# 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[‡,*]

[†]Department of Chemistry and Department of Biochemistry and Molecular Biology, Michigan State University, 578 S. Shaw Lane, East Lansing, Michigan 48824-1322, United States, [‡]San Diego Supercomputer Center, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093-0505, United States, [¥]Department of Chemistry and Biochemistry, University of California San Diego, La Jolla, CA 92093, United States, [#]Facebook, 1 Hacker Way, Menlo Park, California 94025, United States, [%]National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1205 W Clark St, Urbana, IL 61801, United States, [§]NVIDIA Corporation, Santa Clara, CA 95051, [¶]Department of Computer Science and Engineering, Michigan State University, 428 S. Shaw Lane, East Lansing, Michigan 48824-1322, United States.

* Corresponding authors: Kenneth M. Merz, Jr. (merz@chemistry.msu.edu), Andreas W. Götz (agoetz@sdsc.edu)

*KEYWORDS: QUICK, Graphics Processing Units, Quantum Chemistry Software, HF, DFT Package*

**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 4 GPU nodes, each containing 4 NVIDIA V100-SMX2 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 efficiencies remained above 86%. The accelerations on NVIDIA A100, P100 and K80 platforms also have realized parallel efficiencies higher than 74%, paving the way for large-scale *ab initio* electronic structure calculations.

## 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 multi-GPUs has been harnessed into a range of traditional computational chemistry tools,[3–12] however, only a handful of *ab initio* quantum chemical packages[9–13] are among them. 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. Perhaps the lack of multi-GPU capable quantum chemical packages is mainly due to the complexity of load balancing and performance tuning on GPU hardware. To fill this void, we have further improved our open source quantum chemical package called QUantum Interaction Computational Kernel (QUICK) software[14–16] by incorporating multi-GPU capabilities. QUICK is capable of performing efficient *ab initio* Hartree-Fock (HF) and density functional theory (DFT) energy and gradient calculations. For instance, the realized speed-ups for computing B3LYP energy and gradients of small to medium size molecular systems on a single NVIDIA V100 GPU were ca. 30 to 90-fold and 35 to 60-fold with respect to a Skylake CPU platform.[16] In QUICK, the most time-consuming tasks of HF/DFT calculations, electronic repulsion integral (ERI), exchange correlation potential (only in the case of DFT, XC) and their derivatives 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).[17,18] The XC contributions are calculated based on a scheme developed by Pople and co-workers.[19] In addition to computing the above quantities, assembling the Fock matrix and gradient vector are also done on the GPU.

Among the few publications found in the literature regarding the multi-GPU implementation of ERIs, Ufimtsev and Martinez's work[9] 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.[13] 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. A second, but a hybrid ERI engine, has been developed by Kussman and Ochsenfeld,[10] and quite recently, a fragmentation based Fock build algorithm with dynamic load balancing has been reported by Gordon and coworkers.[12] In the context of XC parallelization on multi-GPUs, Williams-Young *et al*.[11] has documented a three level parallelization scheme. In such a scheme, the load balancing is achieved by pre-estimating the FLOPs incurred by batches of grid points.

Our multi-GPU implementation consists of the following features. The message passing interface (MPI)[20] is used to set up the calculation and communicate between compute ranks 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,[19]

$$E = \sum_i^n \left(\psi_i| -\frac{1}{2}\nabla^2|\psi_i\right) - \sum_A^{nucl} Z_A \int \rho(r)|r - r_A|^{-1}dr + \frac{1}{2}\iint \rho(r_1)|r_1 - r_2|^{-1}\rho(r_2)dr_1dr_2 + \int f(\rho(r), \nabla\rho(r))dr \quad (1)$$

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 densities 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,

$$\rho = \sum_i^n |\psi_i|^2. \quad (2)$$

In practice, calculation of the energy using equation (1) requires expressing molecular orbitals and electron density in terms of atomic orbitals ($\phi_\mu$),

$$\psi_i = \sum_\mu^N C_{\mu i}\phi_\mu \,, \rho = \sum_\mu^N \sum_\nu^N \sum_i^n \left(C_{\mu i}\right) * C_{\nu i}\phi_\mu\phi_\nu = \sum_{\mu\nu} P_{\mu\nu}\phi_\mu\phi_\nu, \quad (3)$$

where $C_{\mu i}$ and $C_{\nu i}$ are molecular orbital coefficients and $P_{\mu\nu}$ is the density matrix. Substituting equations (3) into (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 ($K_{\mu\nu}$).

$$K_{\mu\nu} = H_{\mu\nu}^{core} + J_{\mu\nu} + K_{\mu\nu}^{XC}, \quad (5)$$

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

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

and $K_{\mu\nu}^{XC}$ is the XC potential contribution to the Kohn-Sham matrix expressed as,

$$K_{\mu\nu}^{XC} = \int \left[\frac{\partial f}{\partial \rho}\phi_\mu\phi_\nu + \left(3\frac{\partial f}{\partial \gamma}\nabla\rho\right).\nabla(\phi_\mu\phi_\nu)\right] dr\,. \quad (7)$$

Here $\gamma = |\nabla\rho|^2$. (8)

The computationally most expensive task in building the Kohn-Sham matrix is computing the ERIs required in equation (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.

$$(\mu\nu|\lambda\sigma) = \sum_{pqrs} C_{\mu a}C_{\nu b}C_{\lambda c}C_{\sigma d}[ab|cd] \quad (9)$$

Primitive ERIs can be computed and assembled into contracted ERIs using an established algorithm such as OSHGP.[17,18] The second most expensive contribution for constructing the Kohn-Sham matrix is calculating the XC potential. Due to the complexity of XC functionals, this quantity is obtained numerically, involving the formation of a quadrature grid where 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 equation (1) by differentiating with respect to nuclear coordinates.

$$\nabla_A E = \sum_{\mu\nu}^N P_{\mu\nu}\left(\nabla_A H_{\mu\nu}^{core}\right) + \frac{1}{2}\sum_{\mu\nu\lambda\sigma}^N P_{\mu\nu}P_{\lambda\sigma}\nabla_A(\mu\nu|\lambda\sigma) - \sum_{\mu\nu}^N W_{\mu\nu}\left(\nabla_A S_{\mu\nu}\right) - 4\sum_\mu'\sum_\nu^N P_{\mu\nu}\int\left[\frac{\partial f}{\partial \rho}\phi_\nu\nabla\phi_\mu + X_{\mu\nu}\left(\frac{\partial f}{\partial \gamma}\nabla\rho\right)\right] dr. \quad (10)$$

Here $S_{\mu\nu}$ is the overlap matrix, $W_{\mu\nu}$ is the energy weighted density matrix and $X_{\mu\nu}$ is a matrix element given by,

$$X_{\mu\nu} = \phi_\nu\nabla(\nabla\phi_\mu)^t + (\nabla\phi_\mu)(\nabla\phi_\mu)^t. \quad (11)$$

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

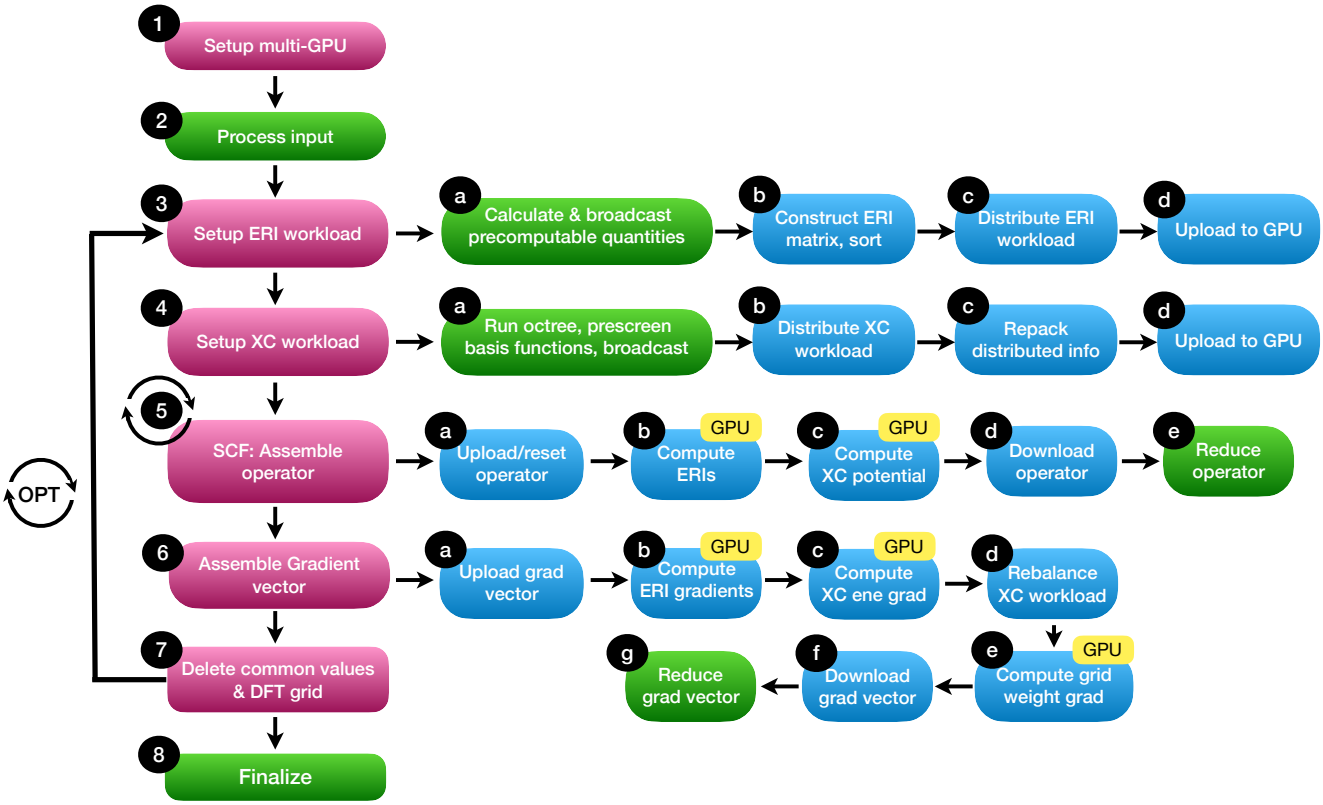


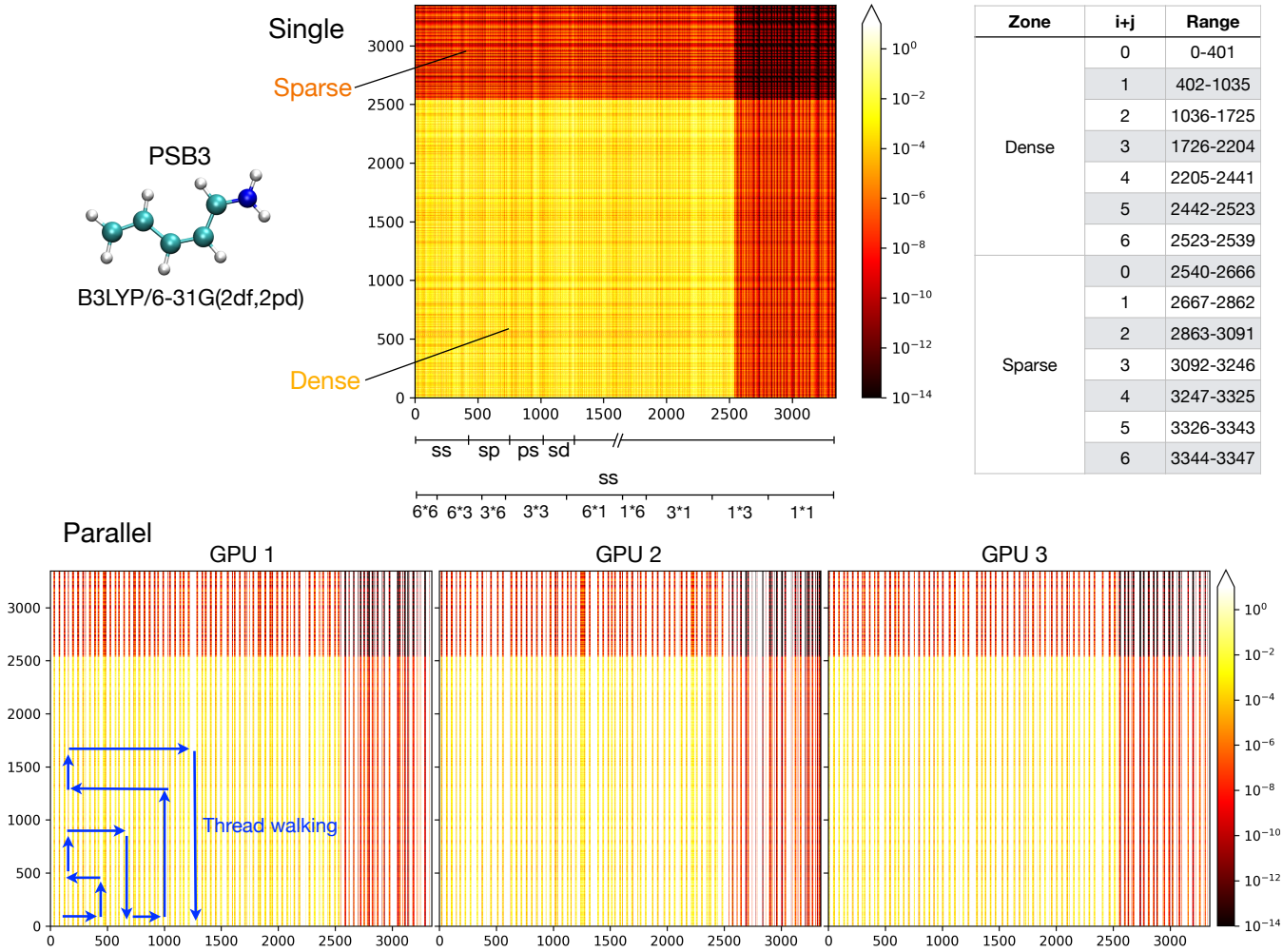

**Fig. 1.** Flowchart depicting the multi-GPU workflow of a DFT geometry optimization calculation. Major and sub steps are denoted by purple and light blue color boxes, respectively. Steps indicated by green boxes are performed only on the root compute rank. OPT denotes optimization and circle arrows 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.

## 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.[21] 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 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.[21] The main type, called global memory, is the largest and accessible to threads located on all streaming multiprocessors. Typically, 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 usage of these memory spaces is essential to write a memory efficient GPU application.

For setting up a multi-GPU program, at least two main options are at one's disposal.[22] The first is to use the CUDA streams, in which a single or multiple cores can be used to handle multiple GPUs in a single node. The second, but rather more complex option, is to employ MPI and allow each compute rank to handle a GPU. The latter option has the advantage that it allows to utilize devices from multiple nodes. Furthermore, for programs already having MPI based CPU parallel implementations, the latter only requires a moderate coding effort. In MPI based multi-GPU programming, one can employ a root-worker model and design algorithms to eliminate the communications between devices. Alternatively, algorithms with device-device communication can be achieved using CUDA-aware MPI technology which, however, is not currently popular among many HPC applications due to the absence of mature MPI programming and performance models.[23] 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.




| 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 |

**Fig. 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 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, excludes bra types (indicated by white vertical strips) thus keeping only a set of ERIs that would be computed on the corresponding GPU. See text for details on the distribution algorithm.

### 3.2 Parallelizing ERI and ERI gradient schemes

The existing implementation of the ERI engine in QUICK can be mainly divided into four parts.[14,15] The first part is comprised of several host functions that process molecular and basis set information, compute Schwarz cutoff values and perform 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 μ, ν, λ, σ 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 **Fig. 1**). The calculated information is then broadcasted 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,[15] 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 $N^2$ x $N^2$ matrix problem with horizontal and vertical directions represented by a bra (*i.e.* [ab|) and ket (*i.e.* |cd]). The elements of such an ERI matrix are organized by four different criteria in each dimension. First, ERIs are sepa-

rated into dense or sparse regions based on the Schwarz cutoff value (see **Fig. 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 sub-zones (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 **Fig. 2**). Based on 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.

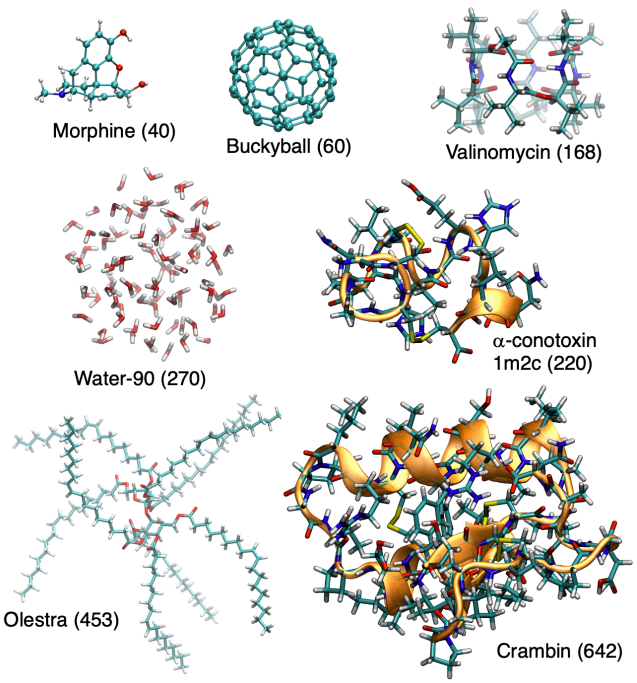



**Fig. 3.** Molecules used for benchmarks in this work. The number of atoms is listed in parenthesis.

**Table 1.** Wall times 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.[a]

| | B3LYP/6-31G[c] | | | | B3LYP/6-31G**[d] | | | | B3LYP/cc-pVDZ[e] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GPUs | ERI[b] | XC[b] | ERI gradient | XC gradient | ERI[b] | XC[b] | ERI gradient | XC gradient | ERI[b] | XC[b] | ERI gradient | XC gradient |
| 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]Each node has 4 NVIDIA V100-SXM2 type GPUs (32 GB), 2 Intel Xeon (R) Gold 6248 CPUs (2.50 GHz) and 374 GB memory. [b]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. [c]2131/4962 contracted/primitive functions. [d]4015/6846 contracted/primitive functions. [e]4015/9224 contracted/primitive functions.

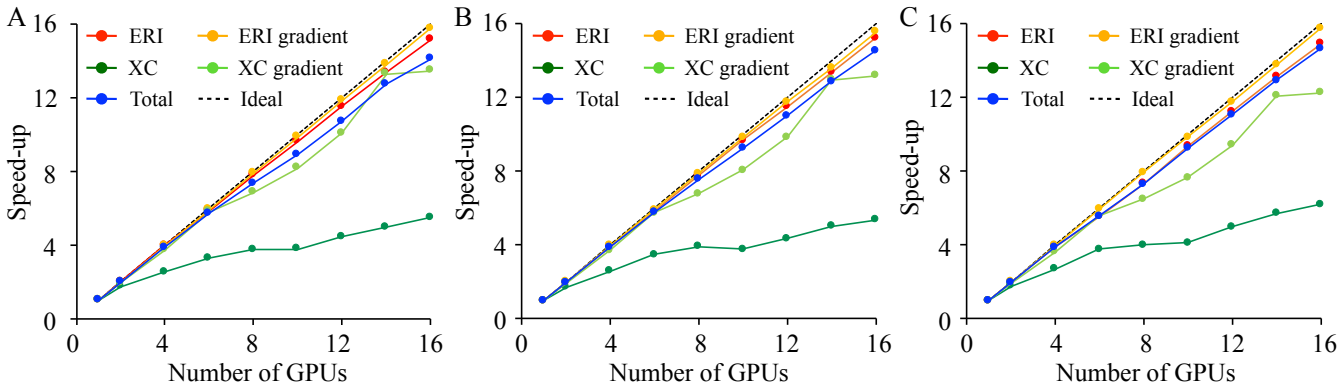



**Fig. 4.** Speed-up of ERI, XC and their gradient calculations for olestra at the B3LYP/6-31G (A), B3LYP/6-31G** (B) and B3LYP/cc-pVDZ levels of theory on up to 4 GPU nodes. Each node consists of 4 NVIDIA V100-SXM2 type GPUs, 2 Intel Xeon (R) Gold 6248 CPUs (2.50 GHz) and 374 GB memory per node. Total time is the summation of the reported time components.

### 3.3 Parallelization of XC and XC gradient schemes

The XC potential calculation in the serial GPU implementation follows a scheme involving three major steps.[16] 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. 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 **Fig. 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 upload repacked data 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 iteration, 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, grid weight gradient computation, is only required for points whose 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 serial 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 sub-sets of grid points, they may end up with 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 is uploaded, grid weight gradients are computed and 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.[16] We therefore limit current benchmarks to QUICK single- vs multi-GPU comparisons. **Fig. 3** depicts the organic molecules and protein systems that we have chosen for our benchmarks.

First, the performance of B3LYP gradient calculations on multiple GPUs is analyzed using olestra ($C_{156}H_{278}O_{19}$, see **Fig. 3** for molecular structure) with 3 different basis sets. The goal here is to analyze the parallel efficiency of the ERI, XC and their gradient computation tasks with different angular momentum basis functions and 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.

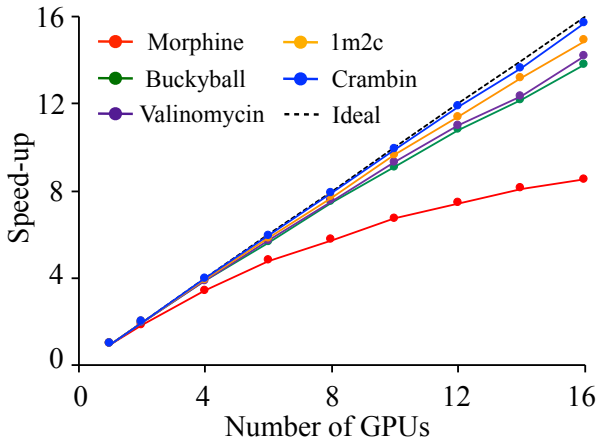



**Fig. 5.** Total speed-up for morphine (40 atoms, 410 basis functions), buckminsterfullerene (buckyball, 60 atoms and 900 basis functions), valinomycin (168 atoms, 1620 basis functions), α-contoxin (1m2c, 220 atoms and 2276 basis functions), crambin (642 atoms, 6504 basis functions) gradient calculations at B3LYP/6-31G** on up to 4 GPU nodes. Each node consists of 4 NVIDIA V100-SXM2 type GPUs, 2 Intel Xeon (R) Gold 6248 CPU (2.50 GHz) and 374 GB memory per node.

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 (R) 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 **Fig. S1**, **Table S1**) and circular thread walking was chosen for the ERI and ERI gradient computation. As documented previously,[15] 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. Based on 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 is due to the fact that both methods display similar performance.

**Table 2.** Wall times in seconds for ERI, XC potential, ERI gradient and XC gradient tasks of a water cluster (270 atoms, 2250 basis functions) gradient calculation at PBE0/def2-SVP level of theory (2250/3510 contracted/primitive functions) on up to 4 GPU nodes of V100, P100 and K80.[a]

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

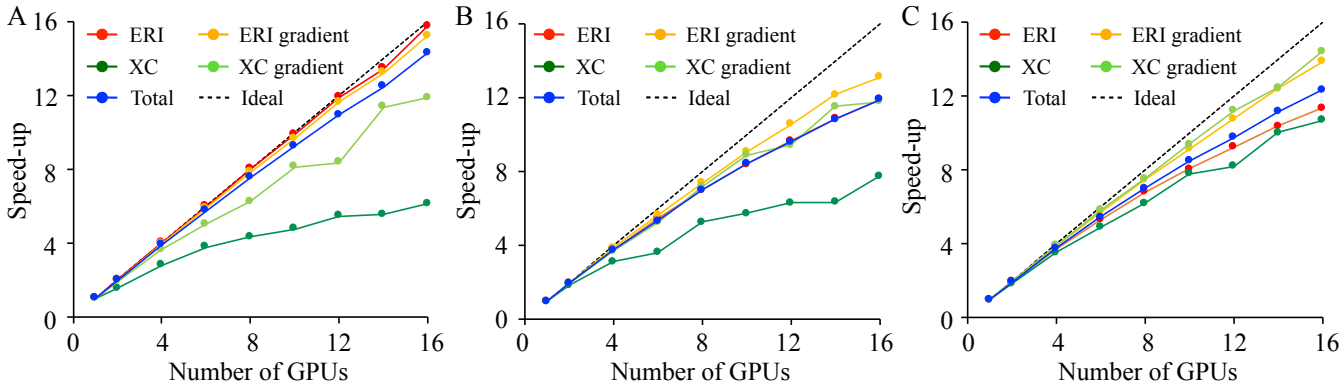



**Fig. 6.** Speed-up of ERI, XC and their gradient calculations for a water cluster (270 atoms, 2250 basis functions) at PBE0/def2-SVP level of theory on up to 4 GPU nodes of V100 (A), P100 (B) and K80 (C). Each V100 node is comprised of 4 NVIDIA V100-SXM2 type GPUs (32 GB), 2 Intel Xeon (R) Gold 6248 CPUs (2.50 GHz) and 374 GB memory. P100 nodes are equipped with 4 NVIDIA P100 type GPUs (16 GB), 2 Intel (R) Xeon (R) E5-2680 v4 CPUs (2.4 GHz) and 128 GB memory. Each K80 node has 4 NVIDIA K80 type GPUs (12 GB), 2 Intel (R) Xeon (R) E5-2680 v3 CPUs (2.5 GHz) and 128 GB memory.

In **Fig. 4**, we report the speed-ups (calculated as T(*serial*)/T(*n*) where T(*serial*) and T(*n*) are the wall times on single and *n* GPUs respectively) of the ERI, ERI gradient, XC potential and XC gradient calculation for olestra 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 operation times are reported in **Table S4** and the parallel efficiencies (calculated as 1/*n* * T(*serial*)/T(*n*) * 100) are reported in **Table S6-S8**. 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, non-linear scaling in speed-up despite the fact that their load balancing remains as impressive as that for the ERIs (see **Fig. S2**). The parallel efficiency for the XC potential diminishes with more 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 remains largely unaffected by the lower, non-linear scaling of the XC potential and energy gradient tasks. The total parallel efficiency in all 3 test cases remains greater than 88% on up to 16 GPUs. This is due to the fact that 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 operation times were small (see **Table S4**).

In **Fig. 5**, we report the total speed-up of B3LYP/6-31G** gradient calculations (Kohn-Sham operator formation during second SCF iteration and gradient computation) for 5 molecular systems of different size (see **Fig. 3** for structures). The parallel efficiencies and combined total times are reported in **Table S9** and **Table S14** respectively. As anticipated, the larger systems display better scaling with high parallel efficiency. For instance, crambin and 1m2c examples show an efficiency greater than 93% on 16 computing ranks. In contrast, for the smallest example morphine, the efficiency drops down to ~53% on 16 ranks. Such a performance decrease is expected since the workload becomes lighter in the presence of more compute 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 **Fig. 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 operation times are reported in **Table S5**. The parallel efficiencies are reported in **Tables S10-S12**. 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.[24–26] The best scaling for ERI and ERI gradient calculations is also observed 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 3 platforms must be due to their significant architectural differences.[24,25,27] The highest overall parallel efficiency (>89%) is achieved on the V100 class of GPUs. In addition to above platforms, we benchmarked the QUICK multi-GPU version on a single NVIDIA DGX A100 node[28] equipped with 8 A100 type GPUs (belonging to recent Ampere microarchitecture).[29] 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 S15**). In contrast, XC and XC gradient times remain substantially same. The observed scaling and parallel efficiencies are similar to that of 8 V100s (see **Fig. S3** and **Table S13**). On all platforms, the load balancing and MPI operation times remain considerably small.

## 5. Conclusions

We have reported the details of a MPI parallel 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, non-linear scaling for the XC and XC gradients resulting in an excellent aggregated parallel efficiency above 86%. Similar scaling is observed on A100, P100 and K80 platforms. The associated total parallel efficiencies were always greater than 74%, paving the way for large-scale *ab initio* electronic structure calculations. The benchmarks in the current study were limited to 4 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 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.

Finally, we recently integrated the QUICK serial GPU version as a library into the development version of the AMBER molecular dynamics package[30] enabling GPU capable quantum mechanics/ molecular mechanics (QM/MM) simulations. The integration of the multi-GPU version is currently in progress. QUICK version 21.03 can be downloaded from https://github.com/merzlab/QUICK under the Mozilla public license free of charge.

## ASSOCIATED CONTENT

Supporting text and Cartesian coordinates of the test molecules. This material is available free of charge via the Internet at http://pubs.acs.org.

## ACKNOWLEDGMENT

We thank Dmitry Pekurovsky from SDSC for improving our one-electron integral code, 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, 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. 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 number 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 Hour. 2018, arXiv:1706.02677v2, arXiv.org e-Print archive. https://arxiv.org/abs/1706.02677v2 (accessed Feb 05, 2021)

(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) 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.

(4) 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.

(5) 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.

(6) 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.

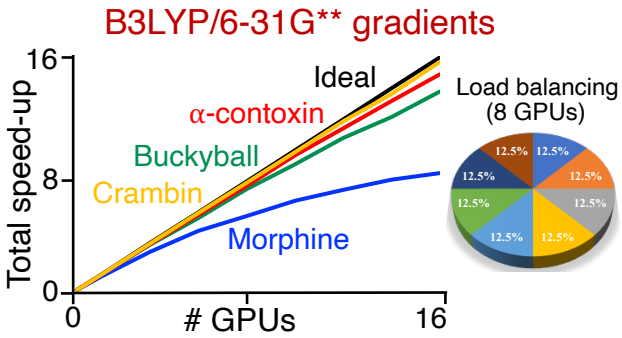
(7) Phillips, J. C.; Sun, Y.; Jain, N.; Bohm, E. J.; Kalé, L. V. Mapping to Irregular Torus Topologies and Other Techniques for Petascale Biomolecular Simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*; IEEE Computer Society, 2014; Vol. 2015-Janua, pp 81–91.

(8) 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.

(9) 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.

(10) Kussmann, J.; Ochsenfeld, C. Hybrid CPU/GPU Integral

Engine for Strong-Scaling Ab Initio Methods. *J. Chem. Theory Comput.* **2017**, *13*, 3153–3159.
(11) 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.
(12) Barca, G. M. J.; Galvez-Vallejo, J. L.; Poole, D. L.; Rendell, A. P.; Gordon, M. S. High-Performance, Graphics Processing Unit-Accelerated Fock Build Algorithm. *J. Chem. Theory Comput.* **2020**, *16*, 7232–7238.
(13) 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.
(14) 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.
(15) 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.
(16) 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.
(17) Obara, S.; Saika, A. Efficient Recursive Computation of Molecular Integrals over Cartesian Gaussian Functions. *J. Chem. Phys.* **1986**, *84*, 3963–3974.
(18) 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.
(19) 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.
(20) 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. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*; Budapest, Hungary, 2004; pp 97–104.
(21) Cheng, J.; Grossman, M.; McKercher, T. *Professional CUDA C Programming*; John Wiley & Sons, Inc.: Indianapolis, 2013; pp 67-264.
(22) Han, J.; Sharma, B. Scalable Multi-GPU Programming. In *Learn CUDA Programming: A Beginner's Guide to GPU Programming and Parallel Computing with CUDA 10.x and C/C++*; Packt Publishing: Birmingham, 2019; pp 241-274.
(23) Li, A.; Song, S. L.; Chen, J.; Li, J.; Liu, X.; Tallent, N. R.; Barker, K. J. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 94–110.
(24) NVIDIA. NVIDIA Tesla V100 GPU Architecture https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf (accessed Feb 25, 2020).
(25) NVIDIA. NVIDIA Tesla P100 https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf (accessed Mar 17, 2020).
(26) NVIDIA. Tesla K80 | NVIDIA https://www.nvidia.com/en-gb/data-center/tesla-k80/ (accessed Jan 9, 2021).
(27) 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 Jan 9, 2021).
(28) NVIDIA. NVIDIA DGX A100 | DATA SHEET | MAY20 https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf (accessed Feb 4, 2021).
(29) NVIDIA. NVIDIA A100 Tensor Core GPU Architecture https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf (accessed Feb 4, 2021).
(30) 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.


B3LYP/6-31G** gradients
Total speed-up
16
8
0
0
# GPUs
16
Ideal
α-contoxin
Buckyball
Crambin
Morphine
Load balancing
(8 GPUs)
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%
12.5%