDIA GPU types. Finally, in section 5, we conclude our discussion by exploring directions for further improvement.

## 2. THEORY

In the Kohn–Sham formalism, the total electronic energy ($E$) of a closed shell system within the generalized gradient approximation (GGA) is given by,[64]

$$E = \sum_{i}^{n}\left(\psi_i\left| -\frac{1}{2}\nabla^2\right|\psi_i\right) - \sum_{A}^{\text{nucl.}} Z_A \int \rho(r)|r - r_A|^{-1}\,\mathrm{d}r + \frac{1}{2}\iint \rho(r_1)|r_1 - r_2|^{-1}\rho(r_2)\,\mathrm{d}r_1\,\mathrm{d}r_2 + \int f(\rho(r), \nabla\rho(r))\,\mathrm{d}r \tag{1}$$

where the first term is the kinetic energy of the electrons, the second is the electron–nuclear interaction energy, the third is the Coulomb self-interaction energy of the electron density and the fourth is for the exchange correlation energy. Furthermore, the $\psi_i$ are spatial molecular orbitals, $Z_A$ is the charge of nuclei $A$, and $\rho$ is the electron density expressed as

$$\rho = 2\sum_{i}^{n}|\psi_i|^2 \tag{2}$$

where $n$ is the number of occupied orbitals. In practice, calculation of the energy using eq 1 requires expressing the molecular orbitals and the electron density in terms of basis functions (atomic orbitals),

$$\psi_i = \sum_{\mu}^{N} C_{\mu i}\phi_\mu \tag{3}$$

$$\rho = 2\sum_{\mu}^{N}\sum_{\nu}^{N}\sum_{i}^{n} C_{\mu i}C_{\nu i}\phi_\mu\phi_\nu = \sum_{\mu\nu} P_{\mu\nu}\phi_\mu\phi_\nu \tag{4}$$

where $\phi_\mu$ ($\mu = 1, ..., N$) are the atomic orbitals, $C_{\mu i}$ and $C_{\nu i}$ are molecular orbital coefficients and $P_{\mu\nu}$ is the density matrix. Substituting eqs 3 and 4 into eq 1 and minimizing with respect to the molecular orbital coefficients under orthonormality constraints leads to a series of linear equations represented by the Kohn–Sham matrix ($K_{\mu\nu}$),

$$K_{\mu\nu} = H_{\mu\nu}^{\text{core}} + J_{\mu\nu} + K_{\mu\nu}^{\text{XC}} \tag{5}$$

Here $H_{\mu\nu}^{\text{core}}$ is the one electron operator matrix. $J_{\mu\nu}$ is the Coulomb matrix given by

$$J_{\mu\nu} = \sum_{\lambda\sigma}^{N} P_{\lambda\sigma}(\mu\nu|\lambda\sigma) \tag{6}$$

where the ERIs over atomic orbitals are defined as

$$(\mu\nu|\lambda\sigma) = \int \frac{\phi_\mu(r_1)\phi_\nu(r_1)\phi_\lambda(r_2)\phi_\sigma(r_2)}{|r_1 - r_2|}\,\mathrm{d}r_1\,\mathrm{d}r_2 \tag{7}$$

$K_{\mu\nu}^{\text{XC}}$ is the XC potential contribution to the Kohn–Sham matrix expressed as

$$K_{\mu\nu}^{\text{XC}} = \int\left[\frac{\partial f}{\partial\rho}\phi_\mu\phi_\nu + \left(3\frac{\partial f}{\partial\gamma}\nabla\rho\right)\cdot\nabla(\phi_\mu\phi_\nu)\right]\mathrm{d}r \tag{8}$$

where $\gamma = |\nabla\rho|^2$ is the gradient invariant.

The computationally most expensive task in building the Kohn–Sham matrix is computing the ERIs required in eq 6. In practice, atomic basis functions are constructed as a linear combination of primitive atom centered Cartesian Gaussian functions and the contracted ERIs can be written in terms of primitive ones:

$$(\mu\nu|\lambda\sigma) = \sum_{abcd} c_{\mu a}c_{\nu b}c_{\lambda c}c_{\sigma d}[ab|cd] \tag{9}$$



https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955–3966

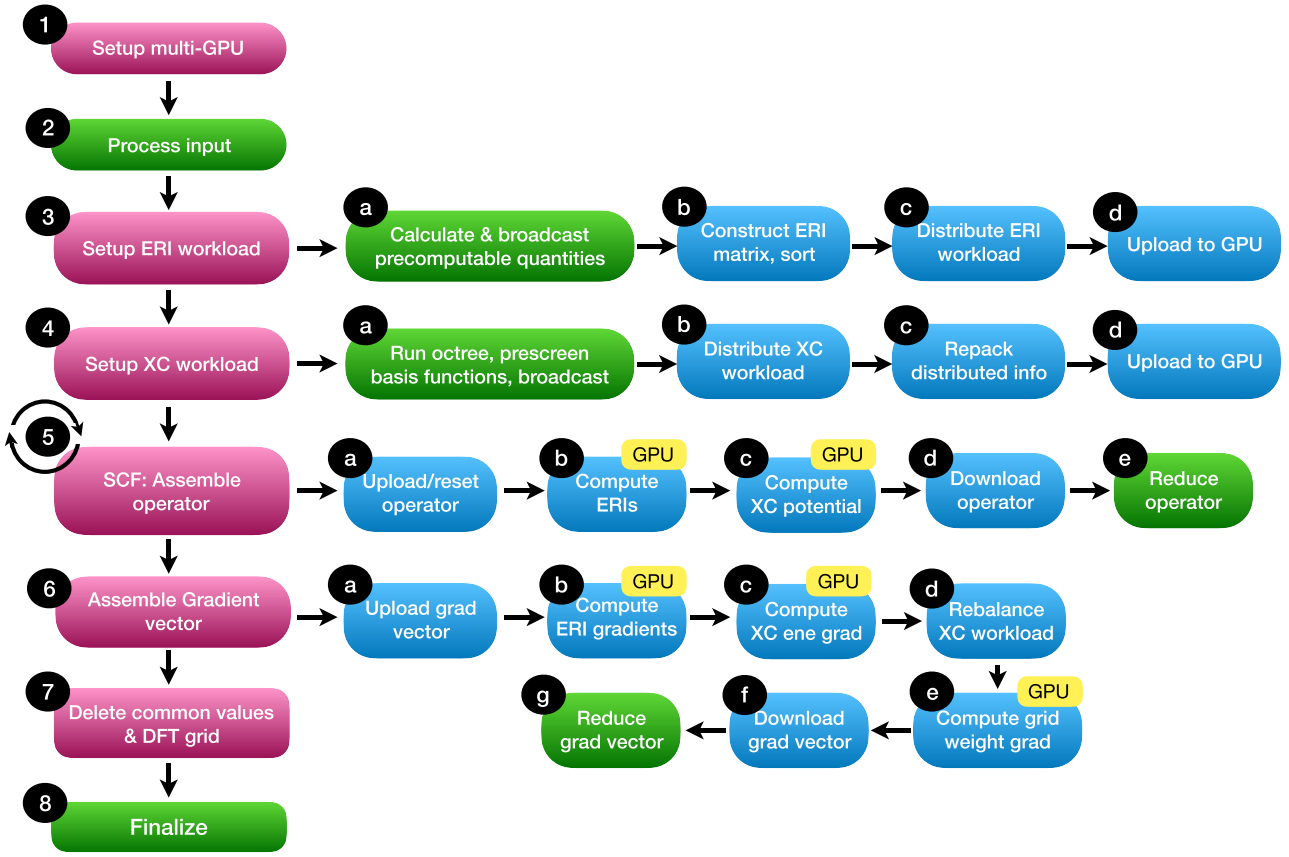



**Figure 1.** Flowchart depicting the multi-GPU workflow of a DFT gradient calculation. Major steps and substeps are denoted by purple and light blue color boxes, respectively. Steps indicated by green boxes are performed only on the root compute rank. Circle arrows indicate iterative steps. Steps marked with yellow boxes containing "GPU" are performed only on the GPU. One-electron integrals and gradients (not shown) are asynchronously computed on the CPU during ERI and ERI gradient steps, respectively. CPUs remain idle during GPU steps 5c, 6c, and 6e.

Here $a$, $b$, $c$, and $d$ are the primitive function indices and $c_{\mu a}$, $c_{\nu b}$, $c_{\lambda c}$, and $c_{\sigma d}$ are the contraction coefficients. Primitive ERIs can be computed and assembled into contracted ERIs using an established algorithm such as OSHGP.[62,63] The second most expensive contribution for constructing the Kohn−Sham matrix is calculating the XC potential. Because of the complexity of XC functionals, this quantity is obtained numerically, involving the formation of a quadrature grid, in which quantities such as electron densities, value of the basis functions, and their gradients are computed at each grid point.

An expression for the molecular gradients can be obtained from eq 1 by differentiating with respect to nuclear coordinates.

$$\begin{aligned}\nabla_A E = &\sum_{\mu\nu}^{N} P_{\mu\nu}(\nabla_A H_{\mu\nu}^{\text{core}}) \\ &+ \frac{1}{2}\sum_{\mu\nu\lambda\sigma}^{N} P_{\mu\nu}P_{\lambda\sigma}\nabla_A(\mu\nu|\lambda\sigma) - \sum_{\mu\nu}^{N} W_{\mu\nu}(\nabla_A S_{\mu\nu}) \\ &- 4\sum_{\mu}^{\prime}\sum_{\nu}^{N} P_{\mu\nu}\int\left[\frac{\partial f}{\partial\rho}\phi_\nu\nabla\phi_\mu + X_{\mu\nu}\left(\frac{\partial f}{\partial\gamma}\nabla\rho\right)\right]\mathrm{d}r\end{aligned} \tag{10}$$

Here $S_{\mu\nu}$ is the overlap matrix, and the primed sum in the fourth term indicates that only $\mu$ centered on nucleus A is considered for the summation. $W_{\mu\nu}$ is the energy weighted density matrix and $X_{\mu\nu}$ is a matrix element given by

$$X_{\mu\nu} = \phi_\nu\nabla(\nabla\phi_\mu)^T + (\nabla\phi_\mu)(\nabla\phi_\nu)^T \tag{11}$$

Similar to eq 5, the most expensive terms in eq 10 are computing ERI gradients (second term) and XC gradients (fourth term).

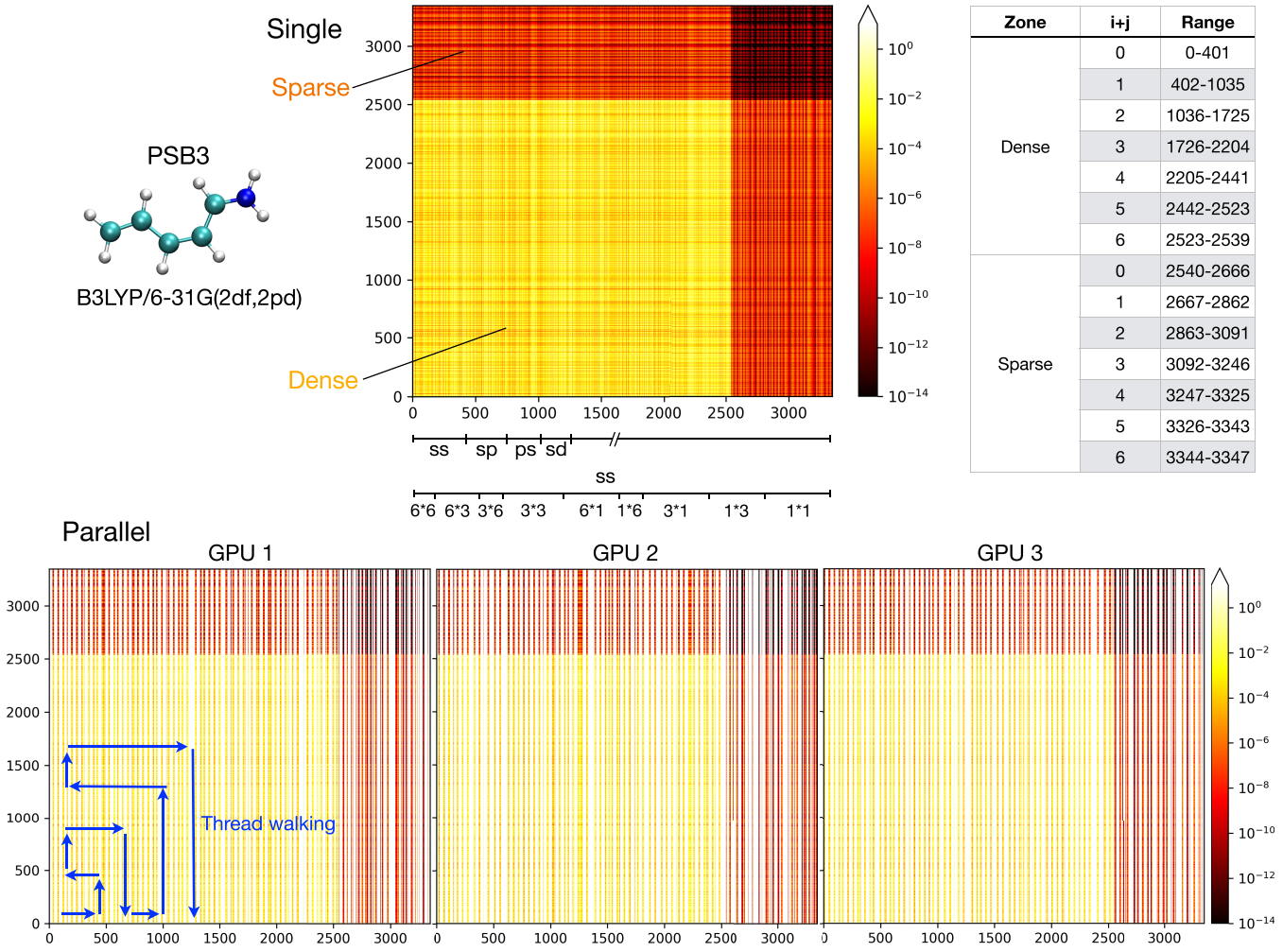
## 3. IMPLEMENTATION

### 3.1. Key Considerations in Multi-GPU Programming.

GPUs allow massive data parallel computations in comparison to classic CPU platforms. However, their hardware architecture is more complex and one needs a proper understanding of the execution and memory models and available multi-GPU programming models in order to write an efficient application. In the context of execution, GPUs use a single instruction multiple data paradigm for performing work.[66] At the microarchitectural level, the graphics processing chip of a GPU consists of a series of streaming multiprocessors. A programmer should organize and map the work to threads which are then assigned to streaming multiprocessors as thread blocks and executed as warps of a certain size (32 for recent NVIDIA architectures). The streaming multiprocessors execute warps by issuing the same instruction for each thread. Therefore, branching in the code should be minimized to avoid thread divergence which leads to performance penalties. A GPU (device) carries its own memory spaces which are physically distinct from the CPU (or host) memory.[66] The main type, called global memory, is the largest and accessible to threads located on all streaming multiprocessors. Typically,



https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966




| Zone | i+j | Range |
|---|---|---|
| Dense | 0 | 0-401 |
| | 1 | 402-1035 |
| | 2 | 1036-1725 |
| | 3 | 1726-2204 |
| | 4 | 2205-2441 |
| | 5 | 2442-2523 |
| | 6 | 2523-2539 |
| Sparse | 0 | 2540-2666 |
| | 1 | 2667-2862 |
| | 2 | 2863-3091 |
| | 3 | 3092-3246 |
| | 4 | 3247-3325 |
| | 5 | 3326-3343 |
| | 6 | 3344-3347 |

**Figure 2.** Distribution of the ERI workload and thread walking. In the single GPU version, a single CPU core constructs the half ERI matrix of a protonated Schiff base (PSB3, top left) and sorts ERIs based on the type, number of primitives, and the estimated value. The row and column indices of the matrix correspond to bra and ket pairs and the colors denote magnitude of the ERI value. The table on the top right indicates the boundaries for different ERI types. In the parallel GPU version, each compute rank prepares and sorts an ERI matrix based on the same criteria and additionally runs a distribution algorithm, excluding bra types (indicated by white vertical strips) and thus keeping only a set of ERIs that would be computed on the corresponding GPU. See text for details on the distribution algorithm.

several GBs of global memory is available on a GPU, however, global memory transactions suffer from high memory latency. A second type of memory called shared memory is available on each streaming multiprocessor, but is relatively small and only accessible by the threads being executed on the same streaming multiprocessor. The constant and texture memory are read only memory types accessible to all threads. These are available in small quantities and the transactions are faster than global memory transactions. Additionally, a certain number of registers is available for threads in the same warp. Register transactions are the fastest; however, their number is very limited. Careful use of these memory spaces is essential to write a performant GPU application.

Different approaches have been discussed in the literature for how to architecture a multi-GPU program.[67] One option is to employ CUDA streams to coordinate the use of multiple GPUs in a single compute node. Alternatively one can write a multithreaded program using OpenMP or POSIX threads[68] for which each thread handles a different GPU in a single compute node. Another option is to employ MPI and allow each compute rank to handle one GPU. The latter option has the advantage that it allows the utilization of devices from multiple nodes, however, at the expense of sacrificing some performance on a single node with respect to CUDA streams. In MPI based multi-GPU programming, one can employ a root-worker model and design algorithms to eliminate the communications between devices. In the root-worker model, the root compute rank assigns computations to worker ranks, the worker ranks perform the computations, and the results are returned to the root. Alternatively, algorithms with device−device communication can be achieved using CUDA-aware MPI technology. Previously, we implemented ERI and XC schemes in a single GPU version of QUICK following the previously discussed execution and memory models. As detailed below, we implement the parallel multi-GPU version adhering to the same philosophy and employing the MPI based root-worker model without direct communication between the GPUs.

**3.2. Parallelizing ERI and ERI Gradient Schemes.** The existing implementation of the ERI engine in QUICK can be mainly divided into four parts.[58,59] The first part comprises several host functions that process molecular and basis set information, compute Schwarz cutoff values, and perform


https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966

presorting of ERIs. Handling CPU−GPU data transfer such as uploading molecular and basis set information, construction of the ERI matrix and downloading the Kohn−Sham matrix are also performed by the functions in the first part. In the second part, there exist several global kernels (i.e., GPU capable functions that can be directly invoked from the host) that go through the $\mu$, $\nu$, $\lambda$, and $\sigma$ indices and invoke kernels that perform the horizontal recurrence relations (HRR) step of the OSHGP algorithm. Assembling the Kohn−Sham matrix is also performed here. The HRR step is carried out by a set of device kernels (GPU capable functions that cannot be directly invoked from the host) belonging to the third part. The fourth part contains a set of complex machine generated device kernels. These kernels perform the vertical recurrence relations (VRR) step.

To extend the above implementation to multi-GPUs, changes are required only for the first two parts. We first assign the GPUs to CPU cores (from now on compute ranks) depending on their local ranks. Input processing and calculating precomputable quantities are done on the root compute rank (see Figure 1). The calculated information is then broadcast to worker ranks. Each compute rank uploads molecular and basis set information and Schwarz cutoff values to their assigned GPUs. The next step is presorting the ERIs. As documented previously,[18,19,59] presorting helps to minimize the thread divergence during ERI computation by ensuring that threads in a warp receive the same instructions to the largest possible extent. In the existing presorting scheme, the four-index ERIs are treated as an $N^2 \times N^2$ matrix problem with horizontal and vertical directions represented by a bra (ab| and ket |cd). The elements of such an ERI matrix are organized by four different criteria in each dimension. First, ERIs are separated into dense or sparse regions based on the Schwarz cutoff value (see Figure 2). The pairs with values greater than $10^{-4}$ fall into the dense zone, while the remaining ones fall into the sparse. Then, ERIs in each zone are sorted based on their shell type, resulting in subzones (type-zones) such as ss, sp, ps, etc. in the matrix. Third, pairs within each type-zone are sorted based on the number of primitive functions creating primitive-zones. Finally, elements in primitive-zones are sorted based on the Schwarz cutoff values. The resulting ERI matrix is used to determine the order of ERI calculation by navigating from one matrix element to another (called thread-walking). In our multi-GPU version, this procedure is replicated on each compute rank, eliminating the need to broadcast the ERI matrix. At this stage, the workload distribution takes place. Focusing on the horizontal direction of the ERI matrix, we divide bra types in the dense region among compute ranks. More specifically, for every compute rank, bins of bra types are created, and the total number of items and the primitive functions are tracked. The assignment of a given bra is then performed by considering its primitive count. The same procedure is repeated for the sparse region and the resulting ERI matrices are well balanced in terms of elements inside each region and workload of shell types (see Figure 2). On the basis of the prepared bins, a set of binary flags is created for every compute rank and uploaded to the global memory of the assigned GPU, and the array pointers are stored in constant memory. During ERI and ERI gradient computation, each thread works on a contracted ERI after checking the value of the corresponding binary flag.

### 3.3. Parallelization of XC and XC Gradient Schemes.

The XC potential calculation in the single GPU implementation follows a scheme involving three major steps.[60] The first step performs grid operations. Here the numerical grid is formed, weights are computed, and the grid is pruned based on the values of the weights. Next, the remaining points are partitioned in space using an octree algorithm as described in our previous work.[60] The values of atom centered basis and primitive functions are then computed at grid points in each spatial bin. The points that have at least one significant basis function are retained in the bin, while the rest is eliminated. Lists of significant basis and primitive function indices are also prepared for each bin and locator maps are constructed to facilitate the retrieval of indices from the lists. Finally, the grid information, basis and primitive function index lists and corresponding maps are uploaded to the GPU. In the second and third steps, electron densities and the XC potential are computed on the GPU. The potential contributions are assembled into the Kohn−Sham matrix residing in global memory as they are computed in later steps.

In our multi-GPU version, the majority of the operations of the first step is done on the root rank (see Figure 1). This includes grid generation, weight computation, pruning, and the preparation of the basis/primitive function index lists and maps. The time spent on such tasks is considerably small and parallelization on multi-GPUs is deemed unnecessary. Prepared data structures are then broadcast to the worker compute ranks. All ranks then run a load distribution algorithm. Here the bins are sorted based on the number of grid points or the product of the grid point-primitive function count. Sorted bins are assigned to ranks using a round robin algorithm, and lists of binary flags are created to record the assignment. At this stage, each rank picks up the assigned list of binary flags and repacked grid points, basis and primitive function lists, and locator maps. It is important to note that unlike ERI kernels, XC kernels perform a large amount of frequent global memory transactions and repacking is vital to maintain coalesced memory access patterns and, hence, kernel performance. The ranks then repack data on the host and upload to their GPUs. Since each rank independently works on a subset of numerical grid points, the kernels performing the second and third steps do not require any changes. The computed XC potentials are assembled into Kohn−Sham operators maintained by each rank. During a given SCF (i.e., self-consistent field) iteration, worker ranks download copies of the operator from the GPU and send them to the root rank to perform the reduction and operator diagonalization.

The calculation of the XC energy nuclear gradients is a two-step procedure implemented in separate kernels in the serial GPU version. The first computes the XC energy gradients and can be used in the multi-GPU version as is. The second, a grid weight gradient computation, is only required for points for which the grid weight is not equal to unity and is dependent on the XC energy at a given grid point, a quantity computed by the former kernel. In the single GPU version, points are filtered on the host and reuploaded to the GPU prior to the second kernel launch. Since different ranks in the multi-GPU version work on subsets of grid points, they may end up with an unequal number of grid points, thus leading to a workload imbalance. Therefore, a load rebalancing step is required after the filtering. Here compute ranks communicate with each other to determine the minimum number of grid points that should be transferred to achieve a balanced workload and then transfer the data accordingly. Following the rebalancing step, the data are uploaded, grid weight gradients are computed and


https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966

assembled into individual gradient vectors. At the end of the calculation, the vectors are downloaded, and are reduced in an analogous way to the Kohn−Sham operator.

## 4. BENCHMARK RESULTS AND DISCUSSION

**4.1. Benchmarking the Multi-GPU Implementation.** Below we present the benchmarking results of our multi-GPU implementation. In past work, we have compared QUICK serial CPU, MPI parallel CPU, and single GPU performance against another GPU capable quantum chemical code.[60] We therefore limit current benchmarks to QUICK single- versus multi-GPU comparisons. Figure 3 depicts the organic molecules and protein systems that we have chosen for our benchmarks.

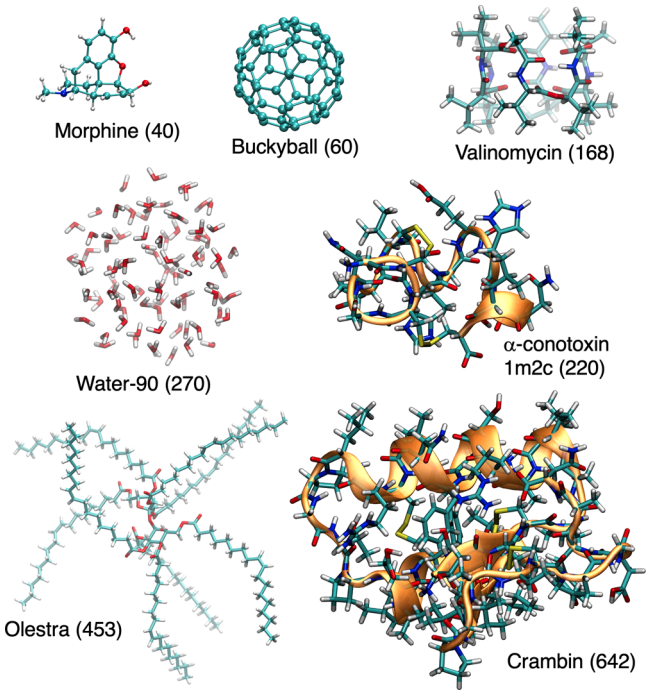



**Figure 3.** Molecules used for benchmarks in this work. The number of atoms is listed in parentheses.

First, the performance of B3LYP gradient calculations on multiple GPUs is analyzed using olestra ($C_{156}H_{278}O_{19}$, see Figure 3 for molecular structure) with three different basis sets. The goal here is to analyze the parallel efficiency of the ERI and XC contributions to the Kohn−Sham operator matrix and the corresponding gradient computation tasks for basis sets with different maximum angular momentum quantum number and basis functions with different contraction levels. Second, a similar investigation is carried out using systems of different sizes; but with the same basis set, aiming to analyze the impact of system size on performance and scalability.

The selected platform for both tests includes four GPU nodes from the recently assembled Expanse cluster at the San Diego Supercomputer Center (SDSC). Each node has four NVIDIA Volta V100-SXM2 type GPUs (32 GB) hosted by two 20-core Intel Xeon Gold 6248 CPUs (2.50 GHz) with 374 GB memory. The nodes are interconnected by 100 GB/s HDR InfiniBand technology. The QUICK code was compiled using the GNU/8.3.1 compiler tool chain, CUDA/10.2 and OpenMPI/4.0.4 with optimization level 2 (-O2). For all calculations, the density matrix cutoff and XC grid pruning cutoff was set to $10^{-8}$. The number of CPU cores employed for a calculation was set to the number of GPUs being used. Prior to the benchmark runs, performance of different ERI thread walking strategies were compared using a set of HF calculations (see Figure S1, Table S1) and circular thread walking was chosen for the ERI and ERI gradient computation. As documented previously,[59] the sorting procedure results in an ERI matrix in which large and small-valued ERIs are most likely distributed circularly at the origin or edge of the matrix. The fact that circular thread-walking displayed better performance over other strategies in the multi-GPU version suggests that dummy regions introduced to ERI matrices of individual compute ranks, which results in idle threads, does not cause significant thread divergence. On the basis of a second comparison (Table S3), numerical grid point count (rather than the product of primitive function and grid point count) based XC load balancing was selected for all benchmarks. This occurs because both methods display similar performance.

In Figure 4, we report the speed-ups (calculated as $T(\text{serial})/T(n)$ where $T(\text{serial})$ and $T(n)$ are the wall times using a single and $n$ GPUs respectively) of the ERI, ERI gradient, XC potential, and XC gradient calculation for olestra

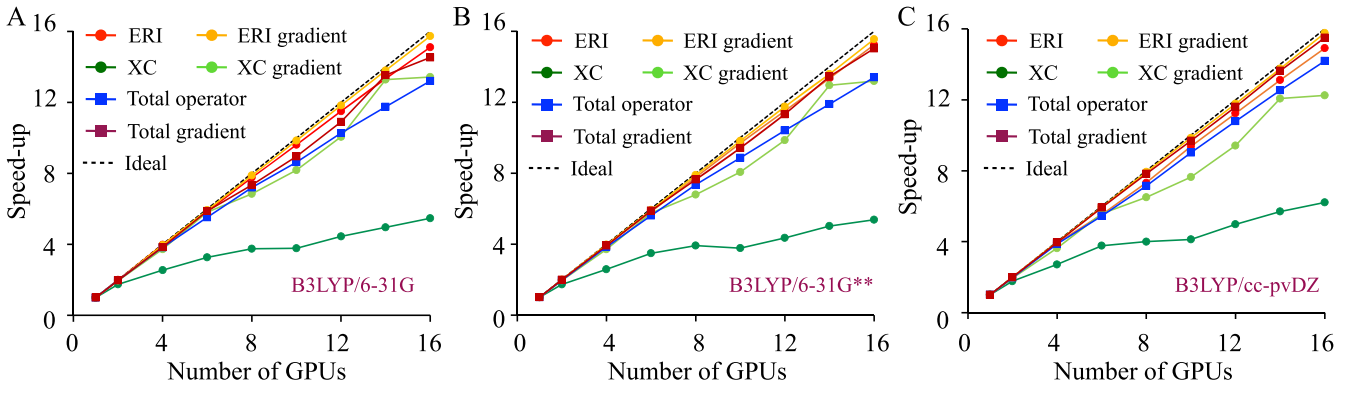



**Figure 4.** Speed-up of ERI, XC, and their gradient calculations for olestra at the B3LYP/6-31G (A), B3LYP/6-31G** (B), and B3LYP/cc-pVDZ (C) levels of theory on up to four GPU nodes. Each node consists of four NVIDIA V100-SXM2 type GPUs, 2 Intel Xeon Gold 6248 CPUs (2.50 GHz), and 374 GB memory per node. Total operator speed-up includes the sum of ERI, XC, and MPI communication time during 19 SCF iterations for B3LYP/6-31G, 18 SCF iterations for B3LYP/6-31G**, and 32 SCF iterations for B3LYP/cc-pVDZ. Total gradient speed-up includes the time for ERI and XC gradient computation and MPI communication.



https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966

**Table 1. Wall Times[a] in Seconds for ERI, XC Potential, ERI Gradient, and XC Gradient Tasks on Olestra (453 Atoms) at Different Levels of Theory on up to 4 GPU Nodes[b]**

| | B3LYP/6-31G[d] | | | | B3LYP/6-31G**[e] | | | | B3LYP/cc-pVDZ[f] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | gradient | | | | gradient | | | | gradient | |
| GPUs | ERI[c] | XC[c] | ERI | XC | ERI[c] | XC[c] | ERI | XC | ERI[c] | XC[c] | ERI | XC |
| 1 | 510.0 | 30.0 | 198.7 | 187.6 | 1795.9 | 67.0 | 784.4 | 192.6 | 11783.9 | 278.1 | 3298.4 | 206.2 |
| 2 | 254.9 | 17.2 | 99.8 | 95.8 | 906.7 | 38.9 | 393.4 | 98.7 | 5941.3 | 157.0 | 1652.9 | 106.1 |
| 4 | 128.1 | 11.8 | 50.2 | 50.4 | 457.3 | 26.0 | 197.3 | 52.0 | 3004.6 | 102.5 | 829.5 | 56.7 |
| 6 | 87.1 | 9.2 | 33.5 | 32.3 | 307.7 | 19.2 | 132.4 | 33.5 | 2131.8 | 73.8 | 554.3 | 37.0 |
| 8 | 65.7 | 8.0 | 25.2 | 27.4 | 230.8 | 17.1 | 99.4 | 28.4 | 1609.9 | 69.5 | 416.0 | 31.7 |
| 10 | 53.1 | 7.9 | 20.2 | 22.9 | 185.3 | 17.8 | 79.8 | 23.9 | 1257.4 | 67.4 | 334.8 | 27.0 |
| 12 | 44.3 | 6.7 | 16.8 | 18.7 | 156.1 | 15.4 | 66.8 | 19.5 | 1049.1 | 55.9 | 280.2 | 21.9 |
| 14 | 38.2 | 6.0 | 14.4 | 14.1 | 134.6 | 13.4 | 57.8 | 14.9 | 898.7 | 48.8 | 239.9 | 17.1 |
| 16 | 33.8 | 5.5 | 12.6 | 14.0 | 117.8 | 12.5 | 50.4 | 14.6 | 790.4 | 44.7 | 209.2 | 16.8 |

[a]Excluding MPI communication times. [b]Each node has 4 NVIDIA V100-SXM2 type GPUs (32 GB), 2 Intel Xeon Gold 6248 CPUs (2.50 GHz), and 374 GB memory. [c]Reported ERI and XC times are the total of 19 iterations for B3LYP/6-31G, 18 iterations for B3LYP/6-31G**, and 32 iterations for B3LYP/cc-pVDZ. [d]2131/4962 contracted/primitive functions. [e]4015/6846 contracted/primitive functions. [f]4015/9224 contracted/primitive functions.

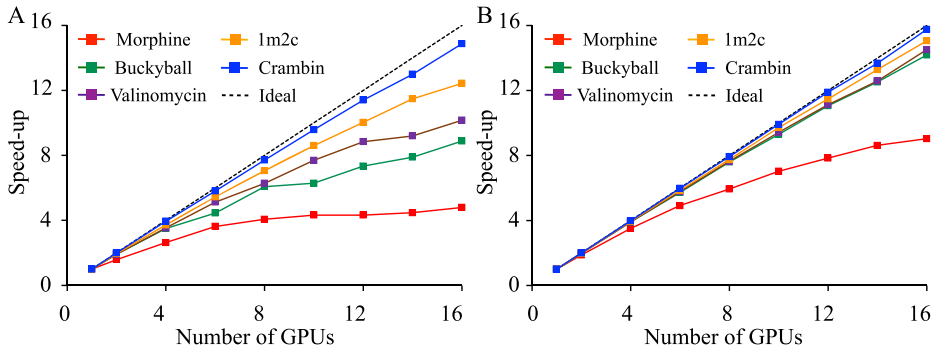



**Figure 5.** Speed-up for (A) Kohn−Sham operator formation during second SCF iteration (includes ERI, XC, and MPI communication time), and (B) gradient computation (includes ERI, XC gradient, and MPI communication time) of morphine (40 atoms, 410 basis functions), buckminsterfullerene (buckyball, 60 atoms and 900 basis functions), valinomycin (168 atoms, 1620 basis functions), α-contoxin (1m2c, 220 atoms and 2276 basis functions), crambin (642 atoms, 6504 basis functions) gradient calculations at B3LYP/6-31G** on up to four GPU nodes. Each node consists of four NVIDIA V100-SXM2 type GPUs, 2 Intel Xeon Gold 6248 CPU (2.50 GHz) and 374 GB memory per node.

using B3LYP with the 6-31G, 6-31G** and cc-pVDZ basis sets. The corresponding wall times are reported in Table 1, load balancing and MPI communication times are reported in Table S4, and the parallel efficiencies (calculated as $1/n \times T(\text{serial})/T(n) \times 100$) are reported in Tables S6−S8. MPI communication time is excluded from the timings in Table 1 because the partial Kohn−Sham matrix, and gradient vector that are sent from the workers to the root task contain both ERI and XC contributions. It is, however, included in the total speedups for Kohn−Sham operator and gradient vector formation in Figure 4. Linear algebra operations are currently executed on a single GPU and are reported in section S3. Three key pieces of information can be immediately obtained from these data. First, the ERI and ERI gradient calculations display near-linear strong scaling and high parallel efficiency in all cases, suggesting that the implemented load balancing scheme is effective. Second, the XC tasks demonstrate a lower, nonlinear scaling in speed-up despite the fact that their load balancing remains as impressive as that for the ERIs (see Figure S2). The parallel efficiency for the XC potential diminishes with increasing number of GPUs; but remains high for the XC gradient computation. Careful examination of device kernels using NVIDIA profiler tools revealed that the performance of all ERI kernels is limited by register availability and only a single thread block can reside on a streaming multiprocessor at a given time. With an increasing number of GPUs, more streaming multiprocessors are available for the computation resulting in the observed near-linear strong scaling. In contrast, the performance of the XC potential and energy gradient kernels are limited by global memory transactions, while the grid weight gradient kernel, which dominates the XC gradient time, is register bound. The reduced kernel efficiency in spite of having a balanced workload can be explained by GPU starvation. As mentioned previously, the parallelism of the XC computation is achieved by assigning numerical grid points to threads. In the presence of sufficient active warps, such as is the case for 1 or 2 GPUs, better latency hiding can be obtained by executing compute operations during the loading of memory and storage. However, achieving such hiding becomes difficult with more GPUs since the workload becomes lighter. The third piece of information from the first set of benchmarks is that the near-linear strong scaling of the total performance for complete Kohn−Sham operator formation and gradient calculations remains largely unaffected by the lower, nonlinear scaling of the XC potential and energy gradient tasks. The total parallel efficiency in all three test cases remains greater than 82% on up to 16 GPUs. This occurs because XC tasks represent only a small fraction of the total time. In all cases, the load balancing times were less than 2 s, and MPI communication times were


https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966

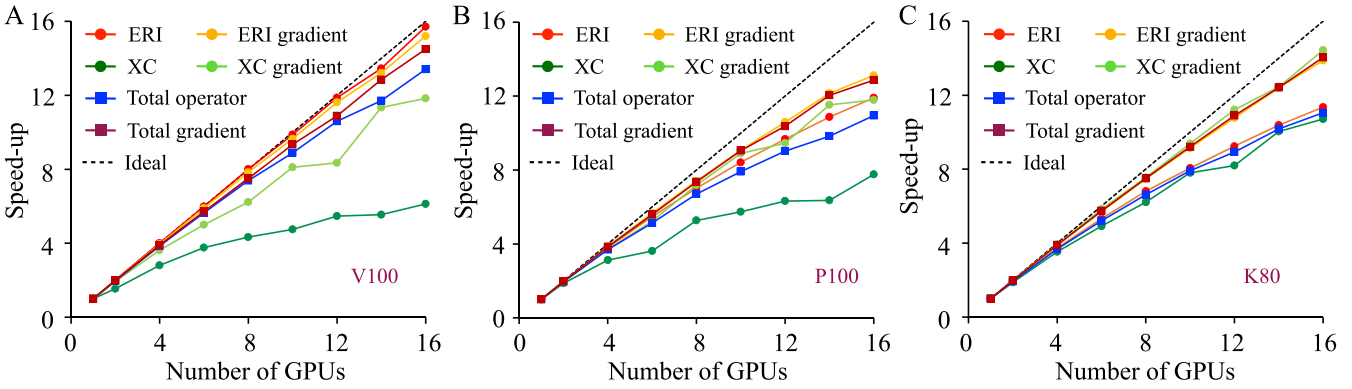



**Figure 6.** Speed-up of ERI, XC, and their gradient calculations for a water cluster (270 atoms, 2250 basis functions) at PBE0/def2-SVP level of theory on up to 4 GPU nodes of V100 (A), P100 (B), and K80 (C). Each V100 node comprises 4 NVIDIA V100-SXM2 type GPUs (32 GB), 2 Intel Xeon Gold 6248 CPUs (2.50 GHz), and 374 GB memory. P100 nodes are equipped with 4 NVIDIA P100 type GPUs (16 GB), 2 Intel Xeon E5−2680 v4 CPUs (2.4 GHz), and 128 GB memory. Each K80 node has 4 NVIDIA K80 type GPUs (12 GB), 2 Intel Xeon E5−2680 v3 CPUs (2.5 GHz), and 128 GB memory. Total operator speed-up includes the sum of ERI, XC, and MPI communication time during 13 SCF iterations. Total gradient speed-up includes the time for ERI and XC gradient computation and MPI communication.

**Table 2. Wall Times[a] in Seconds for ERI, XC Potential, ERI Gradient, and XC Gradient Tasks of a Water Cluster (270 Atoms, 2250 Basis Functions) Single Point Energy Plus Gradient Calculation at PBE0/def2-SVP Level of Theory (2250/3510 Contracted/Primitive Functions) on up to 4 Nodes with V100, P100, and K80 GPUs[b]**

| | V100 | | | | P100 | | | | K80 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | gradient | | | | gradient | | | | gradient | |
| GPUs | ERI[c] | XC[c] | ERI | XC | ERI[c] | XC[c] | ERI | XC | ERI[c] | XC[c] | ERI | XC |
| 1 | 405.5 | 30.8 | 232.0 | 48.1 | 645.2 | 80.1 | 461.0 | 96.5 | 2382.7 | 565.6 | 1697.8 | 723.5 |
| 2 | 202.0 | 20.0 | 116.3 | 24.9 | 327.7 | 42.7 | 233.2 | 49.2 | 1224.5 | 302.0 | 859.1 | 367.2 |
| 4 | 101.3 | 11.0 | 58.6 | 13.3 | 169.6 | 25.6 | 119.0 | 26.1 | 638.6 | 160.1 | 436.9 | 184.5 |
| 6 | 67.7 | 8.2 | 39.3 | 9.6 | 117.9 | 22.1 | 81.3 | 18.4 | 448.1 | 115.1 | 297.6 | 124.1 |
| 8 | 50.7 | 7.1 | 29.6 | 7.7 | 92.0 | 15.2 | 62.3 | 13.4 | 350.3 | 91.1 | 227.1 | 95.9 |
| 10 | 41.1 | 6.5 | 24.0 | 5.9 | 76.7 | 14.0 | 50.8 | 10.8 | 295.9 | 72.5 | 185.8 | 77.1 |
| 12 | 34.2 | 5.6 | 20.0 | 5.8 | 66.7 | 12.7 | 43.6 | 10.2 | 258.0 | 69.0 | 157.7 | 64.5 |
| 14 | 30.2 | 5.5 | 17.6 | 4.2 | 59.4 | 12.6 | 37.9 | 8.4 | 229.5 | 56.4 | 137.0 | 58.1 |
| 16 | 25.8 | 5.0 | 15.3 | 4.1 | 54.2 | 10.3 | 35.1 | 8.2 | 209.9 | 52.8 | 122.3 | 50.1 |

[a]Excluding MPI communication times. [b]Each V100 node comprises 4 NVIDIA V100-SXM2 type GPUs (32 GB), 2 Intel Xeon Gold 6248 CPUs (2.50 GHz), and 374 GB memory. P100 nodes are equipped with 4 NVIDIA P100 type GPUs (16 GB), 2 Intel Xeon E5−2680 v4 CPUs (2.4 GHz), and 128 GB memory. Each K80 node has 4 NVIDIA K80 type GPUs (12 GB), 2 Intel Xeon E5−2680 v3 CPUs (2.5 GHz), and 128 GB memory. [c]Reported ERI and XC times are the total of 13 SCF iterations.

small (see Table S4) having no significant impact on the total operator and gradient formation speed-ups. It is worth noting, however, that the linear algebra operations can become significant in comparison to the Kohn−Sham operator formation, consuming up to 29% of the total SCF time in the worst case of the 6-31G basis set when using 16 GPUs.

In Figure 5, we report the total speed-up of B3LYP/6-31G** Kohn−Sham operator formation and gradient calculation times for five molecular systems of different size (see Figure 3 for structures). The parallel efficiencies and combined total times are reported in Tables S9, S10, S15, and S16 respectively. Times for linear algebra operations are also given in section S6. As anticipated, the larger systems display better scaling with high parallel efficiency. For instance, crambin and 1m2c examples show efficiencies greater than 78% for operator build and 94% for gradient calculation on 16 computing ranks. In contrast, for the smallest example morphine, the operator and gradient formation efficiencies drop down to ∼30% and ∼56% on 16 ranks. Such a performance decrease is expected since the workload becomes lighter in the presence of more computation resources.

**4.2. Performance on Different Microarchitectures.** For all the benchmarks presented so far, we have used NVIDIA V100-SXM2 type GPUs. It is also necessary to document the performance of the QUICK multi-GPU version on other widely used data center cards. For this purpose, we selected 4 NVIDIA P100 and K80 (belonging to Pascal and Kepler microarchitectures respectively) GPU nodes from the SDSC Comet cluster. In Figure 6 and Table 2, we report the speed-ups and wall times for gradient computation for a cluster containing 90 water molecules at the PBE0/def2-SVP level of theory. The load balancing and MPI communication times are reported in Table S5. The parallel efficiencies are reported in Tables S11−S13. Times for linear algebra operations are also reported in section S3. At first glance, one notices the highest single GPU performance for all tasks on the V100 and the lowest on the K80. This trend is expected and consistent with the reported peak FP64 compute power (7.8, 5.3, and 2.9 TFLOPS for the V100, P100, and K80, respectively) and memory bandwidths (900, 780, and 480 GB/s for the V100, P100, and K80, respectively) for each device.[69−71] The best scaling for ERI and ERI gradient calculations is also observed



https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966

on the V100 platform. The associated parallel efficiency is greater than 94%. Such scaling slightly diminishes on the P100 and K80 platforms; however, the parallel efficiency remains above 70%. Exploring for potential performance improvement, we reevaluated the thread walking strategies for the latter platforms (see Table S2). The results suggested that circular thread walking is the most suitable as for V100s. For XC tasks, the best scaling is observed on the K80 platform with parallel efficiencies >66% and >89% for potential and gradient computations, respectively. This is followed by the efficiencies of P100 and then the V100. The different scaling of ERI and XC tasks on the three platforms must be due to their significant architectural differences.[69,70,72] The highest overall parallel efficiency (>89%) is achieved on the V100 class of GPUs. In addition to the above platforms, we benchmarked the QUICK multi-GPU version on a single NVIDIA DGX A100 node[73] equipped with eight A100 type GPUs (belonging to recent Ampere microarchitecture).[74] Owing to the high peak FP64 compute power, ERI and ERI gradient calculations on a single A100 are much faster in comparison to V100 (see Table S17). In contrast, XC and XC gradient times remain substantially the same. The observed scaling and parallel efficiencies are similar to that of eight V100s (see Figure S3 and Table S14). On all platforms, the load balancing and MPI communication times remain considerably small.

## 5. CONCLUSIONS

We have reported the details of a MPI parallel multi-GPU *ab initio* HF/DFT implementation of the QUICK quantum chemical package. Our implementation features static ERI and XC load balancing schemes. Dynamic load balancing is employed in the XC gradient calculations. Benchmarking against the single GPU version on up to 16 GPUs demonstrated near-linear strong scaling behavior for ERIs and ERI gradients and lower, nonlinear scaling for the XC and XC gradients resulting in excellent aggregated parallel efficiencies above 82% for Kohn−Sham operator build and above 91% for gradient computation. Similar scaling is observed on A100, P100, and K80 platforms. The associated parallel efficiencies for operator and gradient calculation were always greater than 68% and 80%, respectively, paving the way for large-scale *ab initio* electronic structure calculations. The benchmarks in the current study were limited to four nodes, which is the maximum allowed per user at the SDSC. The performance scaling on more compute nodes would be informative. We recommend NVIDIA V100 or A100 data center GPUs for the latest QUICK version (v21.03).

The profiling of the ERI and XC kernels has indicated room for potential improvement. For the ERI kernels, the current bottleneck is the register availability. Reordering load and store procedures to reuse available registers and reimplementing large device kernels into smaller kernels may lead to favorable performance on both serial and multi-GPU versions. For the XC kernels, the memory efficiency should be enhanced. In this context, increasing the register and shared memory usage may be viable strategies. Furthermore, the overall performance of single point energy calculations can be improved by utilizing multiple GPUs for linear algebra operations, which are currently performed using a single GPU.

Finally, we recently integrated the QUICK multi-GPU version as a library into the development version of the AMBER molecular dynamics package[75] enabling GPU capable quantum mechanics/molecular mechanics (QM/MM) simulations.[76] QUICK version 21.03 can be downloaded from https://github.com/merzlab/QUICK under the Mozilla public license free of charge, and will also be available as part of AmberTools version 21, freely available.[75]

## ■ ASSOCIATED CONTENT

### Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jctc.1c00145.

Figures and tables with benchmark data for ERI thread walking strategies, XC load balancing strategies, load balancing and MPI communication times, parallel efficiencies, load balancing, and performance on different GPU types (PDF)

QUICK input files with Cartesian coordinates of the benchmark molecules (ZIP)

## ■ AUTHOR INFORMATION


### Corresponding Authors

**Kenneth M. Merz, Jr.** − *Department of Chemistry and Department of Biochemistry and Molecular Biology, Michigan State University, East Lansing, Michigan 48824-1322, United States;* orcid.org/0000-0001-9139-5893; Email: merz@chemistry.msu.edu

**Andreas W. Götz** − *San Diego Supercomputer Center, University of California San Diego, La Jolla, California 92093-0505, United States;* orcid.org/0000-0002-8048-6906; Email: agoetz@sdsc.edu

### Authors

**Madushanka Manathunga** − *Department of Chemistry and Department of Biochemistry and Molecular Biology, Michigan State University, East Lansing, Michigan 48824-1322, United States;* orcid.org/0000-0002-3594-8112

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

**Vinícius Wilian D. Cruzeiro** − *San Diego Supercomputer Center, University of California San Diego, La Jolla, California 92093-0505, United States; Department of Chemistry and Biochemistry, University of California San Diego, La Jolla, California 92093, United States;* orcid.org/0000-0002-4739-5447

**Yipu Miao** − *Facebook, Menlo Park, California 94025, United States*

**Dawei Mu** − *National Center for Supercomputing Applications, University of Illinois at Urbana−Champaign, Urbana, Illinois 61801, United States*

**Kamesh Arumugam** − *NVIDIA Corporation, Santa Clara, California 95051, United States*

**Kristopher Keipert** − *NVIDIA Corporation, Santa Clara, California 95051, United States*

**Hasan Metin Aktulga** − *Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan 48824-1322, United States*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jctc.1c00145


### Notes

The authors declare no competing financial interest.

## ■ ACKNOWLEDGMENTS

We thank Dmitry Pekurovsky from SDSC for improving our one-electron integral code, and Scott Le Grand and Kurt O'hearn for their useful comments on technical aspects of our GPU code. M.M. and A.G. thank SDSC for granted computer time and Mary Thomas, Susan Rathbun, Julia Levites, and the other organizers of the SDSC 2020 GPU Hackathon. M.M. and K.M. are grateful to the Department of Chemistry and Biochemistry and high-performance computer center (iCER HPCC) at the Michigan State University. We also thank Jonathan Lefman and NVIDIA corporation for granting access to the NVIDIA PSG cluster. This research was supported by the National Science Foundation Grant OAC-1835144. 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 to A.G.).

## ■ REFERENCES

(1) Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, large minibatch SGD: Training imagenet in 1 h. 2018, arXiv:1706.02677v2, arXiv.org e-Print archive. https://arxiv.org/abs/1706.02677v2 (accessed 2021-02-05).

(2) Fuhrer, O.; Chadha, T.; Hoefler, T.; Kwasniewski, G.; Lapillonne, X.; Leutwyler, D.; Lüthi, D.; Osuna, C.; Schär, C.; Schulthess, T. C.; et al. Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0. *Geosci. Model Dev.* **2018**, *11*, 1665−1681.

(3) Harvey, M. J.; Giupponi, G.; De Fabritiis, G. ACEMD: Accelerating biomolecular dynamics in the microsecond time scale. *J. Chem. Theory Comput.* **2009**, *5*, 1632−1639.

(4) Götz, A. W.; Williamson, M. J.; Xu, D.; Poole, D.; Le Grand, S.; Walker, R. C. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. *J. Chem. Theory Comput.* **2012**, *8*, 1542−1555.

(5) Salomon-Ferrer, R.; Götz, A. W.; Poole, D.; Le Grand, S.; Walker, R. C. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. *J. Chem. Theory Comput.* **2013**, *9*, 3878−3888.

(6) Phillips, J. C.; Hardy, D. J.; Maia, J. D. C.; Stone, J. E.; Ribeiro, J. V.; Bernardi, R. C.; Buch, R.; Fiorin, G.; Hénin, J.; Jiang, W.; et al. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *J. Chem. Phys.* **2020**, *153*, 044130−044130.

(7) Phillips, J. C.; Sun, Y.; Jain, N.; Bohm, E. J.; Kale, L. V. Mapping to Irregular Torus Topologies and Other Techniques for Petascale Biomolecular Simulation. In SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, Supercomputing Conference, New Orleans, LA, November 16−21, 2014, pp 81−91.

(8) Kutzner, C.; Páll, S.; Fechner, M.; Esztermann, A.; De Groot, B. L.; Grubmüller, H. Best bang for your buck: GPU nodes for GROMACS biomolecular simulations. *J. Comput. Chem.* **2015**, *36*, 1990−2008.

(9) Kutzner, C.; Páll, S.; Fechner, M.; Esztermann, A.; de Groot, B. L.; Grubmüller, H. More bang for your buck: Improved use of GPU nodes for GROMACS 2018. *J. Comput. Chem.* **2019**, *40*, 2418−2431.

(10) Eastman, P.; Swails, J.; Chodera, J. D.; McGibbon, R. T.; Zhao, Y.; Beauchamp, K. A.; Wang, L. P.; Simmonett, A. C.; Harrigan, M. P.; Stern, C. D.; et al. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS Comput. Biol.* **2017**, *13*, e1005659−e1005659.

(11) Brown, W. M.; Wang, P.; Plimpton, S. J.; Tharrington, A. N. Implementing molecular dynamics on hybrid high performance computers - Short range forces. *Comput. Phys. Commun.* **2011**, *182*, 898−911.

(12) Brown, W. M.; Kohlmeyer, A.; Plimpton, S. J.; Tharrington, A. N. Implementing molecular dynamics on hybrid high performance computers - Particle-particle particle-mesh. *Comput. Phys. Commun.* **2012**, *183*, 449−459.

(13) Adjoua, O.; Lagardère, L.; Jolly, L.-H.; Durocher, A.; Very, T.; Dupays, I.; Wang, Z.; Inizan, T. J.; Célerse, F.; Ren, P.; et al. Tinker-HP: Accelerating Molecular Dynamics Simulations of Large Complex Systems with Advanced Point Dipole Polarizable Force Fields Using GPUs and Multi-GPU Systems. *J. Chem. Theory Comput.* **2021**, *17*, 2034−2053.

(14) Anderson, J. A.; Lorenz, C. D.; Travesset, A. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comput. Phys.* **2008**, *227*, 5342−5359.

(15) Anderson, J. A.; Glaser, J.; Glotzer, S. C. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Comput. Mater. Sci.* **2020**, *173*, 109363−109363.

(16) Glaser, J.; Schwendeman, P. S.; Anderson, J. A.; Glotzer, S. C. Unified memory in HOOMD-blue improves node-level strong scaling. *Comput. Mater. Sci.* **2020**, *173*, 109359−109359.

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

(18) 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.

(19) 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.

(20) Shi, G.; Kindratenko, V.; Ufimtsev, I.; Martinez, T. Direct self-consistent field computations on GPU clusters. 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 19−23 April 2010; 2010; pp 1−8.

(21) 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.; et al. TeraChem: Accelerating electronic structure and ab initio molecular dynamics with graphical processing units. *J. Chem. Phys.* **2020**, *152*, 224110−224110.

(22) 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, J. W.; et al. 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.

(23) Genovese, L.; Videau, B.; Ospici, M.; Deutsch, T.; Goedecker, S.; Méhaut, J. F. Daubechies wavelets for high performance electronic structure calculations: The BigDFT project. *C. R. Mec.* **2011**, *339*, 149−164.

(24) 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.

(25) Andrade, X.; Alberdi-Rodriguez, J.; Strubbe, D. A.; Oliveira, M. J. T.; Nogueira, F.; Castro, A.; Muguerza, J.; Arruabarrena, A.; Louie, S. G.; Aspuru-Guzik, A.; et al. Time-dependent density-functional theory in massively parallel computer architectures: the octopus project. *J. Phys.: Condens. Matter* **2012**, *24*, 233202−233202.

(26) Hacene, M.; Anciaux-Sedrakian, A.; Rozanska, X.; Klahr, D.; Guignon, T.; Fleurat-Lessard, P. Accelerating VASP electronic structure calculations using graphic processing units. *J. Comput. Chem.* **2012**, *33*, 2581−2589.

(27) Hutchinson, M.; Widom, M. VASP on a GPU: Application to exact-exchange calculations of the stability of elemental boron. *Comput. Phys. Commun.* **2012**, *183*, 1422−1426.

(28) Maintz, S.; Eck, B.; Dronskowski, R. Speeding up plane-wave electronic-structure calculations using graphics-processing units. *Comput. Phys. Commun.* **2011**, *182*, 1421−1427.



https://doi.org/10.1021/acs.jctc.1c00145
*J. Chem. Theory Comput.* 2021, 17, 3955−3966

(29) Hakala, S.; Havu, V.; Enkovaara, J.; Nieminen, R. *Parallel electronic structure calculations using multiple graphics processing units (GPUs)*; Springer: Berlin, Heidelberg, 2013; pp 63−76.
(30) Yan, J.; Li, L.; O'Grady, C. Graphics Processing Unit acceleration of the Random Phase Approximation in the projector augmented wave method. *Comput. Phys. Commun.* **2013**, *184*, 2728−2733.
(31) Jia, W.; Cao, Z.; Wang, L.; Fu, J.; Chi, X.; Gao, W.; Wang, L. W. The analysis of a plane wave pseudopotential density functional theory code on a GPU machine. *Comput. Phys. Commun.* **2013**, *184*, 9−18.
(32) Jia, W.; Fu, J.; Cao, Z.; Wang, L.; Chi, X.; Gao, W.; Wang, L. W. Fast plane wave density functional theory molecular dynamics calculations on multi-GPU machines. *J. Comput. Phys.* **2013**, *251*, 102−115.
(33) Romero, J.; Phillips, E.; Ruetsch, G.; Fatica, M.; Spiga, F.; Giannozzi, P. A Performance Study of Quantum ESPRESSO's PWscf Code on Multi-core and GPU Systems. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*; Jarvis, S., Wright, S., Hammond, S., Eds.; Springer International Publishing: 2018; Vol. *10724*, pp 67−87.
(34) Giannozzi, P.; Baseggio, O.; Bonfà, P.; Brunato, D.; Car, R.; Carnimeo, I.; Cavazzoni, C.; De Gironcoli, S.; Delugas, P.; Ferrari Ruffino, F.; et al. Quantum ESPRESSO toward the exascale. *J. Chem. Phys.* **2020**, *152*, 154105−154105.
(35) Kang, S.; Woo, J.; Kim, J.; Kim, H.; Kim, Y.; Lim, J.; Choi, S.; Kim, W. Y. ACE-Molecule: An open-source real-space quantum chemistry package. *J. Chem. Phys.* **2020**, *152*, 124110−124110.
(36) Wilkinson, K.; Skylaris, C.-K. Porting ONETEP to graphical processing unit-based coprocessors. 1. FFT box operations. *J. Comput. Chem.* **2013**, *34*, 2446−2459.
(37) Kühne, T. D.; Iannuzzi, M.; Del Ben, M.; Rybkin, V. V.; Seewald, P.; Stein, F.; Laino, T.; Khaliullin, R. Z.; Schütt, O.; Schiffmann, F.; et al. CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations. *J. Chem. Phys.* **2020**, *152*, 194103−194103.
(38) Huhn, W. P.; Lange, B.; Yu, V. W. z.; Yoon, M.; Blum, V. GPU acceleration of all-electron electronic structure theory using localized numeric atom-centered basis functions. *Comput. Phys. Commun.* **2020**, *254*, 107314−107314.
(39) Jia, W.; Wang, J.; Chi, X.; Wang, L. W. GPU implementation of the linear scaling three dimensional fragment method for large scale electronic structure calculations. *Comput. Phys. Commun.* **2017**, *211*, 8−15.
(40) 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.
(41) 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.; et al. Recent developments in the general atomic and molecular electronic structure system. *J. Chem. Phys.* **2020**, *152*, 154102−154102.
(42) DePrince, A. E.; Hammond, J. R. Coupled Cluster Theory on Graphics Processing Units I. The Coupled Cluster Doubles Method. *J. Chem. Theory Comput.* **2011**, *7*, 1287−1295.
(43) DePrince, A. E.; 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.
(44) 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.
(45) Bhaskaran-Nair, K.; Ma, W.; Krishnamoorthy, S.; Villa, O.; Van Dam, H. J. J.; Aprà, E.; Kowalski, K. Noniterative multireference coupled cluster methods on heterogeneous CPU-GPU systems. *J. Chem. Theory Comput.* **2013**, *9*, 1949−1957.
(46) Ma, W.; Krishnamoorthy, S.; Villa, O.; Kowalski, K.; Agrawal, G. Optimizing tensor contraction expressions for hybrid CPU-GPU execution. *Clust. Comput.* **2013**, *16*, 131−155.
(47) Fales, B. S.; Curtis, E. R.; Johnson, K. G.; Lahana, D.; Seritan, S.; Wang, Y.; Weir, H.; Martínez, 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.
(48) 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−e25894.
(49) Peng, C.; Lewis, C. A.; Wang, X.; Clement, M. C.; Pierce, K.; Rishi, V.; Pavošević, F.; Slattery, S.; Zhang, J.; Teke, N.; et al. Massively Parallel Quantum Chemistry: A high-performance research platform for electronic structure. *J. Chem. Phys.* **2020**, *153*, 044120−044120.
(50) 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*, 184105−184105.
(51) Aprà, 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.; et al. NWChem: Past, present, and future. *J. Chem. Phys.* **2020**, *152*, 184102−184102.
(52) 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*, 581058−581058.
(53) Kowalski, K.; Bair, R.; Bauman, N. P.; Boschen, J. S.; Bylaska, E. J.; Daily, J.; de Jong, W. A.; Dunning, T.; Govind, N.; Harrison, R. J. From NWChem to NWChemEx: Evolving with the Computational Chemistry Landscape. *Chem. Rev.* **2021**, *121*, 4962.
(54) Gawande, N.; Kowalski, K.; Palmer, B.; Krishnamoorthy, S.; Apra, E.; Manzano, J.; Amatya, V.; Crawford, J. Accelerating the Global Arrays ComEx Runtime Using Multiple Progress Ranks. 2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI), Denver, Colorado, 17 Nov. 2019, pp 29−38.
(55) Kussmann, J.; Ochsenfeld, C. Hybrid CPU/GPU Integral Engine for Strong-Scaling Ab Initio Methods. *J. Chem. Theory Comput.* **2017**, *13*, 3153−3159.
(56) 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.
(57) 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.
(58) Miao, Y.; Merz, 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.
(59) Miao, Y.; Merz, 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.
(60) Manathunga, M.; Miao, Y.; Mu, D.; Götz, A. W.; Merz, K. M. Parallel Implementation of Density Functional Theory Methods in the Quantum Interaction Computational Kernel Program. *J. Chem. Theory Comput.* **2020**, *16*, 4315−4326.
(61) Gabriel, E.; Fagg, G. E.; Bosilca, G.; Angskun, T.; Dongarra, J. J.; Squyres, J. M.; Sahay, V.; Kambadur, P.; Barrett, B.; Lumsdaine, A.; et al. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. *Recent Advances in Parallel Virtual Machine and Message Passing Interface* **2004**, *Vol. 3241*, 97−104.
(62) Obara, S.; Saika, A. Efficient recursive computation of molecular integrals over Cartesian Gaussian functions. *J. Chem. Phys.* **1986**, *84*, 3963−3974.


https://doi.org/10.1021/acs.jctc.1c00145
J. Chem. Theory Comput. 2021, 17, 3955−3966

(63) 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.
(64) Pople, J. A.; Gill, P. M. W.; Johnson, B. G. Kohn—Sham density-functional theory within a finite basis set. *Chem. Phys. Lett.* **1992**, *199*, 557−560.
(65) Barca, G.; Poole, D.; Vallejo, J.; Alkan, M.; Bertoni, C.; Rendell, A.; Gordon, M. *Scaling the Hartree-Fock Matrix Build on Summit*. SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, Los Alamitos, CA, USA, 9−19 Nov. 2020; IEEE Computer Society: Los Alamitos, CA, USA, 2020; pp 1−14.
(66) Cheng, J.; Grossman, M.; McKercher, T. *Professional CUDA C Programming*; Wrox Press Ltd.: 2013; Vol. 53, pp 1689−1699.
(67) Han, J.; Sharma, B. *Learn CUDA Programming: A beginner's guide to GPU programming and parallel computing with CUDA 10.x and C/C++*; Packt Publishing: 2019.
(68) Pacheco, P. *An Introduction to Parallel Programming*; Elsevier Science: 2011.
(69) NVIDIA. NVIDIA Tesla V100 GPU Architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf (accessed 2020-02-25).
(70) NVIDIA. NVIDIA Tesla P100. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf (accessed 2021-01-09).
(71) NVIDIA. Tesla K80 | NVIDIA. https://www.nvidia.com/en-gb/data-center/tesla-k80/ (accessed 2021-01-09).
(72) Microway In-Depth Comparison of NVIDIA Tesla Kepler GPU Accelerators | Microway. https://www.microway.com/knowledge-center-articles/in-depth-comparison-of-nvidia-tesla-kepler-gpu-accelerators/ (accessed 2021-01-09).
(73) NVIDIA. NVIDIA DGX A100 | DATA SHEET | MAY20. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf (accessed 2021-01-09).
(74) NVIDIA. NVIDIA A100 Tensor Core GPU Architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf (accessed 2021-01-09).
(75) Case, D. A.; Belfon, K.; Ben-Shalom, I. Y.; Brozell, S. R.; Cerutti, D. S.; Cheatham, T. E.; Iii; Cruzeiro, V. W. D.; Darden, T. A.; Duke, R. E.; et al. *AMBER 2020*, University of California: San Francisco, CA, 2020.
(76) Cruzeiro, V. W. D.; Manathunga, M.; Merz, 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.


https://doi.org/10.1021/acs.jctc.1c00145
*J. Chem. Theory Comput.* 2021, 17, 3955−3966