Double-Buffered, Heterogeneous CPU + GPU Integral Digestion Algorithm for Single-Excitation Calculations Involving a Large Number of Excited States

Adrian F. Morrison,[a,b] Evgeny Epifanovsky,[b] and John M. Herbert *[a]

The most widely used quantum-chemical models for excited states are single-excitation theories, a category that includes configuration interaction with single substitutions, time-dependent density functional theory, and also a recently developed *ab initio* exciton model. When a large number of excited states are desired, these calculations incur a significant bottleneck in the "digestion" step in which two-electron integrals are contracted with density or density-like matrices. We present an implementation that moves this step onto graphical processing units (GPUs), and introduce a double-buffer scheme that minimizes latency by computing integrals on the central processing units (CPUs) concurrently with their digestion on the GPUs. An automatic code generation scheme simplifies the implementation of high-performance GPU kernels. For the exciton model, which requires separate excited-state calculations on each electronically coupled chromophore, the heterogeneous implementation described here results in speedups of 2–6× versus a CPU-only implementation. For traditional time-dependent density functional theory calculations, we obtain speedups of up to 5× when a large number of excited states is computed. © 2018 Wiley Periodicals, Inc.

DOI:10.1002/jcc.25531

Introduction

In an effort to increase the applicability and insight provided by quantum-chemical calculations, a great deal of work has been done to increase the size of the chemical systems that can be treated with popular algorithms using reasonable computational resources. There are generally two avenues that are pursued to make progress in this way: development of novel algorithms that either reduce the scaling or run more efficiently in certain situations by taking advantage of new physical *ansätze* or clever mathematical manipulations; or alternatively, one may reimplement existing algorithms to take advantage of new facets of rapidly evolving modern computer architectures. If both of these strategies can be pursued in tandem that is clearly ideal.

Pursuing the first of these strategies, we have recently introduced a novel method for computing excited-state properties of extended molecular aggregates.[1–5] Our method is based on the old idea of an exciton model,[6–9] originally due to Frenkel[6] and Davydov,[7] but is a fully *ab initio* realization of this model that we therefore dub the *ab initio* Frenkel-Davydov exciton model (AIFDEM).[5] This method accelerates an excited-state calculation by dividing it into monomer-sized components and pairwise couplings between them, yet remains faithful to the accuracy of excitation energies computed for the supersystem (at the level of configuration interaction with single substitutions), to within 0.1–0.2 eV.[1,2] Parallel efficiency is excellent by design, and a robust parallel implementation with near-perfect scaling was used to perform excited-state calculations in nanoscale systems with the equivalent of up to 50,000 basis functions, on laboratory-scale hardware, that is, a few hundred processors at most.[2,5] Systems of this size were found to exhibit quantum coherence effects in their excited-state energy-transfer dynamics, which are missing from small model systems.[2] Moving quantum chemistry to new architectures, and specifically graphical processing units (GPUs), represents a promising approach to extend these methods to the nanoscale,[10–12] a strategy that we apply here to the AIFDEM.

About 10 years ago, what had been a steady increase in frequency for integrated circuit switching hit a wall due to rapidly increasing thermal output, the primary means of improving computational capability up to that time. Since that point, the means for improving computational capabilities has been hardware that operates on many tasks concurrently, with the degree of parallelism increasing over time. Multiple central processing unit (CPU) cores are fabricated on a single chip and many chips are chained together such that the most powerful modern supercomputers have tens of thousands of CPU cores. Writing software for such massively parallel machines is challenging and often fundamentally limited by the nature of the algorithm. Indeed, as core counts increase there is a degree of "diminishing returns," as many problems are bottlenecked by the necessary communication of data and synchronization among the

[a] *Adrian F. Morrison, John M. Herbert*
Department of Chemistry and Biochemistry, The Ohio State University, Columbus, Ohio
E-mail: herbert@chemistry.ohio-state.edu
[b] *Adrian F. Morrison, Evgeny Epifanovsky*
Q-Chem Inc., Pleasanton, California

Contract Grant sponsor: U.S. Department of Energy;Contract Grant number: DE-SC0008550

many cores. As specialized vector processors, GPUs offer a path to both complement and move beyond the coarse-grained task parallelism offered by multicore CPUs. These processors have stricter constraints on their programming models but have raw processing capabilities that outstrip general-purpose CPUs and, with an efficient implementation, many problems can be significantly accelerated.[13]

As a practical yet expensive computational problem, quantum chemistry has naturally been an attractive target for GPU acceleration.[14] Electronic structure calculations based on Gaussian orbitals typically have two primary operations that are computationally expensive: generation of electron repulsion integrals (ERIs) and also linear algebra, the latter including tensor operations. The specific bottleneck depends upon the method in question, but these two potential bottlenecks have each been targeted for GPU acceleration. ERI generation was originally tackled on GPUs by Yasuda[15] and by Martínez and coworkers,[16–19] then later by others.[19–24] In early work, there was difficulty extending GPU-based ERI algorithms to basis sets with high angular momentum, as the intermediates required for computing high angular momentum shells were too large to store in GPU cache and registers, while the recursive nature of the integrals generation algorithm became rather complex for the "same instruction, multiple data" (SIMD)-style operations where GPUs excel. Evolution in hardware capability and algorithm design has since mitigated this difficulty,[19–22] to the point that a GPU-based ERI algorithm for arbitrary angular momentum has recently been reported.[22] That said, implementations for Gaussian basis sets with angular momentum beyond *d* functions are not yet widely available.

Based on these GPU-accelerated ERIs, GPU-based algorithms for self-consistent field (SCF) calculations were subsequently reported,[25–27] including density functional theory (DFT),[18] semiempirical methods,[28] and also several resolution-of-identity SCF methods.[22,29] GPU-based algorithms for single-reference excited-state methods, including both configuration interaction singles (CIS) and time-dependent DFT (TDDFT), are also available.[18,30]

Although ERI generation is a good candidate for GPU acceleration, the complicated nature of ERI algorithms, including their reliance on integral screening and control-flow,[17,24,31–33] is not ideally suited for these devices. Automated code generation can help in this respect,[19] but for methods based on correlated, post-Hartree–Fock wave functions, often the bottleneck computational steps involve linear algebra and tensor operations. Examples of GPU-accelerated post-Hartree–Fock algorithms include implementations of second-order Møller–Plesset perturbation theory,[34–38] symmetry-adapted perturbation theory,[39] coupled-cluster theory,[40–43] complete active space methods,[44–47] and full configuration interaction.[12] Taking advantage of the fact that the correlated part of the calculation can be cast as a series of matrix multiplications, these algorithms achieve performance enhancements by exploiting GPU-specific versions of basic linear algebra libraries. For higher-level many-body theories, the more complicated nature of the tensor operations may require specialized libraries to provide a general, GPU-accelerated tensor framework,[48] as for example in the case of coupled-cluster theory.[49]

In this work, we explore a GPU-accelerated version of the AIFDEM by exploiting the SIMD-amenable nature of its bottleneck computational step, namely, contraction of numerous density-like matrices with ERIs. This step also incurs significant cost in single-excitation theories of excited states, namely, TDDFT and CIS,[50] if the number of excited states is large. This might be the case in a semiconductor, where even truncated models lead to a very large density of states,[51] or in calculations of optical rotation parameters and circular dichroism spectra within a sum-over-states formalism,[52–55] where several hundred excited states may be required to converge the spectrum.[55–57] AIFDEM calculations use single-excitation calculations on individual monomers as basis states to describe collective excitations, and as such a large number of monomer excited states is often required. We discuss the implementation details including a double-buffer scheme to hide CPU-to-GPU transfer latency, and machine generated code that simplifies the implementation of specialized kernels. Finally, the algorithm is benchmarked with synthetic examples as well as real systems, and its performance is detailed.

Background

The computationally intensive part of an AIFDEM calculation involves formation of a Hamiltonian whose matrix elements are given by

$$H_{AB}=\sum_{i\in A}\sum_{j\in B}t^i t^j \xi^{ij}_\alpha \xi^{ij}_\beta \Gamma^{ij} \tag{1}$$

where

$$\begin{aligned}\Gamma^{ij}=&\left(\mathbf{G}^{ij}_\alpha+\mathbf{G}^{ij}_\beta\right)\cdot\mathbf{h}+\frac{1}{2}\mathbf{G}^{ij}_\alpha\cdot\mathbf{\Pi}\cdot\mathbf{G}^{ij}_\alpha\\&+\frac{1}{2}\mathbf{G}^{ij}_\beta\cdot\mathbf{\Pi}\cdot\mathbf{G}^{ij}_\beta+\mathbf{G}^{ij}_\alpha\cdot\mathbf{\Pi}^\circ\cdot\mathbf{G}^{ij}_\beta.\end{aligned} \tag{2}$$

The notation and underlying theory is explained more fully in Ref. [4]. Briefly, we define the contraction of matrices **R** and **S** as

$$\mathbf{R}\cdot\mathbf{S}=\sum_{ij}R_{ij}S_{ij}, \tag{3}$$

where in this work, at least one of **R** or **S** is always a symmetric matrix. Indices *i* and *j* denote natural transition orbitals (NTOs) on individual fragments (monomers) *A* and *B*, which contribute to generalized density matrices $\mathbf{G}^{ij}_\alpha$ and $\mathbf{G}^{ij}_\beta$. The matrix **h** is the one-electron Hamiltonian, the t^i are the fragment configuration interaction (CI) coefficients, and factors ξ^{ij}_α and ξ^{ij}_β ensure consistent normalization and phase.

Our concern is with the two-electron part of Γ^{ij}. The four-index tensor of anti-symmetrized ERIs is indicated by $\mathbf{\Pi}$ in equation 2,

$$\Pi_{\mu\nu\lambda\sigma}=(\mu\nu\|\lambda\sigma)\ , \tag{4}$$

whereas $\mathbf{\Pi}^\circ$ includes Coulomb integrals only, $\Pi^\circ_{\mu\nu\lambda\sigma}=(\mu\nu|\lambda\sigma)$. The dimension of *ij* is typically on the order of dozens to several hundred generalized densities $\mathbf{G}_{\sigma}^{ij}$, and contraction of ERIs with this many density-like matrices constitutes the overwhelming computational bottleneck in AIFDEM calculations. As noted above, excited-state methods such as CIS and TDDFT[50] which are the most widely used quantum chemistry methods for electronically excited states, also include a significant contraction cost, when a large number of excited states is required.

In the language of ERI algorithms, contraction of integrals with density-like matrices is sometimes called "digestion."[32,33,58–61] A call-graph diagram in Figure 1 demonstrates the proportion of either a CIS calculation or an AIFDEM calculation that is taken up by the digestion step. The latter calculation is rather modest, including just one excited state on each of four H_2O monomers, which is the minimal basis to compute the lowest collective excitation of the entire $(H_2O)_4$ cluster, yet digestion already consumes 80% of the total calculation time. For comparison, a CIS calculation of the lowest excited state of $(H_2O)_4$ spends only 30% of its time in the digestion step, although this step will consume an increasingly large fraction of the total job time as more excited states are requested.

The digestion procedure in question can be written as the contraction of a four-index tensor containing the ERIs with a three-index tensor representing the generalized densities. The result is a Fock-like matrix that, operationally, is another three-index tensor. This operation can be expressed as

$$F_{\mu\nu}^{I} = \sum_{\lambda\sigma} (\mu\nu\|\lambda\sigma) G_{\lambda\sigma}^{I} , \tag{5}$$

where we have replaced the composite index *ij* with a single index *I* for simplicity, and $(\mu\nu\|\lambda\sigma)$ is an ERI in the atomic orbital (AO) basis. Essentially, the same digestion procedure is required in traditional CIS and TDDFT calculations, as discussed below.

An analogous digestion step, albeit with slightly more symmetry, is performed in the ground-state SCF iterations as well, but only one or two densities need to be digested, depending on whether the calculation is restricted or unrestricted. ERI libraries are usually highly optimized for this task and the digestion step is not a significant fraction of the cost. Integrals are typically generated and then immediately digested in batches that are small enough to fit in CPU caches. When the dimension of the density or density-like matrices exceeds a certain limit, however, these quantities no longer fit in cache and the requisite trips to main memory severely impact performance. On the bright side, a large degree of data parallelism becomes apparent when many density-like matrices are present as essentially the same operation is performed for each, and the integrals can therefore be reused. This general concept is the foundation of vector processing (i.e., SIMD operations), for which specialized instructions and execution units are present in modern processor architectures. These types of operations are abundant in graphics rendering, and thus, GPUs are designed around this type of work. (Technically, typical GPUs are designed such that parallel operations execute in their own register spaces and as such are more appropriately designated single instruction, multiple thread processors, or SIMT.) With these considerations in mind, we expect that a GPU implementation of equation 5 will accelerate AIFDEM calculations by a significant degree.

We note in passing that equation 5 can also be accelerated by means of a resolution-of-identity approximation, also known as "density fitting."[62–64] The resolution-of-identity procedure approximates the four-index tensor $\mathbf{\Pi}$ in terms of three-index integrals and two-index fitting coefficients for products of Gaussian basis functions, expanded in a (larger) auxiliary Gaussian basis set. Although the resulting algorithm exhibits the same formal scaling as the original one, the prefactor (and thus

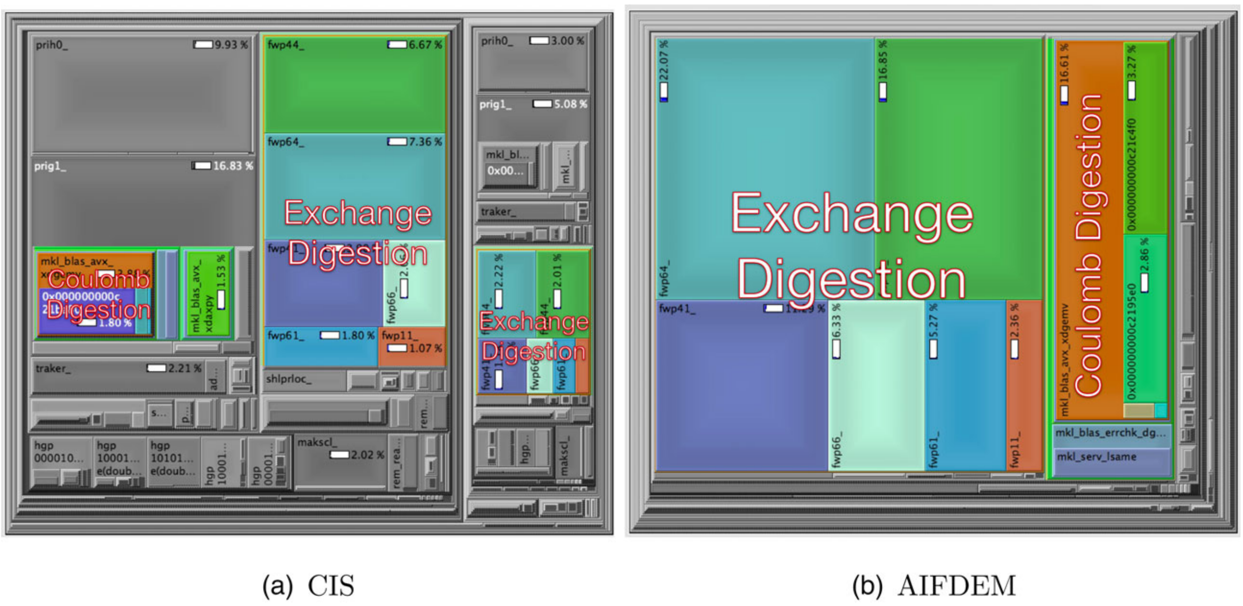

Figure 1. Call graph diagrams for calculations on $(H_2O)_4$ using the 6-31G basis set: (a) a CIS calculation of only one excited state, and (b) an AIFDEM calculation including one excited state per monomer. Both calculations used the Q-Chem program.[58] The diagrams outline all the major routines called by the program during the course of the calculation, with the size of the boxes representing the relative amount of wall time spent in the indicated routine. The routines that perform integral digestion are colored and labeled. [Color figure can be viewed at wileyonlinelibrary.com]

the computing time) is often dramatically reduced,[63–70] and this approach has been used in particular in the context of TDDFT.[65,67,71] The AIFDEM would likely benefit from a resolution-of-identity implementation but the need for digestion with a large number of density-like matrices necessitates a more specialized implementation as compared to the normal TDDFT one. Such an implementation would itself be amenable to GPU acceleration, but these improvements lie beyond the scope of this work.

Finally, we note that a different GPU-accelerated "*ab initio* exciton model" has been reported by Martínez and coworkers.[72–74] While the underlying physical model is similar in spirit, the version developed by Martínez *et al*. invokes significant simplifying approximations, including a dipole–dipole approximation for the coupling matrix elements H_{AB}, and neglect of exchange interactions. These approximations make sense in the context of the photosynthetic light-harvesting complex examined in Ref. [73], where the individual chromophores are well separated, but may be problematic in densely packed molecular crystals and aggregates, and in molecular liquids, which are the applications that we have so far targeted using the AIFDEM.[1–5] GPU acceleration reported in Refs. [72–74] amounts to acceleration of ERIs needed for traditional TDDFT calculations, whereas we have tackled the digestion step instead.

Computational Details

Algorithm design

We have designed a heterogeneous CPU + GPU implementation of equation 5, in which integral are generated on the CPU and digested on the GPU. In some ways, this approach plays to the strengths of each machine: integral generation requires branching recursion relations and generates many shared intermediate quantities, whereas digestion entails straightforward (if copious) multiply-add operations. However, this strategy runs directly up against what is perhaps the biggest challenge in GPU algorithm design, namely, transfer of data from CPU memory to GPU memory via the high-latency PCIE bus, potentially incurring significant performance penalties. Recent versions of NVIDIA's CUDA platform have introduced capabilities for concurrent data transfer and compute on GPU devices as well as APIs that allow for asynchronous calls from the perspective of the CPU. Utilizing these capabilities, our algorithm is implemented using a double-buffer strategy that hides the PCIE transfer latency by concurrent execution of digestion on the GPU, data transfer over the bus, and integral generation on the CPU. This strategy reduces stalls, in which one part of the machine sits idle waiting for data. Our approach is outlined schematically in Figure 2, which compares it to a traditional CPU-based digestion algorithm.

Our approach preorganizes integrals into batches that fit into available device memory, and furthermore makes use of two instances of a buffer data structure with host and device components. At step N, the CPU generates the Nth batch of integrals and stores them in host buffer 0. A synchronization call here will block the CPU until device buffer 0 is available and then the integrals are transferred asynchronously to the device so that the CPU can immediately begin to generate batch $N + 1$ in host buffer 1. Meanwhile, the GPU has been digesting batch $N - 2$ while batch $N - 1$ is transferring to device buffer 1, such that these integrals are now available for the GPU to digest when batch $N - 2$ is complete. By the time batch $N - 1$ is digested, the integrals for batch N are available on the GPU and the cycle continues until all batches are digested. In practice, there are bubbles in this pipeline, typically due to generation of high angular momentum integrals that cause the GPU to stall, but we find that the majority of the work overlaps nicely on our testbed, leading to good performance as evidenced by results presented below.

Kernel design

The heart of any GPU algorithm is the computational kernels that run on the device and the challenge is to map the

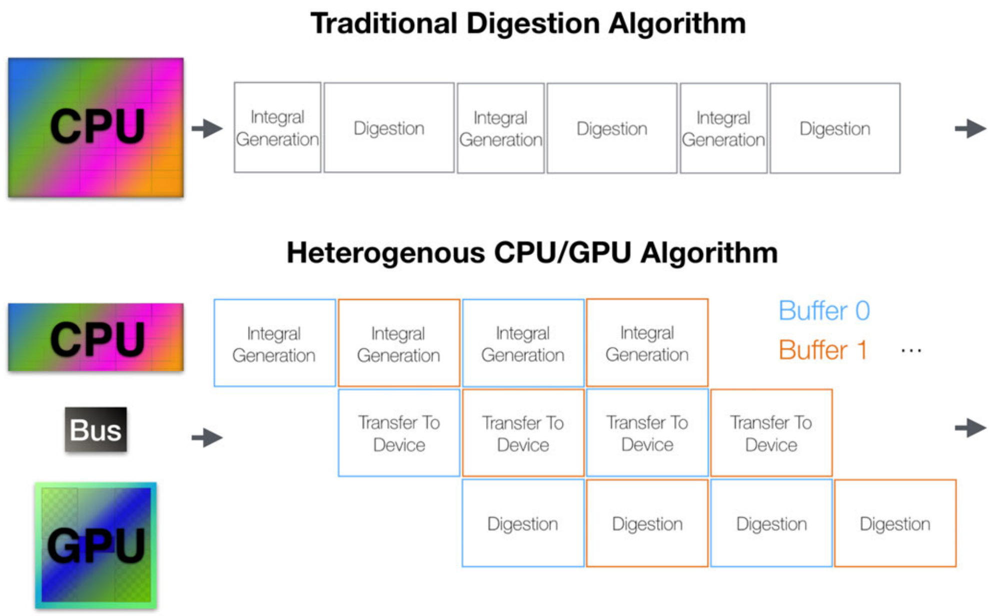

Figure 2. Schematic depiction of a traditional CPU-based digestion algorithm used in CIS and TDDFT calculations, as compared to the double-buffered, CPU + GPU algorithm implemented in this work. [Color figure can be viewed at wileyonlinelibrary.com]

particular problem to its hardware layout. Due to the symmetry of the two-electron integrals, we can avoid computing all of the integrals by transposing the tensor during digestion. With the usual notation,[75] where **J**, **K**, and **P** denote the Coulomb, exchange, and density-like matrices, respectively, the necessary transpositions to construct **J** are

$$J^I_{\mu\nu} = \sum_{\lambda\sigma} (\mu\nu|\lambda\sigma) P^I_{\lambda\sigma} \tag{6a}$$

and

$$J^I_{\mu\nu} = \sum_{\lambda\sigma} (\lambda\sigma|\mu\nu) P^I_{\lambda\sigma} \,, \tag{6b}$$

while for **K** they are

$$K^I_{\mu\nu} = \sum_{\lambda\sigma} (\mu\lambda|\nu\sigma) P^I_{\lambda\sigma} \tag{7a}$$

$$K^I_{\mu\nu} = \sum_{\lambda\sigma} (\lambda\mu|\nu\sigma) P^I_{\lambda\sigma} \tag{7b}$$

$$K^I_{\mu\nu} = \sum_{\lambda\sigma} (\mu\lambda|\sigma\nu) P^I_{\lambda\sigma} \tag{7c}$$

$$K^I_{\mu\nu} = \sum_{\lambda\sigma} (\lambda\mu|\sigma\nu) P^I_{\lambda\sigma} \,. \tag{7d}$$

These equations assume that **P** is symmetric but the generalized density matrices **G** used in the AIFDEM (and in CIS and TDDFT calculations) are not. If **P** is not symmetric, then construction of the exchange matrix in equation (7) should be modified according to

$$K^I_{\nu\mu} = \sum_{\lambda\sigma} (\mu\lambda|\nu\sigma) P^I_{\sigma\lambda} \tag{8a}$$

$$K^I_{\nu\mu} = \sum_{\lambda\sigma} (\lambda\mu|\nu\sigma) P^I_{\sigma\lambda} \tag{8b}$$

$$K^I_{\nu\mu} = \sum_{\lambda\sigma} (\mu\lambda|\sigma\nu) P^I_{\sigma\lambda} \tag{8c}$$

$$K^I_{\nu\mu} = \sum_{\lambda\sigma} (\lambda\mu|\sigma\nu) P^I_{\sigma\lambda} \,. \tag{8d}$$

When dealing with higher-order tensors, it is often advantageous to reorder these quantities in memory prior to contraction, if this reordering can reduce a given tensor operation to a matrix multiplication for which highly optimized routines are available. As discussed below, however, the integrals in our algorithm are digested in small blocks, in which case the memory operations associated with transposition outweigh the efficiency benefits of matrix multiplication.[76] As such, in our implementation the blocks of the integral tensor are not reordered after production, however their elements are accessed in the patterns dictated by equations (6) and (8).

Integrals are generated in blocks corresponding to shell quartets, and rather than accumulate the entire tensor it is sensible to digest the integrals block-wise by quartet. The dimension of each block is then determined by the number of functions on each center in the quartet. Therefore, there are a finite number of possibilities for the contraction dimension, that is, the ranges of $\mu\nu\lambda\sigma$ in equations (6)–(8), dictated by the degree of angular momentum and occurrence of multishells in a given basis set. Our strategy was to write a single kernel and utilize C++ templates to generate specialized code for each quartet dimension and integral transposition. This maximizes opportunities for compiler-level optimization and also simplifies the implementation, as a new kernel can be written with single line of code. Tuning the kernels by hand for every quartet and contraction length might result in better performance, but places the programmer's sanity at risk. Note that the sheer number of different parameters ensures that each individual kernel runs only for a fraction of the total job time, making specialized optimizations impractical. A source-code illustration of the template strategy is presented in Figure 3.

Although a kernel can be implemented in a single line of code for a given quartet size, and there are a finite number of possible contraction lengths (multishell component dimensions), the number of possible kernels will still grow rapidly as new basis sets are implemented. We used automatic code generation to simplify the implementation of new basis sets, via

```
//digestion kernel template
template<size_t na, size_t nb, size_t nc, size_t, nd>
void cudigest(F,W,P){

   // J transpositions
   contraction_kernel<tensor_J1,na,nb,nc,nd>(F,W,P);
   contraction_kernel<tensor_J2,nc,nd,na,nb>(F,W,P);

    // K transpositions
   contraction_kernel<tensor_K1,na,nc,nb,nd>(F,W,P);
   contraction_kernel<tensor_K2,nb,nc,na,nd>(F,W,P);
   contraction_kernel<tensor_K3,na,nd,nb,nc>(F,W,P);
   contraction_kernel<tensor_K4,nb,nd,na,nc>(F,W,P);

}
// example kernel implementations

//(dd|dd)
cudigest<6,6,6,6>(F,W,P);

//(d sp|d s)
cudigest<6,4,6,1>(F,W,P);
```

Figure 3. Illustration of the C++ template strategy used for GPU kernel generation. A single *contraction kernel* is instantiated with *tensor* data structures that invoke integral tensor access patterns corresponding to equations (6) and (7). The kernel is also templated over the number of components of each shell in the quartet, *na nb*, *nc*, and *nd*. New kernels are implemented by specifying these dimensions as template parameters.

auxiliary Python scripts. The code generator takes as inputs a list of basis sets and the set of possible shell component dimensions for each basis; these can be printed in a preliminary dry run. For each basis, the script will generate all possible quartets that are unique under the eightfold symmetry of the integral tensor, and cull any repeated quartets among the basis sets. Finally, the necessary C++ files are written to the source directory, which in practice is more than a single line per kernel to reduce compile time. For the basis sets used in this work, a total of 65 kernels were required.

We have found it advantageous to exclude very small quartets, such as $1 \times 1 \times 1 \times 1$, from the GPU digestion as the amount of GPU work is not worth the cost of the memory operations needed to move it onto the GPU. Several CPU threads are reserved to compute and digest these cases entirely on the CPU. In principal, these threads will also handle any quartet cases for which CUDA kernels have not been implemented. For the testbed considered here we found that reserving three such threads was optimal, but this will need to be tuned for other machines.

The basic contraction kernel assigns a group of GPU threads to compute contributions to a given **J** or **K** matrix, arising from a given integral quartet, for a portion of the *I* subspace. Each thread accumulates a single **J** or **K** matrix element. Integral quartets are presorted by dimension so all contractions for a given quartet class are launched simultaneously; these thread blocks will run concurrently on the device. The kernel will initially load the current block of integrals and density-like matrices into fast level 2 cache, called shared memory in CUDA parlance. After contraction, the matrix elements are summed into the global **J** and **K** matrices in device memory, using atomic add operations provided by the CUDA API. For the applications considered here, we do not find these atomic operations to be a major bottleneck, and they furthermore eliminate the need for temporary buffers, thus reducing memory requirements significantly.

Results

All of our benchmarks were run on a single node with 2× Intel Xeon E5–2680 v4 CPUs (28 total cores), 128 GB of main memory, and a NVIDIA Tesla P100 GPU with 16 GB of memory. All tests were performed at the Ohio Supercomputer Center.[77] We note that our algorithm utilizes the GPU as well as all CPU cores on a node, therefore, we compare our accelerated CPU + GPU algorithm to the CPU only case, with the same CPU type and memory, rather than a direct CPU versus GPU comparison.

Synthetic benchmarks

We first tested our algorithm against the equivalent CPU-only integral routine (as implemented in Q-Chem,[58] v. 5.0) on a set of synthetic benchmarks. Here, density-like matrices, which we henceforth call "pseudo-densities," were generated randomly in the AO basis set of the given test systems. These pseudo-densities are asymmetric and restricted to a single spin.

Results in Figure 4 demonstrate impressive speedups. For a linear chain of 100 helium atoms, the speedup (relative to the CPU-only algorithm reported previously[1]) increases from ≈3 to ≈12.5 as the number of pseudo-densities is increased from 8 to 256. For that system, we use a toy basis set (not intended for practical calculations) that contains only *d* functions, so that the only quartet class is (*dd*|*dd*). As such, results for this test case represent a practical upper limit on the performance that we can expect, as this single shell-quartet class has the maximum dimension ($6 \times 6 \times 6 \times 6$) of any system and basis set considered here and so provides an optimal balance of memory and arithmetic operations for the kernel.

For a realistic test case, namely, pentacene dimer in the cc-pVDZ basis set, the CPU + GPU algorithm is already slightly faster than CPU-only Q-Chem at 8 pseudo-densities and approximately twice as fast at 32 pseudo-densities. For pentacene monomer in the cc-pVDZ basis, the CPU + GPU algorithm achieves a 4× speedup relative to CPU-only Q-Chem when digesting 256 pseudo-densities. (This is a realistic number of generalized densities for a typical AIFDEM calculation.) Pentacene dimer in the same basis achieves approximately 6× speedup, indicating that relative performance improves as system size increases. There is plateau in the speedup provided by the GPU + CPU algorithm that is evident in Figure 4 for all test cases when the number of pseudo-densities increases from 128 to 256. The fundamental scaling of the CPU + GPU algorithm is identical to the CPU only version and at 128 pseudo-densities the GPU is saturated with digestion work. After this point, the speedup of the CPU + GPU algorithm, while significant, remains constant as the number of pseudo-densities increases.

AIFDEM benchmarks

We have adapted the AIFDEM electronic structure method to use our GPU-accelerated integral digestion algorithm, in a locally modified version of Q-Chem,[58] v. 5.0. We next investigate its performance, considering as a first example the pentacene dimer in a variety of AO basis sets, and varying also the NTO threshold that is used in the calculations. This threshold

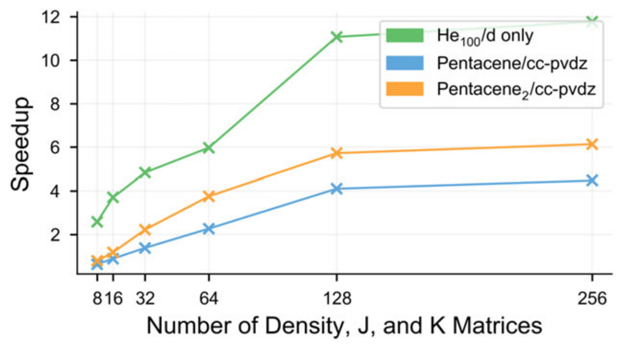

Figure 4. Speedup for the heterogeneous CPU + GPU algorithm relative to a CPU-only algorithm running on equivalent hardware, for synthetic benchmarks involving randomly generated AO density matrices. The He_{100} system uses a toy basis set containing only *d* functions and represents a practical upper limit on expected performance improvements. [Color figure can be viewed at wileyonlinelibrary.com]

represents a controlled approximation for discarding terms in equation 1, based on the magnitudes of t^i and t^j.[1,2] Specifically, we retain enough CI coefficients to obtain a specified fraction of the norm $\|\mathbf{t}\|$ of the transition density. This truncation limits the number of generalized densities $\mathbf{G}_\sigma^{ij}$ that must be digested, and this number grows approximately quadratically as the threshold is tightened due to the product t^it^j that appears in equation 1. An 80% threshold is sufficient for an accuracy of ≈0.1 eV with respect to a supersystem calculation,[1] which we judge to be acceptable.

Timing comparisons between the CPU-only and CPU + GPU algorithms are provided in Table 1, for two different basis sets and two different values of the NTO threshold. The number of generalized densities $\mathbf{G}_\sigma^{ij}$ that must be digested is listed in each case and at first glance, the timings might not seem to correlate with those in Figure 4. For instance, a calculation with pentacene dimer in the cc-pVDZ basis set yields a 2.3× speedup relative to the CPU-only algorithm in Q-Chem, and while this is impressive it is not the ≈6× speedup that one might have expected for this calculation with 162 generalized density matrices, based on Figure 4. This is because the AIFDEM requires integrals over spin-unrestricted densities, $\mathbf{G}_\alpha^{ij}$ and $\mathbf{G}_\beta^{ij}$, although the Coulomb matrix can be formed from the total density. As such, the effective number of density-like matrices digested is approximately twice the value given in Table 1 and the CPU-only algorithm does not see an increase in wall time to the same degree as the GPU algorithm when moving from a restricted to an unrestricted density, perhaps due to some overhead that exists even in the restricted case. Nevertheless, the hybrid CPU + GPU algorithm affords speedups of ≈4× for the aug-cc-pVDZ test case, again indicating the value of this algorithm for larger systems or those with less sparsity.

Lastly we performed tests on a much larger system, namely, a substructure of a self-assembled organic semiconductor nanotube.[51] The complete nanotube is comprised of ~350 naphthalene tetracarboxylate diimide chromophores; a nine-unit substructure is used here, which is depicted in Figure 5 and which we have considered also in previous work.[1,2,51] Timings for this system are presented in Table 2. These calculations use a charge-embedded version of the AIFDEM,[2] such that no more than two monomers are included in the electronic structure calculation at any one time, with the other monomers described using atom-centered point charges. These calculations use Pople basis sets and calculations on the entire nine-unit structure in Figure 5 would amount to 3150 basis functions for 6-31G* and 3942 functions for 6-31+G*. Speedups for this system are generally better than those for pentacene dimer and range from 2.4× for the 6-31G* basis and 75% NTO threshold, to more than 5× when diffuse functions are added (6-31+G*) and the threshold tightened to 85%. The larger speedups as compared to pentacene dimer likely arise from the combination of a larger system (with a correspondingly larger amount of contraction work) as well as the use of Pople-style combined *sp* shells, resulting in more optimal contraction dimensions than if *s* and *p* shells were separated, as is done in (aug-)cc-pVDZ.

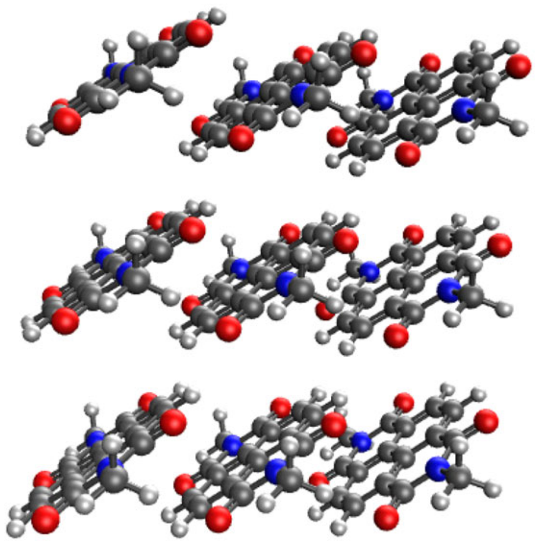
Figure 5. Nine-unit model of an organic semiconductor nanotube, from Ref. [51]. The individual chromophores in this model system are *N*, *N*′-dimethyl-1,4,5,8-naphthalene tetracarboxylic diimide, $C_{16}O_4N_2H_{10}$. [Color figure can be viewed at wileyonlinelibrary.com]

TDDFT benchmarks

Thus far, we have focused on examples where our CPU + GPU algorithm was applied to AIFDEM calculations, but it may be

Table 1. Wall times for an AIFDEM calculation of the singlet excited states of pentacene dimer, comparing a CPU-only algorithm to the hybrid CPU + GPU algorithm.

	Basis set			
	cc-pVDZ		aug-cc-pVDZ	
NTO threshold[a]	75%	85%	75%	85%
No. of gen. densities[b]	162	648	200	648
CPU-only time[c] (s)	3,462	13,679	114,866	365,567
CPU + GPU time[c] (s)	1,533	5,265	31,660	97,149
Speedup factor	2.26	2.60	3.63	3.76

[a] Percentage of $\|\mathbf{t}\|$ retained in the fragment calculations.
[b] Representative; for a single diagonal matrix element.
[c] Excludes fragment calculation time.

Table 2. Wall times for a charge-embedded AIFDEM calculation of the singlet excited states of the nine-unit naphthalene diimide structure in Figure 5.

	Basis set			
	6-31G*		6-31+G*	
NTO threshold[a]	75%	85%	75%	85%
No. of gen. densities[b]	200	648	200	722
CPU-only time[c] (s)	16,718	50,876	101,135	379,751
CPU + GPU time[c] (s)	7,008	21,821	22,104	72,912
Speedup factor	2.38	4.58	4.58	5.21

[a] Percentage of $\|\mathbf{t}\|$ retained in the fragment calculations.
[b] Representative; for a single diagonal matrix element.
[c] Excluding fragment calculation time.

useful to accelerate traditional CIS or TDDFT calculations as well, within the Tamm-Dancoff approximation[50] in the latter case. The bottleneck step in these single-excitation theories is the formation of the subspace vectors σ_{ia}^I required by the iterative eigensolver[78]:

$$\sigma_{ia}^I = \sum_{jb} \Big[(\epsilon_a - \epsilon_i)\delta_{ij}\delta_{ab} + (ai|jb) + C_{\mathrm{HF}}(ab|ji) + \big(ai|\hat{f}_{\mathrm{xc}}|jb\big) \Big]\, t_{jb}^I. \tag{9}$$

See Ref. [50] for a complete explanation of the notation. Briefly, ERIs $(ai|jb)$ and $(ab|ji)$ are expressed in the molecular orbital (MO) basis, with indices $i, j, \ldots$ and $a, b, \ldots$ corresponding to occupied and virtual MOs, respectively. The quantities ϵ_i and ϵ_a are orbital eigenvalues, and $\mathbf{t}^I$ is a (trial) transition density matrix for the Ith excited state.

As in the AIFDEM, the bottleneck step in CIS and TDDFT calculations is formation of the two-electron contribution to the subspace vectors, and this can be written in terms of a Fock-like matrix $\widetilde{\mathbf{F}}$:

$$\widetilde{F}_{ia}^I = \sum_{\mu\nu} c_{\mu a} c_{\nu i} \sum_{\lambda\sigma} \Big[(\mu\nu|\lambda\sigma) + C_{\mathrm{HF}}(\mu\sigma|\lambda\nu) + \big(\mu\nu|\hat{f}_{\mathrm{xc}}|\lambda\sigma\big) \Big] \widetilde{P}_{\lambda\sigma}^I. \tag{10}$$

Here, $\widetilde{\mathbf{P}}^I$ is the transition density matrix transformed to the AO basis:

$$\widetilde{P}_{\lambda\sigma}^I = \sum_{jb} c_{\lambda j}\, t_{jb}^I\, c_{\sigma b}\ . \tag{11}$$

Contraction of the transition density matrix with the ERIs in equation 11 is analogous to equation 5 and is operationally equivalent. In this case, the dimension of the space indexed by I is the number of unconverged roots at a given iteration of Davidson's procedure.[79] Initially, this dimension equals the number of desired excited states, but decreases as the roots iteratively converge.[78]

Typical calculations of this type do not request more than 10–20 excited states, but there are certain cases where a much larger number of states is necessary, and it is in these cases where equation 11 is a good candidate for our GPU-accelerated digestion algorithm. If the excited-state manifold is dense with near degeneracies, for example, then the state of interest might be of high rank. The electronic structure of the nine-chromophore model in Figure 5 is nearly semiconductor-like, and its lowest optically bright transition (at the LRC-ωPBE/3-21G* level of theory) is actually the 27th excited state.[51] Numerous excited states are also required to compute circular dichroism spectra within a sum-over-states formalism,[52–55] as this approach often requires hundreds of excited states to converge, even for small organic molecules.[55–57] Circular dichroism spectra for pyrrole (C_4H_5N) reported in Ref. [55], for example, require a few dozen excited states just to reproduce the rough features of the spectrum, while 200 states are required to begin to resolve details in the short wavelength region, and 300 states to obtain a fully converged spectrum.

Table 3. Wall times for TDDFT calculations on a guanine/cytosine base pair.

	Basis set			
	cc-pVDZ		aug-cc-pVDZ	
No. of states	200	300	200	300
CPU-only time[a] (s)	312	537	2902	7691
CPU + GPU time[a] (s)	275	453	1088	1639
Speedup factor	1.13	1.23	2.67	4.69

[a] Excluding SCF time, B3LYP level.

With these examples in mind, we have adapted the CIS/TDDFT code in a development version of Q-Chem,[58] v. 5.0, to use our GPU-accelerated integral digestion algorithm, and have benchmarked it against the original, CPU-only implementation. We took a Watson-Crick guanine/cytosine base pair as a test system and computed several hundred states at the B3LYP level of theory using two different basis sets; results are presented in Table 3. As with the AIFDEM calculations, the speedup afforded by the CPU + GPU implementation increases in larger basis sets, and is only 10%–20% in the cc-pVDZ basis but larger in aug-cc-pVDZ. Larger speedups are also obtained as the number of transition densities increases, which is dictated in this case by the number of excited states that is requested. In that sense, CIS and TDDFT calculations are somewhat less amenable to GPU acceleration as compared to the AIFDEM calculations, because the number of density-like matrices that must be digested decreases (and thus the relative advantage of using GPUs also decreases) as the Davidson iterations proceed. Regardless, the CPU + GPU algorithm does quite well in the aug-cc-pVDZ example, with speedups approaching 5×.

Finally, as a realistic example where one might need 100+ excited states to obtain an electronic absorption spectrum we consider the cobalt(II) corrinoid complex shown in Figure 6, which is a truncated model of the adenosylcobalamin cofactor, from a joint experimental and computational study in Ref. [80]. In TDDFT calculations using either the local density approximation (as in Ref. [80]) or the BP86 generalized gradient approximation (which was preferred in other studies of cobalamins[81,82]), one must compute ~50, 100, or 200 excited states to reach 4, 5, or 6 eV (respectively) above the ground state. Other functionals have been examined for this system as

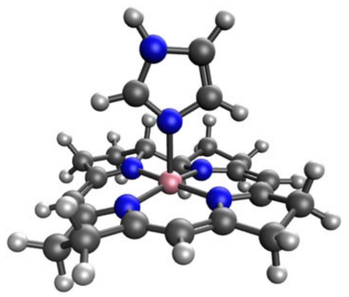

Figure 6. Structure (from Ref. 80) of a truncated Co^{2+} corrinoid model of the base-off form of cobalt(II)cobalamin. [Color figure can be viewed at wileyonlinelibrary.com]

well,[81,82] and we have performed TDDFT calculations on the complex in Figure 6 using B3LYP, computing 200 excited states to obtain the spectrum up to 6 eV above the ground state. A CPU-only version of Q-Chem required 3351 s (def2-SVP basis set) or 23,499 s (def2-SVPD basis). Using our GPU + CPU algorithm we were able to complete the same calculations in 1714 and 7551 s, respectively, corresponding to speedups of 2× for def2-SVP and 3× for def2-SVPD. This is in line with results for the other examples considered here.

Conclusions

We have reported a hybrid CPU + GPU algorithm for efficient digestion of two-electron integrals with a large number of density-like matrices, by appealing to the intrinsically data-parallel nature of this operation. This effort specifically targets the *ab initio* exciton model that we developed previously,[1–5] for which the digestion step proves to be a significant bottleneck. For synthetic benchmarks on real systems, we demonstrate speedups of over 6× as compared to a CPU-only algorithm running on the same hardware, and speedups of 2–6× for calculations using the AIFDEM on real systems. For CIS and TDDFT calculations where numerous excited states are required, as in a sum-over-states approach to circular dichroism spectroscopy or simply for a system such as cobalt(II)cobalamin with a high density of excited states, where 100+ states might be required to compute an absorption spectrum, we have also demonstrated speedups of up to 5×. Larger speedups as compared to a CPU-only algorithm are obtained when the basis set is augmented with diffuse functions (and for Pople-type basis sets in particular), and in any circumstance that increases the amount of digestion work, such as the use of tighter thresholds or calculation of a larger number of excited states.

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Chemical Sciences, Geosciences, and Biosciences under Award No. DE-SC0008550. Calculations were performed at the Ohio Supercomputer Center under project no. PAA-0003.[77] A.F.M. acknowledges a fellowship from the Lubrizol Corporation. J.M.H. is a fellow of the Alexander von Humboldt Foundation and serves on the Board of Directors of Q-Chem Inc.

Keywords: quantum chemistry · excited states · graphics processing units · TDDFT

How to cite this article: Adrian F. Morrison, E. Epifanovsky, John M. Herbert. *J. Comput. Chem.* **2018**, 39, 2173–2182. DOI: 10.1002/jcc.25531

[1] A. F. Morrison, Z.-Q. You, J. M. Herbert, J. Chem. Theory Comput. **2014**, 10, 5366.
[2] A. F. Morrison, J. M. Herbert, J. Phys. Chem. Lett. **2015**, 6, 4390.
[3] A. F. Morrison, J. M. Herbert, J. Phys. Chem. Lett. **2017**, 8, 1442.
[4] Morrison, A. F.; Herbert, J. M. J. Chem. Phys. **2017**, 146, 224110.
[5] J. M. Herbert, X. Zhang, A. F. Morrison, J. Liu, Acc. Chem. Res. **2016**, 49, 931.
[6] J. Frenkel, Phys. Rev. **1931**, 37, 17.
[7] A. S. Davydov, Theory of Molecular Excitons, McGraw-Hill, New York, **1962**.
[8] M. Kasha, Radiat. Res. **1963**, 20, 55.
[9] M. Kasha, H. R. Rawls, M. A. El-Bayoumi, Pure Appl. Chem. **1965**, 11, 371.
[10] I. S. Ufimtsev, N. Luehr, T. J. Martinez, J. Phys. Chem. Lett. **2011**, 2, 1789.
[11] H. J. Kulik, N. Luehr, I. S. Ufimtsev, T. J. Martinez, J. Phys. Chem. B **2012**, 116, 12501.
[12] B. S. Fales, B. G. Levine, J. Chem. Theory Comput. **2015**, 11, 4708.
[13] D. van der Spoel, B. Hess, WIREs Comput. Mol. Sci. **2011**, 1, 710.
[14] R. C. Walker, A. W. Götz, Eds., Electronic Structure Calculations on Graphics Processing Units, Wiley, Chichester, UK, **2016**.
[15] K. Yasuda, J. Comput. Chem. **2008**, 29, 334.
[16] I. S. Ufimtsev, T. J. Martínez, J. Chem. Theory Comput. **2008**, 4, 222.
[17] C. Song, L.-P. Wang, T. Sachse, J. Preiß, M. Presselt, T. J. J. Martínez, Chem. Phys. **2015**, 143, 014114.
[18] Luehr, N.; Sisto, A.; Martínez, T. J. In Electronic Structure Calculations on Graphics Processing Units; Walker, R. C., Götz, A. W., Eds.; Wiley: Chichester, UK, **2016**; chapter 4, pages 67–100.
[19] C. Song, L.-P. Wang, T. J. Martínez, J. Chem. Theory Comput. **2016**, 12, 92.
[20] A. Asadchev, V. Allada, J. Felder, B. M. Bode, M. S. Gordon, T. L. Windus, J. Chem. Theory Comput. **2010**, 6, 696.
[21] Y. Miao, K. M. Merz, J. Chem. Theory Comput. **2015**, 11, 1449.
[22] J. Kalinowski, F. Wennmohs, F. Neese, J. Chem. Theory Comput. **2017**, 13, 3160.
[23] C. A. Renison, K. D. Fernandes, K. J. Naidoo, J. Comput. Chem. **2015**, 36, 1410.
[24] K. D. Fernandes, C. A. Renison, K. J. Naidoo, J. Comput. Chem. **2015**, 36, 1399.
[25] K. Yasuda, J. Chem. Theory Comput. **2008**, 4, 1230.
[26] I. S. Ufimtsev, T. J. Martinez, J. Chem. Theory Comput. **2009**, 5, 1004.
[27] I. S. Ufimtsev, T. J. Martinez, J. Chem. Theory Comput. **2009**, 5, 2619.
[28] Wu, X.; Koslowski, A.; Thiel, W. In Electronic Structure Calculations on Graphics Processing Units; Walker, R. C., Götz, A. W., Eds.; Wiley: Chichester, UK, **2016** 239–258.
[29] T. Yoshikawa, H. Nakai, J. Comput. Chem. **2015**, 36, 164.
[30] C. M. Isborn, N. Luehr, I. S. Ufimtsev, T. J. Martínez, J. Chem. Theory Comput. **2011**, 7, 1814.
[31] P. M. W. Gill, J. A. Pople, Int. J. Quantum Chem. **1991**, 40, 753.
[32] P. M. W. Gill, Adv. Quantum Chem. **1994**, 25, 141.
[33] T. R. Adams, R. D. Adamson, P. M. W. Gill, J. Chem. Phys. **1997**, 107, 124.
[34] L. Vogt, R. Olivares-Amaya, S. Kermes, Y. Shao, C. Amador-Bedolla, A. Aspuru-Guzik, J. Phys. Chem. A **2008**, 112, 2049.
[35] R. Olivares-Amaya, M. A. Watson, R. G. Edgar, L. Vogt, Y. Shao, A. Aspuru-Guzik, J. Chem. Theory Comput. **2010**, 6, 135.
[36] Olivares-Amaya, R.; Jinich, A.; Watson, M. A.; Aspuru-Guzik, A. In Electronic Structure Calculations on Graphics Processing Units; Walker, R. C., Götz, A. W., Eds.; Wiley: Chichester, UK, **2016**; 259–278.
[37] D. G. Tomlinson, A. Asadchev, M. S. Gordon, J. Comput. Chem. **2016**, 37, 1274.
[38] Song, C.; Martinez, T. J. J. Chem. Phys. **2016**, 144, 174111.
[39] R. M. Parrish, K. C. Thompson, T. J. Martínez, J. Chem. Theory Comput. **2018**, 14, 1737.
[40] W. Ma, S. Krishnamoorthy, K. Kowalski, J. Chem. Theory Comput. **2011**, 7, 1316.
[41] A. Asadchev, M. S. Gordon, J. Chem. Theory Comput. **2013**, 9, 3385.
[42] De Prince III, A. E.; Hammond, J. R.; Sherrill, C. D. In Electronic Structure Calculations on Graphics Processing Units; Walker, R. C., Götz, A. W., Eds.; Wiley: Chichester, UK, **2016**; 279–300.
[43] Ma, W.; Bhaskaran-Nair, K.; Villa, O.; Tumeo, E. A. A.; Krishnamoorthy, S.; Kowalski, K. In Electronic Structure Calculations on Graphics Processing Units; Walker, R. C., Götz, A. W., Eds.; Wiley: Chichester, UK, **2016**; 301–326.
[44] Hohenstein, E. G.; Luehr, N.; Ufimtsev, I. S.; Martínez, T. J. J. Chem. Phys. **2015**, 142, 224103.
[45] Snyder Jr., J. W.; Hohenstein, E. G.; Luehr, N.; Martínez, T. J. J. Chem. Phys. **2015**, 143, 154107.
[46] Snyder, J. W.; Fales, B. S.; Hohenstein, E. G.; Levine, B. G.; Martínez, T. J. J. Chem. Phys. **2017**, 146, 174113.

[47] Fales, B. S.; Shu, Y.; Levine, B. G.; Hohenstein, E. G. J. Chem. Phys. **2017**, 147, 094104.
[48] E. Epifanovsky, M. Wormit, T. Kuś, A. Landau, D. Zuev, K. Khistyaev, P. Manohar, I. Kaliman, A. Dreuw, A. I. Krylov, J. Comput. Chem. **2013**, 34, 2293.
[49] I. A. Kaliman, A. I. Krylov, J. Comput. Chem. **2017**, 38, 842.
[50] A. Dreuw, M. Head-Gordon, Chem. Rev. **2005**, 105, 4009.
[51] M. Gao, S. Paul, C. D. Schwieters, Z.-Q. You, H. Shao, J. M. Herbert, J. R. Parquette, C. P. Jaroniec, J. Phys. Chem. C **2015**, 119, 13948.
[52] K. B. Wiberg, Y. Wang, S. M. Wilson, P. H. Vaccaro, J. R. Cheeseman, J. Phys. Chem. A **2006**, 110, 13995.
[53] M. Seth, J. Autschbach, T. Ziegler, J. Chem. Theory Comput. **2007**, 3, 434.
[54] P. Štěpánek, P. Bouř, J. Comput. Chem. **2015**, 36, 723.
[55] P. Štěpánek, P. Bouř, J. Comput. Chem. **2013**, 34, 1531.
[56] S. Tonzani, G. C. Schatz, J. Am. Chem. Soc. **2008**, 130, 7607.
[57] Kaminský, J.; Kříž, J.; Bouř, P. J. Chem. Phys. **2017**, 146, 144301.
[58] Y. Shao, Z. Gan, E. Epifanovsky, A. T. B. Gilbert, M. Wormit, J. Kussmann, A. W. Lange, A. Behn, J. Deng, X. Feng, D. Ghosh, M. Goldey, P. R. Horn, L. D. Jacobson, I. Kaliman, R. Z. Khaliullin, T. Kús, A. Landau, J. Liu, E. I. Proynov, Y. M. Rhee, R. M. Richard, M. A. Rohrdanz, R. P. Steele, E. J. Sundstrom, H. L. Woodcock, III., P. M. Zimmerman, D. Zuev, B. Albrecht, E. Alguire, B. Austin, G. J. O. Beran, Y. A. Bernard, E. Berquist, K. Brandhorst, K. B. Bravaya, S. T. Brown, D. Casanova, C.-M. Chang, Y. Chen, S. H. Chien, K. D. Closser, D. L. Crittenden, M. Diedenhofen, R. A. DiStasio, Jr., H. Do, A. D. Dutoi, R. G. Edgar, S. Fatehi, L. Fusti-Molnar, A. Ghysels, A. Golubeva-Zadorozhnaya, J. Gomes, M. W. D. Hanson-Heine, P. H. P. Harbach, A. W. Hauser, E. G. Hohenstein, Z. C. Holden, T.-C. Jagau, H. Ji, B. Kaduk, K. Khistyaev, J. Kim, J. Kim, R. A. King, P. Klunzinger, D. Kosenkov, T. Kowalczyk, C. M. Krauter, K. U. Lao, A. Laurent, K. V. Lawler, S. V. Levchenko, C. Y. Lin, F. Liu, E. Livshits, R. C. Lochan, A. Luenser, P. Manohar, S. F. Manzer, S.-P. Mao, N. Mardirossian, A. V. Marenich, S. A. Maurer, N. J. Mayhall, C. M. Oana, R. Olivares-Amaya, D. P. O'Neill, J. A. Parkhill, T. M. Perrine, R. Peverati, P. A. Pieniazek, A. Prociuk, D. R. Rehn, E. Rosta, N. J. Russ, N. Sergueev, S. M. Sharada, S. Sharma, D. W. Small, A. Sodt, T. Stein, D. Stück, Y.-C. Su, A. J. W. Thom, T. Tsuchimochi, L. Vogt, O. Vydrov, T. Wang, M. A. Watson, J. Wenzel, A. White, C. F. Williams, V. Vanovschi, S. Yeganeh, S. R. Yost, Z.-Q. You, I. Y. Zhang, X. Zhang, Y. Zhao, B. R. Brooks, G. K. L. Chan, D. M. Chipman, C. J. Cramer, W. A. Goddard, III., M. S. Gordon, W. J. Hehre, A. Klamt, H. F. Schaefer, III., M. W. Schmidt, C. D. Sherrill, D. G. Truhlar, A. Warshel, X. Xu, A. Aspuru-Guzik, R. Baer, A. T. Bell, N. A. Besley, J.-D. Chai, A. Dreuw, B. D. Dunietz, T. R. Furlani, S. R. Gwaltney, C.-P. Hsu, Y. Jung, J. Kong, D. S. Lambrecht, W. Liang, C. Ochsenfeld, V. A. Rassolov, L. V. Slipchenko, J. E. Subotnik, T. Van Voorhis, J. M. Herbert, A. I. Krylov, P. M. W. Gill, M. Head-Gordon, Mol. Phys. **2015**, 113, 184.
[59] R. A. Di Stasio, Jr., R. P. Steele, M. Head-Gordon, Mol. Phys. **2007**, 105, 2731.
[60] Grimme, S.; Neese, F. J. Chem. Phys. **2007**, 127, 154116.
[61] McKemmish, L. K. J. Chem. Phys. **2015**, 142, 134104.
[62] R. A. Kendall, H. A. Früchtl, Theor. Chem. Acc. **1997**, 97, 158.
[63] M. Feyereisen, G. Fitzgerald, A. Komornicki, Chem. Phys. Lett. **1993**, 208, 359.
[64] G. R. Ahmadi, J. Almlöf, Chem. Phys. Lett. **1995**, 246, 364.
[65] R. Bauernschmitt, M. Häser, O. Treutler, R. Ahlrichs, Chem. Phys. Lett. **1997**, 264, 573.
[66] F. Weigend, M. Häser, J. Patzelt, R. Ahlrichs, Chem. Phys. Lett. **1998**, 294, 143.
[67] F. Neese, G. Olbrich, Chem. Phys. Lett. **2002**, 362, 170.
[68] X. Ren, P. Rinke, V. Blum, J. Wieferink, A. Tkatchenko, A. Sanfilippo, K. Reuter, M. Scheffler, New J. Phys. **2012**, 14, 053020.
[69] S. F. Manzer, E. Epifanovsky, M. Head-Gordon, J. Chem. Theory Comput. **2015**, 11, 518.
[70] Manzer, S.; Horn, P. R.; Mardirossian, N.; Head-Gordon, M. J. Chem. Phys. **2015**, 143, 024113.
[71] A. Ipatov, A. Fouqueau, C. P. del Valle, F. Cordova, M. E. Casida, A. M. Köster, A. Vela, C. J. Jamorski, J. Mol. Struct. (Theochem) **2006**, 762, 179.
[72] A. Sisto, D. R. Glowacki, T. J. Martinez, Acc. Chem. Res. **2014**, 47, 2857.
[73] A. Sisto, C. Stross, M. W. van der Kamp, M. O'Connor, S. McIntosh-Smith, G. T. Johnson, E. G. Hohenstein, F. R. Manby, D. R. Glowacki, T. J. Martinez, Phys. Chem. Chem. Phys. **2017**, 19, 14924.
[74] X. Li, R. M. Parrish, F. Liu, S. I. L. K. Schumacher, T. J. Martíez, J. Chem. Theory Comput. **2017**, 13, 3493.
[75] A. Szabo, N. S. Ostlund, Modern Quantum Chemistry, Macmillan, New York, **1982**.
[76] Y. Shi, U. N. Niranjan, A. Anandkumar, C. Cecka, IEEE 23rd International Conference on High Performance Computing, Institute of Electrical and Electronics Engineers: Hyderabad, India. **2016**.
[77] Ohio Supercomputer Center. (it's what OSC uses to report its citation statistics to the funding agencies):
[78] R. E. Stratmann, G. E. Scuseria, M. J. Frisch, J. Chem. Phys. **1998**, 109, 8218.
[79] E. R. Davidson, J. Comput. Phys. **1975**, 17, 87.
[80] T. A. Stich, N. R. Buan, T. C. Brunold, J. Am. Chem. Soc. **2004**, 126, 9735.
[81] K. Kornobis, N. Kumar, B. M. Wong, P. Lodowski, M. Jaworska, T. Andruniów, K. Ruud, P. M. Kozlowski, J. Phys. Chem. A **2011**, 115, 1280.
[82] H. Solheim, K. Kornobis, K. Ruud, P. M. Kozlowski, J. Phys. Chem. B **2011**, 115, 737.

Received: 4 March 2018
Revised: 12 May 2018
Accepted: 14 June 2018