JCTC Journal of Chemical Theory and Computation

Accelerating Correlated Quantum Chemistry Calculations Using Graphical Processing Units and a Mixed Precision Matrix Multiplication Library

Roberto Olivares-Amaya,[†,§] Mark A. Watson,[†,§] Richard G. Edgar,[†] Leslie Vogt,[†] Yihan Shao,[‡] and Alán Aspuru-Guzik*[,†]

Department of Chemistry and Chemical Biology, Harvard University, 12 Oxford Street, Cambridge, Massachusetts 02138 and Q-Chem, Inc., 5001 Baum Boulevard, Suite 690, Pittsburgh, Pennsylvania 15213

Received October 13, 2009

Abstract: Two new tools for the acceleration of computational chemistry codes using graphical processing units (GPUs) are presented. First, we propose a general black-box approach for the efficient GPU acceleration of matrix−matrix multiplications where the matrix size is too large for the whole computation to be held in the GPU's onboard memory. Second, we show how to improve the accuracy of matrix multiplications when using only single-precision GPU devices by proposing a heterogeneous computing model, whereby single- and double-precision operations are evaluated in a mixed fashion on the GPU and central processing unit, respectively. The utility of the library is illustrated for quantum chemistry with application to the acceleration of resolution-of-the-identity second-order Møller−Plesset perturbation theory calculations for molecules, which we were previously unable to treat. In particular, for the 168-atom valinomycin molecule in a cc-pVDZ basis set, we observed speedups of 13.8, 7.8, and 10.1 times for single-, double- and mixed-precision general matrix multiply (SGEMM, DGEMM, and MGEMM), respectively. The corresponding errors in the correlation energy were reduced from −10.0 to −1.2 kcal mol^{-1} for SGEMM and MGEMM, respectively, while higher accuracy can be easily achieved with a different choice of cutoff parameter.

1. Introduction

Ever since scientists began to solve the equations of molecular quantum mechanics using numerical methods and computational tools, the interplay between fundamental theory and application has been inextricably linked to exponential advances in hardware technology. Indeed, many influential contributions to quantum chemistry have been motivated by insights into how best to utilize the available computational resources within the same theoretical model. One example is Almlöf's appreciation of the discrepancy that had appeared between data storage capacity and raw processor speed.[1] His subsequent introduction of the direct SCF technique transformed calculations from being memory (or disk) bound into being processor bound; previously impossible applications could be attempted by using additional processor time.

We are now witnessing yet another era in the optimization of quantum chemistry codes, following an explosion of interest in the application of coprocessors, such as graphics processing units (GPUs) to general scientific computing.[2] This interest in GPUs and related massively parallel processors is largely driven by their tremendous cost to performance ratio (in operation counts per second per unit of currency), which arises from the economies of scale in their manufacture and in their great demand in numerous multimedia applications. Another key factor in their widespread uptake for scientific use is the recent release of NVIDIA's compute unified device architecture (CUDA) programming interface

* Corresponding author. E-mail: aspuru@chemistry.harvard.edu.
[†] Harvard University.
[‡] Q-Chem, Inc.
[§] Contributed equally to this work.

that allows development of algorithms for the GPU using a relatively simple extension of the standard C language.[2]

A GPU is an example of a stream-processing architecture[3] and can outperform a general-purpose central processing unit (CPU) for certain tasks because of the intrinsic parallelization within the device, which uses the single-instruction, multiple data (SIMD) paradigm. Typical GPUs contain multiple arithmetic units (streaming processors), which are typically arranged in groups of eight to form multiprocessors that share a fast access memory and an instruction unit; all eight processors execute the same instruction thread simultaneously on different data streams. In contrast, in multiple-core or parallel CPU architectures, each thread must have an instruction explicitly coded for each piece of data. One of the most recent GPU cards, the Tesla C1060 from NVIDIA, contains 240 streaming processors, can provide up to 933 GFLOPS of single-precision computational performance, and has a cost which is approximately 1 order of magnitude less than an equivalent CPU cluster.

GPUs are, therefore, well-suited for high-performance applications with dense levels of data parallelism where very high accuracy is not required. (Although double-precision cards are available, in the case of NVIDIA GPUs, they have a peak FLOP count approximately 10 times less than those of single precision cards.) The challenge for scientists wanting to exploit the efficiency of the GPU is to expose the SIMD parallelism in their problem and to efficiently implement it on the new architecture. A key component of this task is a careful consideration of the memory hierarchy to efficiently hide memory access latency.

Already, GPUs have been recruited extensively by the scientific community to treat a wide range of problems, including finite-difference time-domain algorithms[4] and *n*-body problems in astrophysics.[5] For computational chemistry, GPUs are emerging as an extremely promising architecture for molecular dynamics simulations,[6,7] quantum Monte Carlo,[8] density functional theory, self-consistent field calculations,[9−14] and correlated quantum chemistry[15] methods. Efficiency gains of between one and three orders of magnitude using NVIDIA graphics cards have been reported compared to conventional implementations on a CPU. In this way, new domains of scientific application have become amenable to calculation where, previously, extremely expensive and rare supercomputing facilities would have been required.

As an example of the more general impact of accelerator technologies, Brown et al.[16] have accelerated density functional theory up to an order of magnitude using a Clearspeed coprocessor. The Clearspeed hardware is a proprietary compute-oriented stream architecture promising raw performance comparable to that of modern GPUs, while offering double-precision support and an extremely low power consumption. The challenges of efficiently utilizing the Clearspeed boards are similar to those of GPUs, requiring a fine-grained parallel programming model with a large number of lightweight threads. Thus, the algorithmic changes suggested for their work and for ours have a common value independent of the precise hardware used, which will of course change with time.

In the current work, we introduce two new techniques with general utility for the adoption of GPUs in quantum chemistry. First, we propose a general approach for the efficient GPU acceleration of matrix−matrix multiplications, where the matrix size is too large for the whole computation to be held in the GPU's onboard memory, requiring the division of the original matrices into smaller pieces. This is a major issue in quantum chemical calculations where matrix sizes can be very large.

Second, we describe how to improve the accuracy of general matrix−matrix multiplications when using single-precision GPUs, where the 6−7 significant figures are often insufficient to achieve 'chemical accuracy' of 1 kcal mol^{-1}. To solve this problem, we have implemented a new algorithm within a heterogeneous computing model, whereby the numerically large contributions to the final result are computed and accumulated on a double-precision device (typically the CPU), and the remaining small contributions are efficiently treated by the single-precision GPU device.

We have applied these ideas in an extension of our previously published GPU-enabled implementation of resolution-of-the-identity second-order Møller−Plesset perturbation theory (RI-MP2).[17−20] Thus, the paper begins in Section 2 with an overview of the RI-MP2 method and our previous GPU implementation. In Sections 3 and 4, we discuss our new matrix-multiplication library and its performance. In Section 5, we examine the accuracy and the speedups achieved when applying the technology to RI-MP2 calculations on molecules with up to 168 atoms, and we end the paper with some brief conclusions.

2. GPU Acceleration of RI-MP2

One of the most widely used and computationally least expensive correlated treatments for electronic structure is the second-order Møller−Plesset perturbation theory (MP2). MP2 is known to produce equilibrium geometries of comparable accuracy to density functional theory (DFT),[21] but unlike many popular DFT functionals is able to capture long-range correlation effects, such as the dispersion interaction. For many weakly bound systems where DFT results are often questionable, MP2 is essentially the least expensive and most reliable alternative.[22] The expression for computing the MP2 correlation energy takes the form:

$$E^{(2)} = \sum_{ijab} \frac{(ia|jb)^2 + \frac{1}{2}[(ia|jb) - (ib|ja)]^2}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b} \qquad (1)$$

in terms of the $\{i, j\}$ occupied and $\{a, b\}$ virtual molecular orbitals (MOs) that are eigenfunctions of the Fock operator with eigenvalues $\{\varepsilon\}$. The MO integrals:

$$(ij|ab) = \sum_{\mu\nu\lambda\sigma} C_{\mu i} C_{\nu j} C_{\lambda a} C_{\sigma b} (\mu\nu|\lambda\sigma) \qquad (2)$$

are obtained by contracting two-electron integrals over the (real) atomic orbital (AO) basis functions:

$$(\mu\nu|\lambda\sigma) = \int\int \phi_\mu(\mathbf{r}_1)\phi_\nu(\mathbf{r}_1)\phi_\lambda(\mathbf{r}_2)\phi_\sigma(\mathbf{r}_2)\mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2 \qquad (3)$$

where $\mathbf{C}$ is the matrix of MO coefficients describing the expansion of each MO as a linear combination of the AOs. One way to considerably reduce the computational cost associated with traditional MP2 calculations (which formally scales as $O(N^5)$ with the number of basis functions) is to exploit the linear dependence inherent in the product space of atomic orbitals. This allows one to expand products of AOs as linear combinations of atom-centered auxiliary basis functions, P:

$$\rho_{\mu\nu}(\mathbf{r}) = \mu(\mathbf{r})\nu(\mathbf{r}) \approx \tilde{\rho}_{\mu\nu}(\mathbf{r}) = \sum_P C_{\mu\nu,P}P(\mathbf{r}) \tag{4}$$

and to, therefore, approximate all the costly four-center, two-electron integrals in terms of only two- and three-center integrals:

$$\widetilde{(\mu\nu|\lambda\sigma)} = \sum_{P,Q} (\mu\nu|P)(P|Q)^{-1}(Q|\lambda\sigma) \tag{5}$$

where we have assumed that the expansion coefficients are determined by minimizing the Coulomb self-repulsion of the residual density. The result is equivalent to an approximate insertion of the resolution-of-the-identity (RI).

All our work is implemented in a development version of Q-Chem 3.1,[23] where the RI-MP2 correlation energy is evaluated in five steps, as described elsewhere.[15] Previously we showed that step 4, the formation of the approximate MO integrals, was by far the most expensive operation for medium- to large-sized systems and requires the matrix multiplication:

$$\widetilde{(ia|jb)} \approx \sum_Q B_{ia,Q}B_{jb,Q} \tag{6}$$

where:

$$B_{ia,Q} = \sum_P (ia|P)(P|Q)^{-1/2} \tag{7}$$

The evaluation of eq 6 is typically an order of magnitude more expensive than eq 7. We shall concentrate on these two matrix multiplications in this work. Consistent with our previous paper,[15] we will repeatedly refer to these evaluations as steps 3 (eq 7) and 4 (eq 6) as we investigate the accuracy and the efficiency of our new GPU implementation.

Included in the CUDA software development toolkit is an implementation of the BLAS linear algebra library, named CUBLAS.[24] As previously reported,[15] we accelerated the matrix multiplication in eq 6 by simply replacing the BLAS *GEMM routines with corresponding calls to CUBLAS SGEMM. This initial effort achieved an overall speedup of 4.3 times for the calculation of the correlation energy of the 68-atom doeicosane ($C_{22}H_{46}$) molecule with a cc-pVDZ basis set using a single GPU. At this early stage in development, we used the GPU purely as an accelerator for *GEMM and made no effort to keep data resident on the device.

In the present work, we further explore the acceleration of our RI-MP2 code through the application of CUBLAS combined with two new techniques. These enable us to perform more accurate calculations on larger molecules and basis sets involving larger matrices, while also mitigating the errors associated with single-precision GPUs. We discuss both techniques in the following section.

3. GPU Acceleration of GEMM

In large-scale quantum chemistry calculations, the size of the fundamental matrices typically grows as the square of the number of atomic basis functions (even if the number of non-negligible elements is much smaller). Moreover, intermediate matrices are sometimes even larger, such as the **B** matrices of eq 7.

A GPU can only accelerate a calculation that fits into its onboard memory. While the most modern cards designed for research can have up to 4 GiB of RAM, consumer level cards may have as little as 256 MiB (with some portion possibly devoted to the display). If we wish to run large calculations, but only have a small GPU available, then some means of dividing the calculation up and staging it through the GPU must be found.

Next, we consider the question of accuracy arising from the use of single-precision GPU cards. It turns out, that many operations do not require full double-precision support to achieve acceptable accuracy for chemistry, but nevertheless, single precision is not always sufficient.[13] Double-precision capable GPUs have only become available within the past year and so are not yet widespread. Moreover, we cannot rely on the support of double-precision cards by manufacturers in the future, since the commercial driving force behind such processors is the wealth of multimedia applications that do not require high precision. We address this problem with the introduction of a new way to balance the desire for GPU acceleration with a need for high accuracy.

3.1. Cleaving GEMMs. Consider the matrix multiplication:

$$\mathbf{C} = \mathbf{A}\cdot\mathbf{B} \tag{8}$$

where **A** is a $(m \times k)$ matrix, and **B** is a $(k \times n)$ matrix, making **C** an $(m \times n)$ matrix. We can divide **A** into a column vector of $r + 1$ matrices:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_r \end{pmatrix} \tag{9}$$

where each entry $\mathbf{A}_i$ is a $(p_i \times k)$ matrix, and $\sum_{i=0}^{r} p_i = m$. In practice, all the p_i will be the same, with the possible exception of p_r, which will be an edge case. In a similar manner, we can divide **B** into a row vector of $s + 1$ matrices:

$$\mathbf{B} = (\mathbf{B}_0 \quad \mathbf{B}_1 \quad \cdots \quad \mathbf{B}_s) \tag{10}$$

where each $\mathbf{B}_j$ is an $(k \times q_j)$ matrix, and $\sum_{j=0}^{s} q_j = n$. Again all the q_j will be the same, with the possible exception of q_s. We then form the outer product of these two vectors:

$$\mathbf{C} = \begin{pmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_r \end{pmatrix} \cdot (\mathbf{B}_0 \quad \mathbf{B}_1 \quad \cdots \quad \mathbf{B}_s) \tag{11}$$

$$= \begin{pmatrix} \mathbf{A}_0\cdot\mathbf{B}_0 & \mathbf{A}_0\cdot\mathbf{B}_1 & \cdots & \mathbf{A}_0\cdot\mathbf{B}_s \\ \mathbf{A}_1\cdot\mathbf{B}_0 & \mathbf{A}_1\cdot\mathbf{B}_1 & & \mathbf{A}_1\cdot\mathbf{B}_s \\ \vdots & & \ddots & \\ \mathbf{A}_r\cdot\mathbf{B}_0 & & & \mathbf{A}_r\cdot\mathbf{B}_s \end{pmatrix} \tag{12}$$

Each individual $\mathbf{C}_{ij} = \mathbf{A}_i\mathbf{B}_j$ is a $(p_i \times q_j)$ matrix and can be computed independently of all the others. Generalizing this to a full *GEMM implementation, which includes the possibility of transposes being taken, is tedious but straightforward.

We have implemented this approach for the GPU, as a complete replacement for *GEMM. The p_i and q_j values are chosen such that each submultiplication fits within the currently available GPU memory. Each multiplication is staged through the GPU, and the results assembled on the CPU. This process is hidden from the user code, which simply sees a standard *GEMM call.

3.2. Heterogeneous Computing with MGEMM. With the problem of limited memory solved, we will now demonstrate how to overcome the lack of double precision GPU hardware. Again, consider the matrix multiplication:

$$\mathbf{C} = \mathbf{A}\cdot\mathbf{B} \tag{13}$$

We can split each matrix element-wise into ‘large’ and ‘small’ components, giving:

$$\begin{aligned}\mathbf{C} &= (\mathbf{A}^{\text{large}} + \mathbf{A}^{\text{small}})(\mathbf{B}^{\text{large}} + \mathbf{B}^{\text{small}})\\ &= \mathbf{A}\cdot\mathbf{B}^{\text{large}} + \mathbf{A}^{\text{large}}\cdot\mathbf{B}^{\text{small}} + \mathbf{A}^{\text{small}}\cdot\mathbf{B}^{\text{small}}\end{aligned}$$

The $\mathbf{A}^{\text{small}}\mathbf{B}^{\text{small}}$ term consists entirely of ‘small’ numbers and can be run in single precision on the GPU (using the cleaving approach described above, if needed). The other two terms contain ‘large’ numbers and need to be run in double precision. However, since each of the ‘large’ matrices should be sparse, these terms each consist of a dense-sparse multiplication. We only store the nonzero terms of the $\mathbf{A}^{\text{large}}$ and $\mathbf{B}^{\text{large}}$ matrices, cutting the computational complexity significantly. Consider:

$$C'_{ik} = A_{ij}B_{jk}^{\text{large}} \tag{14}$$

Only a few B_{jk}^{large} will be nonzero, and we consider each in turn. For a particular scalar B_{jk}^{large}, only the kth column of $\mathbf{C}'$ will be nonzero and is equal to the product of B_{jk}^{large} and the jth column vector of $\mathbf{A}$. This nonzero column vector C'_{ik} can be added to the final result, $\mathbf{C}$, and the next B_{jk}^{large} can be considered. A similar process can be applied to the $\mathbf{A}^{\text{large}}\mathbf{B}^{\text{small}}$ term (producing *row* vectors of $\mathbf{C}$). Again, this approach can be generalized to a full *GEMM implementation, including transposes.

The remaining question is that of splitting the matrices. We have taken the simple approach of defining a cutoff value, δ. If $|A_{ij}| > \delta$, that element is considered ‘large,’ otherwise it is considered to be ‘small.’

We have implemented our algorithm we have dubbed MGEMM for mixed-precision general matrix multiply. It operates similarly to the other *GEMM routines but takes one extra argument—the value of δ.

4. MGEMM Benchmarks

We will now discuss some benchmarks for MGEMM. Our aim is to assess the speed and accuracy of MGEMM for various matrix structures and the choice of cutoff tolerance compared to a DGEMM call on the CPU. In particular, it is important to benchmark how much computational speed is gained using the mixed-precision MGEMM with the GPU as a function of the loss in accuracy compared to DGEMM.

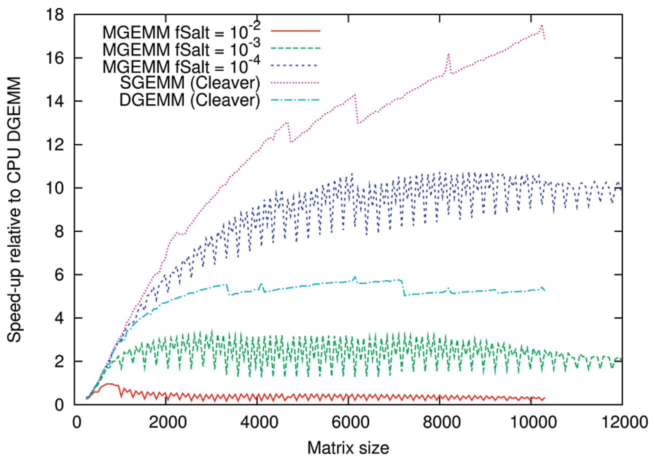

Figure 1. Speedup for various *GEMM calls as a function of (square) matrix size (averaged over 10 runs). Most elements were in the range [−1, 1], with the ‘salt’ values in the range [90, 110]. Times are scaled relative to running DGEMM on the CPU.

Throughout this section, CPU calculations were made using an Intel Xeon E5472 (Harpertown) processor clocked at 300 GHz attached to an NVIDIA Tesla C1060 (packaged into a Tesla S1070). The GPU calls were limited to 256 MiB of RAM to model a more restricted GPU in a typical BOINC (Berkeley Open Infrastructure for Network Computing) client.[25,26]

4.1. Using Model Matrices. In Figure 1 we show the speedup for a variety of *GEMM calls using matrices of increasing (square) size. Three different types of matrices were considered, based on the number of randomly scattered ‘large’ elements. All the matrices were initialized with random values in the range [−1, 1], forming the ‘background’ and ‘salted’ with a fraction f_{salt} of random larger values in the range [90, 110]. The size of the MGEMM cutoff parameter δ was chosen such that all the salted elements were considered ‘large’.

There are three MGEMM curves plotted, for different values of $f_{\text{salt}} = 10^{-2}$, 10^{-3}, and 10^{-4}. The SGEMM(cleaver) curve corresponds to doing the full matrix multiplication on the GPU using the GEMM(cleaver) and includes the time taken to down convert the matrices to single precision on the CPU. The DGEMM(cleaver) curve corresponds to a full double-precision matrix multiplication on the GPU, which is possible for modern cards, and we include it for completeness. Square matrices were used in all cases, with no transpositions in the *GEMM calls. All the runs were performed 10 times, and speedups are obtained relative to the time taken for the corresponding DGEMM call on the CPU.

Examining the results, we see that SGEMM on the GPU gives a speedup of 17.1 times over running DGEMM on the CPU for a matrix of size 10 048 × 10 048 and is even faster for larger matrices. This represents an upper bound for the speedups we can hope to obtain with MGEMM for such matrices. The speedups increase significantly as the matrices become larger due to the masking of memory access latencies and due to other overheads when employing the GPU for more compute-intensive processes.

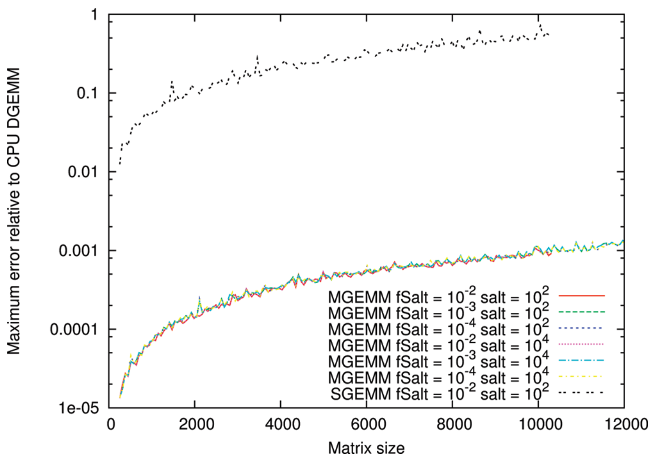

Figure 2. Maximum absolute error in a single element for various GEMM calls as a function of matrix (square) size. Most elements were in the range [−1, 1], with the 'salt' values in the range [90, 110] or [9 990, 10 010]. A CPU DGEMM call was taken as the reference calculation.

Considering the MGEMM results, we see that the speedups are strongly dependent on the number of large elements which must be evaluated in double precision on the CPU. For the relatively high value of $f_{\text{salt}} = 10^{-2}$, running MGEMM was actually slower than running DGEMM on the CPU alone. This is understandable when one considers the extra steps in the MGEMM algorithm. In addition to down converting the matrices to single precision, the CPU has to perform cache-incoherent operations on the 'large' multiplications. We store our matrices column major, so the operations performed in eq 14 are cache coherent. However, it is easy to see that the corresponding operations for $\mathbf{C}' = \mathbf{A}^{\text{large}}\mathbf{B}^{\text{small}}$ will be cache incoherent for both $\mathbf{C}'$ and $\mathbf{B}^{\text{small}}$ (recall that $\mathbf{A}^{\text{large}}$ will be stored as individual elements). This brings a huge penalty over a standard *GEMM implementation, which is tiled for cache coherency.

In contrast, for $f_{\text{salt}} = 10^{-4}$, there is much less penalty to running MGEMM over SGEMM on the GPU, due to the small fraction of large elements computed on the CPU. Speedups of approximately 10 times are observed for the largest matrices. For $f_{\text{salt}} = 10^{-3}$, the performance is naturally reduced, and speedups of approximately 2 times relative to CPU DGEMM are obtained for the largest matrices. In this case, MGEMM runs approximately 2.5 times slower than full DGEMM on the GPU (available in the most modern cards). We may also note that the thresholds for matrix cleaving can be discerned. They start at matrix sizes of 3 344 for double precision and 4 729 for single precision. These are detectable on the curves but do not alter the times significantly.

In Figure 2, we examine the accuracy of MGEMM for various matrix structures. Shown in the figure are the maximum absolute errors of a single element (relative to the CPU DGEMM result) plotted as a function of matrix size, for different fractions f_{salt} and sizes of salted values. As before, all the matrices were initialized with random values in the range [−1, 1], but now the salting sizes were grouped into two ranges: [90, 110] and [9 990, 10 010]. There is one curve using SGEMM corresponding to a fraction of salted values, $f_{\text{salt}} = 0.01$, in the range [90, 110], and several MGEMM curves.

Looking at the figure, we see that the salted SGEMM calculation produces substantial errors for the largest matrices, which are of the same order of magnitude as the background elements themselves. In contrast, the errors are significantly reduced when using MGEMM and are the same regardless of the fraction or size of the salted elements. In fact, these limiting MGEMM errors are the same as the errors observed when using SGEMM on a pair of unsalted random matrices. Essentially, MGEMM is limiting the maximum error in any element to that of the 'background' matrix computed in single precision, since the cutoff tolerance guarantees that all the salted contributions will be computed in double precision on the CPU.

The order of magnitude of the limiting error can be rationalized from a consideration of the number of single-precision contributions per output element (approximately 1 000−10 000 in this case) and the expected error in each (approximately 10^{-6}−10^{-7} for input matrices with a random background on [−1, 1]). A consequence of this observation is that an upper bound to the maximum error can be estimated from a consideration of only the matrix size and the cutoff parameter δ, although this estimate will be very conservative in cases where there is no obvious 'constant background', as we shall see in the following.

4.2. Using RI-MP2 Matrices. For a more realistic assessment of MGEMM for quantum chemistry applications, we also ran benchmarks on two pairs of matrices taken from an RI-MP2 calculation on the taxol molecule in a cc-pVDZ basis, as described below in Section 5. In this case, the MGEMM cutoff parameter δ will no longer be dimensionless but rather will take the same units as the input matrix elements, which, for eqs 6 and 7, are all computed in atomic units. For simplicity, we have dropped these units in the following discussion and assumed their implicit understanding based on the matrices that the δ-value is referring to.

As summarized in Section 2, our RI-MP2 implementation has two steps involving significant matrix multiplications: the evaluation of eqs 6 and 7. As described in the section and as consistent with our previous work,[15] we shall refer to these two matrix multiplications as step 3 (eq 7) and step 4 (eq 6) throughout the following discussion. Although step 3 is typically an order of magnitude faster than step 4, we need to take care to study it since we are interested not only in speed but also error accumulation using MGEMM.

For the case of taxol in a cc-pVDZ basis, the full $(P|Q)^{-1/2}$ matrix is of size 4 186 × 4 186. However, in the Q-Chem implementation, the full $(ia|P)$ and $B_{ia,Q}$ matrices do not need to be explicitly constructed. Instead, it is sufficient to loop over discrete batches of i, depending on available memory. As seen above, larger matrices deliver a greater speedup when multiplied on the GPU, thus, there is a motivation for choosing as large a batch size (over i) as possible in our GPU calculations. In these test benchmarks, we chose batch sizes of one and seven based on the available CPU memory, such that the $(ia|P)$ and $B_{ia,Q}$ matrices have dimensions of 897 × 4 186 and 6 279 × 4 186, respectively. We do not batch the step 3 matrices since there are only $O(N)$

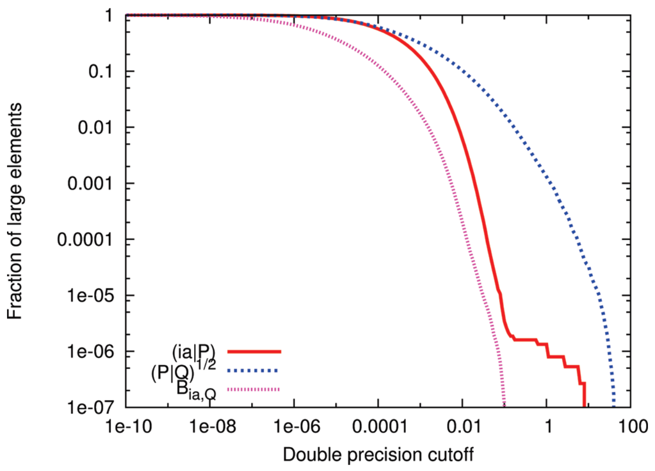

Figure 3. Fraction of 'large' elements as a function of the cutoff parameter, δ, for the taxol RI-MP2 matrices in steps 3 (eq 7) and 4 (eq 6) of the algorithm outlined in Section 2.

multiplications taking place, and the more computationally intensive process is step 4, which has order $O(N^2)$ operations.

We note that the structure of these matrices was found to be very different from the model matrices considered in the previous subsection, Section 4.1. Specifically, the distribution of large and small elements was structured, as described below. In the case of the $(P|Q)^{-1/2}$ matrix, involving only the auxiliary basis set, the large elements were heavily concentrated on the top left-hand corner in a diagonal fashion, while the other matrices were observed to have a striped vertical pattern of large elements. In the current implementation, the main issue affecting the efficiency of MGEMM is the ratio of large to small elements in the input matrices, but in general we can also expect the sparsity structure to impact performance. In cases where the structure is known in advance, a more specialized treatment could give worthwhile speedups, but this is beyond the scope of the current work.

The precise fractions of large and small elements for the taxol case are plotted in Figure 3 with varying cutoff parameter δ for both step 3 and 4 matrices. We should note that these curves are only for one particular i-batch, as explained above, and not for the full matrices. However, to ensure that the results are representative of the full matrix, we checked the distributions from the other batches, and we chose the most conservative matrices for our plots, which had large elements across the broadest range of δ-values.

Looking at the curves, it is significant that the step 3 matrices have a greater fraction of large elements than the step 4 matrices, and specifically, the $(P|Q)^{-1/2}$ matrix has the largest elements of all. This means that for a constant δ-value, we can expect MGEMM to introduce larger errors in the step 3 matrix multiplications than in step 4. In future work, it could be advantageous to tailor the δ-value for different steps in an algorithm, or even different input matrices, but in this first study, we use a constant δ-value throughout any given calculation.

In the model matrices of the previous subsection, Section 4.1, the distribution would have resembled a step function around $\delta = 1.0$, rapidly dropping from 1.0 to the chosen

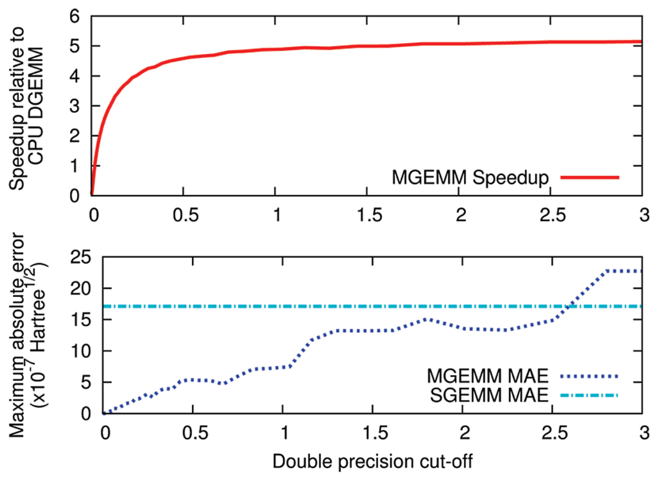

Figure 4. Results from the step 3 (eq 7) matrix multiplication in a taxol RI-MP2 calculation as a function of the cutoff variable δ. Top: MGEMM speedups relative to a CPU DGEMM calculation. Bottom: Maximum absolute error (Hartree$^{1/2}$) in a single element of the output matrix for MGEMM and SGEMM runs.

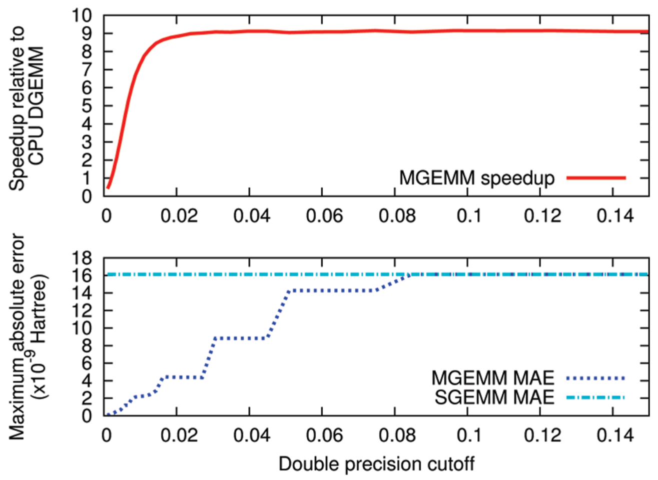

Figure 5. Results from the step 4 (eq 6) matrix multiplication in a taxol RI-MP2 calculation as a function of the cutoff variable δ. Top: MGEMM speedups relative to a CPU DGEMM calculation. Bottom: Maximum absolute error (Hartree) in a single element of the output matrix for MGEMM and SGEMM runs.

fraction of salted values for $\delta > 1.0$ and rapidly stepping again to 0 for δ-values beyond the salt size. In contrast, we see a continuous decay of element values in the real matrices across many orders of magnitude. In Figure 1, MGEMM was seen to outperform DGEMM for a fraction of salts of order 10^{-4}. Comparing to Figure 3, this suggests that δ should be greater than 0.01 to ensure significant MGEMM speedups when considering the $(ia|P)$ and $B_{ia,Q}$ matrices, while the fraction of large elements in the $(P|Q)^{-1/2}$ matrices only becomes this small for δ-values of order 10.

Having analyzed the distributions, we can consider their effect on the accuracy and the speedups compared to those of the model benchmarks. On the top plots of Figures 4 and 5, we show how the speedup for various *GEMM calls (compared to a CPU DGEMM call) varies with δ, averaged over 10 calls. We see that the MGEMM performance varies continuously from being almost the same speed as CPU DGEMM to reaching the GPU SGEMM limit for sufficiently large cutoff values. As expected, for the step 4 matrices, significant speedups are only observed for δ-values greater than approximately 0.01. Similarly, for step 3, the greatest speedups are only observed for much larger δ-values, approximately 1 to 2 orders of magnitude greater than for step 4. The limiting values for the speedups are approximately five and nine times for steps 3 and 4, respectively. This difference is mainly due to the different sizes of the matrices used in each benchmark, recognizing that the smaller matrices used in step 3 will give smaller speedups (cf. Figure 1).

Considering the MGEMM accuracy, the bottom plots in Figures 4 and 5 show the maximum absolute errors of a single element (relative to the CPU DGEMM result) plotted as a function of δ. As δ increases, the MGEMM errors steadily increase, as expected, with the single precision limit being approached for sufficiently large δ. Again we see significant differences between steps 3 and 4, as expected from the element distributions. First, the errors in step 3 are approximately 2 orders of magnitude greater than in step 4. Moreover, in step 4, the errors reach the SGEMM limit for $\delta \sim 0.1$, while the errors in step 3 continue to increase for cutoff values an order of magnitude larger. Examining Figure 3, it is expected that the relatively large fraction of elements greater than 1.0 in the $(P|Q)^{-1/2}$ matrix are responsible for these observations.

Unexpectedly, however, the errors are not seen to steadily converge to the SGEMM limit for step 3 in the same way as for step 4, with errors larger than SGEMM being observed for $\delta > 2.5$. We have performed additional tests to understand why this may be happening, and our conclusion is that it results from error cancellation effects. To verify this idea, we repeated similar calculations replacing all matrix elements with their absolute values, so that any error cancellation would be essentially removed. The result was a monotonic curve much more similar to that observed for step 4, showing the same steady convergence to the SGEMM limit (not shown).

We may now consider the advantages of using MGEMM over SGEMM in terms of accuracy and speed. Comparing the subplots in Figures 4 and 5, we can see that for a rather modest performance decrease from approximately five to four times and nine to seven times for steps 3 and 4, respectively, an order of magnitude reduction in the errors can be obtained. However, it might be noted that in all cases the maximum errors are rather small in these tests, being only of order 10^{-6} in the worst case. Considering real RI-MP2 applications, we might, therefore, expect the final errors in the molecular energy to be almost negligible, using single precision only. However, in Section 5, the benchmarks show that for larger molecules, the errors propagate such that the resulting correlation energy errors are too large to be acceptable.

Finally, from Figure 2, we can estimate an upper bound on the maximum absolute error of each element for different δ-values. Since the matrix dimension is approximately 4 000, the choice $\delta = 0.1$ would give a conservative error bound of approximately $4\,000 \times 10^{-6} \times 0.1$, which is of order 10^{-4}. However, because the matrices do not have a ‘constant background’ of 0.1, this estimate is very conservative, and the observed error in Figure 5 is much less.

5. RI-MP2 Acceleration Benchmarks

In this section, our intention is to perform full RI-MP2 quantum chemistry calculations on real molecules and to benchmark the speedups and the accuracy in the resulting molecular energy that can be obtained when using the GPU. In this case, we include in the timings all steps required to compute the RI-MP2 correlation energy (after the SCF cycle has finished), while the GPU *GEMM libraries are used to accelerate the matrix multiplications in steps 3 (eq 7) and 4 (eq 6), as described in the previous sections. As a result, the observed speedups will be reduced compared to the previous benchmarks, since not all steps are accelerated.

For all these benchmarks, we used an AMD Athlon 5600+ CPU clocked at 300 GHz, combined with an NVIDIA Tesla C1060 GPU with 4 GiB of RAM. For some calculations, the GPU was limited to 256 MiB of RAM, as described below.

We emphasize that only the latest GPU cards have double-precision support to enable CUBLAS DGEMM, while older cards also have limited memory, which significantly constrains the size of even the CUBLAS SGEMM matrix multiplications. Our previous attempts to use GPUs to accelerate RI-MP2 calculations were limited to molecular systems with less than 500 basis functions[15] due to this constraint. However, using the matrix cleaver in the (MGEMM) library, we are now able to run calculations of a size limited only by the CPU specification, independent of the GPU memory.

For our test systems, we chose a set of linear alkanes (C_8H_{18}, $C_{16}H_{34}$, $C_{24}H_{50}$, $C_{32}H_{66}$, and $C_{40}H_{82}$) as well as two molecules of pharmaceutical interest, taxol ($C_{47}H_{51}NO_{14}$) and valinomycin ($C_{54}H_{90}N_6O_{18}$), and we considered the cc-pVDZ and cc-pVTZ[27] basis sets.

The matrix cleaver and MGEMM were implemented in a modified version of the Q-Chem 3.1 RI-MP2 code previously described.[15] Concerning the batching over occupied orbitals, as discussed in Section 4.2, only the step 4 matrices were batched. For all molecules, the batch size was chosen dynamically based on the matrix sizes and the available CPU memory (for taxol, this results in a batch size of seven, as used before). However, in these benchmarks the batching issue is less important, since we were limited to only 256 MiB of GPU RAM, which means that large batches would have to be cleaved by the MGEMM library in any case.

First, in Table 1, we benchmarked the reference case of either CUBLAS SGEMM or DGEMM for each test molecule using the double-ζ basis set. The table shows the speedup in computing the RI-MP2 correlation energy and the error relative to a standard CPU calculation (for SGEMM only). The speedups and SGEMM errors are seen to be greater for the larger molecules, as expected, with the largest speedups observed for valinomycin at 13.8 and 7.8 times, using SGEMM and DGEMM, respectively. However, while CUBLAS DGEMM gives essentially no loss of accuracy, the

Table 1. Speedups using CUBLAS SGEMM and DGEMM and Total Energy Errors Relative to CPU DGEMM for Various Molecules in a cc-pVDZ Basis

	speedup		
molecule	SGEMM	DGEMM	SGEMM energy error (kcal mol^{-1})
C_8H_{18}	2.1	1.9	−0.05616
$C_{16}H_{34}$	4.5	3.7	−0.12113
$C_{24}H_{50}$	6.9	5.2	−0.62661
$C_{32}H_{66}$	9.0	6.4	−0.75981
$C_{40}H_{82}$	11.1	7.2	−1.12150
taxol	11.3	7.1	−6.26276
valinomycin	13.8	7.8	−9.99340

Table 2. MGEMM Speedups and Total Energy Errors with Respect to CPU DGEMM for Various Molecules in a cc-pVDZ and a cc-pVTZ Basis

	speedup		energy error (kcal mol^{-1})	
molecule	double-ζ	triple-ζ	double-ζ	triple-ζ
C_8H_{18}	1.9	2.7	−0.01249	−0.03488
$C_{16}H_{34}$	3.8	5.6	−0.00704	−0.04209
$C_{24}H_{50}$	5.8	8.2	−0.14011	−0.33553
$C_{32}H_{66}$	7.9	9.2	−0.08111	−0.29447
$C_{40}H_{82}$	9.4	10.0	−0.13713	−0.51186
taxol	9.3	10.0	−0.50110	−1.80076
valinomycin	10.1	–	−1.16363	–

SGEMM error is approximately −10.0 kcal mol^{-1}, which is well beyond what is generally accepted as chemical accuracy.

The results from Table 1 highlight the need for MGEMM to reduce the errors when double-precision GPUs are unavailable. As an initial test of MGEMM for this purpose, we repeated the calculation of the taxol molecule in the double-ζ basis set (1 123 basis functions) for various choices of cutoff value δ. Figure 6 shows the speedup relative to CPU DGEMM as well as the absolute error in the energy.

As the cutoff increases, the MGEMM speedup increases rapidly to the asymptotic limit of 10.6 times, which is slightly less than the SGEMM limit of 11.3 times due to the MGEMM overhead. In contrast, the energy error in this range increases almost linearly toward the SGEMM limit. Recalling Figures 4 and 5, it seems that the errors are dominated by the step 3 operations, where we form the $B_{ia,Q}$ matrices, since these errors are also seen to steadily increase over the range of cutoff values considered in Figure 6. The overall speedups are also seen to have a similar shape to the step 3 speedups but are approximately twice as large. This reflects the greater speedups in step 4, noting that step 4 on the CPU is the most expensive step in the algorithm.

To achieve a target accuracy of 1.0 kcal mol^{-1}, Figure 6 shows that a cutoff value of $\delta < 2.0$ in the case of taxol

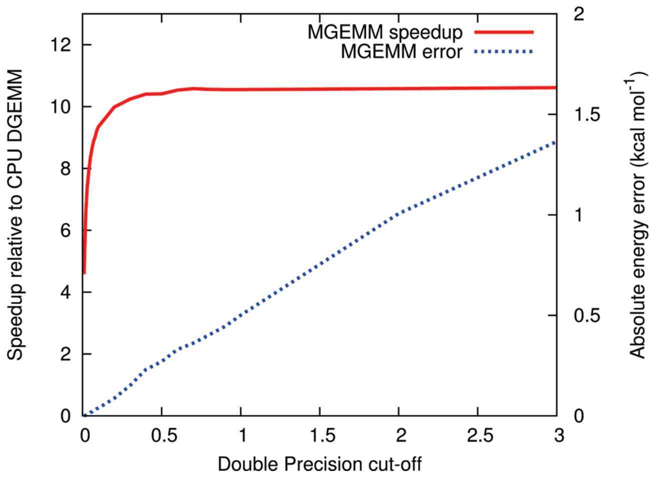

Figure 6. Taxol MGEMM calculation using a double-ζ basis set with respect to the double precision cutoff (δ). We plot the MGEMM speedup relative to CPU DGEMM, and it shows a rapid increase with δ toward an asymptotic value of 10.6 times. We also show the energy difference relative to CPU DGEMM, which is seen to increase steadily over the range of δ-values chosen but is significantly less than the previously computed SGEMM error of 6.6276 kcal mol^{-1}.

in a double-ζ basis is necessary. However, trading the accuracy and the speedup, a good choice of cutoff would be $\delta = 1.0$. This gives an error of 0.5 kcal mol^{-1}, which is an order of magnitude smaller than using SGEMM, with a speedup very close to the MGEMM limit and with only about 7% less than that of the SGEMM limit.

In Table 2, we explore the performance of MGEMM using a constant cutoff value of $\delta = 1.0$. The table shows the speedups and the total energy errors for each molecule in both the double- and triple-ζ basis sets. In this particular case, we have limited the GPU to use only 256 MiB of RAM to mimic the capability of older cards and to emphasize the use of the MGEMM cleaver. This will naturally result in a loss of speedup compared to utilizing a larger GPU memory. In the case of taxol, the reduction is approximately 20%, but obviously still much faster than a calculation using only the CPU.

Looking at Table 2, the trends are the same as in Table 1, but the MGEMM errors are seen to be approximately an order of magnitude less than the SGEMM errors (for the larger molecules). For valinomycin in the cc-pVDZ basis, the SGEMM speedup is reduced from 13.8 to 10.1 times using MGEMM, but the error in the total energy is also reduced from −10.0 to −1.2 kcal mol^{-1}, which is now very close to chemical accuracy. Moreover, while CUBLAS DGEMM clearly has the advantage (when available) of not introducing errors, if −1.2 kcal mol^{-1} is an acceptable accuracy, MGEMM may even be favored, since the DGEMM speedup is only 7.8 times compared to that of 10.1 times. Moreover, since the error increases as δ is increased, there will be a substantial error cancellation when obtaining energy differences. Thus, the apparent error in MGEMM will approach the DGEMM value.

It is unsurprising that the errors are larger when using the triple-ζ basis. The manner in which the errors grow can be anticipated using the arguments mentioned in Section 4, where we estimate an upper bound on the maximum absolute error from MGEMM by consideration of a constant background of elements no larger than the cutoff threshold and the size of the input matrices. In practice, this upper bound can be rather conservative. Moreover, if the quantity of interest is the final energy, we must also take into account how the matrices are used after the application of MGEMM (e.g., if they are multiplied by large numbers). Nevertheless, a topic of future study could be the search for a more sophisticated method for determining a safe and optimal δ-value for a given size of acceptable error in the final energy.

6. Conclusion

We have developed and implemented two new tools for the acceleration of computational chemistry codes using graphical processing units (GPUs). First, we proposed a general black-box approach for the efficient GPU acceleration of matrix−matrix multiplications, where the matrix size is too large for the whole computation to be held in the GPU's onboard memory. Second, we have shown how to improve the accuracy of matrix multiplications when using only single-precision GPU devices by proposing a heterogeneous computing model whereby both single- and double-precision operations are evaluated in a mixed fashion on the GPU and CPU, respectively.

This matrix cleaver and mixed-precision matrix multiplication algorithm have been combined into a general library named MGEMM,[28] which may be called like a standard SGEMM function call with only one extra argument, the cutoff parameter δ, which describes the partitioning of single- and double-precision work. Benchmarks of general interest have been performed to document the library's performance in terms of accuracy and speed.

Compared to a CPU DGEMM implementation, MGEMM is shown to give speedups approaching the CUBLAS SGEMM case when very few operations require double precision, corresponding to a large δ-value (which is equivalent to having a large fraction of small elements in the input matrices). However, when the fraction of large elements approaches 0.1% or greater, much less benefit is seen. Concerning accuracy, MGEMM restricts the maximum error in an element of the output matrix to an upper bound, based on the size of the matrix and the choice of δ-value. In practice, this upper bound is usually conservative. In general, the precise performance achieved with MGEMM is strongly dependent on the distribution of large and small values in the input matrices, as we have shown.

To illustrate the utility of MGEMM for quantum chemistry, we have implemented it into the Q-Chem program package to accelerate RI-MP2 calculations. We have considered both the use of modern high-end GPU cards, with up to 4 GiB of memory and with double-precision capability, as well as legacy cards, with only single-precision capability and with potentially only 256 MiB of RAM. Greater speedups but also larger absolute errors in the correlation energy were observed with the larger test molecules. In particular, for the 168-atom valinomycin molecule in a cc-pVDZ basis set, we observed speedups of 13.8, 10.1, and 7.8 times for SGEMM, MGEMM, and DGEMM, respectively. The corresponding errors in the correlation energy were -10.0, -1.2 kcal mol^{-1}, and essentially zero, respectively. The MGEMM δ-value was chosen as 1.0 for these benchmarks.

We have also suggested ways in which the size of the MGEMM error may be parametrized in terms of a conservative error bound. In addition, we have observed that the correlation energy error grows approximately linearly with the choice of δ-value, which may suggest a route to the *a priori* determination of the δ for a given target accuracy.

As we submit this paper for publication, we have become aware of the planned release of the next-generation GPU from NVIDIA, currently code-named Fermi. This card will have double-precision support with a peak performance only a factor of 2 less than that of single-precision operations. However, despite the emergence of double-precision GPU devices, it is our hope that the current work will provide a framework for thinking about other mixed-precision algorithms. Even with the more widespread availability of double-precision cards in the future, we have seen how MGEMM can run faster than CUBLAS DGEMM, if a specified level of accuracy is tolerated. Indeed, practical calculations on GPUs are very often bound by memory bandwidth to/from the device, rather than raw operation count. In these cases, the transfer and processing of only single-precision data could effectively double the performance compared to that of naive double-precision calculations.

Moreover, we are interested in the use of commodity GPUs as part of a grid-computing environment, such as the BOINC network. CUDA capable GPUs are extremely common in legacy gaming devices, but most of the client machines will not host the latest high-end hardware. We, therefore, see a significant application for MGEMM in leveraging these large numbers of legacy cards to overcome their lack of RAM and double-precision arithmetic. We are, therefore, optimistic overall about the role MGEMM can play in helping to accelerate computations using GPUs in the near future.

Acknowledgment. The authors acknowledge financial support from National Science Foundation (NSF) "Cyber-Enabled Discoveries and Innovations'' (CDI) Initiative Award PHY-0835713 as well as NVIDIA. The authors acknowledge the technical support by the High Performance Technical Computing Center at the Faculty of Arts and Sciences of Harvard University and the National Nanotechnology Infrastructure Network Computation project for invaluable support. R.O.A. wishes to thank CONACyT and Fundación Harvard en México for financial support. L.A.V. acknowledges the NSF for funding. The authors thank Prof. Tomás Arias for stimulating discussions.

References

(1) Almlöf, J.; Taylor, P. R. In *Advanced Theories and Computational Approaches to the Electronic Structure of Molecules*; Dykstra, C., Ed.; D. Reidel: Dordrecht, 1984.

(2) *CUDA Programming Guide*; NVIDIA: Santa Clara, CA; http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf. Accessed September 30, 2009.

(3) Kapasi, U. J.; Rixner, S.; Dally, W. J.; Khailany, B.; Ahn, J. H.; Mattson, P.; Owens, J. D. *Computer* **2003**, *36*, 54.

(4) Krakiwsky, S. E.; Turner, L. E.; Okoniewski, M. M. Graphics processor unit (GPU) acceleration of finite-difference time-domain (FDTD) algorithm. *ISCAS* **2004**, (5), 265.

(5) Hamada, T.; Iitaka, T. The Chamomile Scheme: An Optimized Algorithm for N-body simulations on Programmable Graphics Processing Units. 2007, arXiv:astroph/073100.arXiv.org e-Print archive. http://arxiv.org/abs/astro-ph/0703100 (accessed Dec. 7, 2009).

(6) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007**, *28*, 2618.

(7) Anderson, J. A.; Lorenz, C. D.; Travesset, A. *J. Comput. Phys.* **2008**, *227*, 5342.

(8) Anderson, A. G.; Goddard-III, W. A.; Schröder, P. *Comput. Phys. Commun.* **2007**, *177*, 298.

(9) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334.

(10) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230.

(11) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222.

(12) Ufimtsev, I. S.; Martinez, T. J. *Comput. Sci. Eng.* **2008**, *10*, 26.

(13) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004.

(14) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 2619.

(15) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049.

(16) Brown, P.; Woods, C.; McIntosh-Smith, S.; Manby, F. R. *J. Chem. Theory Comput.* **2008**, *4*, 1620.

(17) Feyereisen, M.; Fitzgerald, G.; Komornicki, A. *Chem. Phys. Lett.* **1993**, *208*, 359.

(18) Weigend, F.; Häser, M.; Patzelt, H.; Ahlrichs, R. *Chem. Phys. Lett.* **1998**, *294*, 143.

(19) Werner, H. J.; Manby, F. R. *J. Chem. Phys.* **2006**, *124*, 054114.

(20) Maschio, L.; Usvyat, D.; Manby, F. R.; Casassa, S.; Pisani, C.; Schültz, M. *Phys. Rev. B: Condens. Matter* **2007**, *76*, 075101.

(21) Frenking, G.; Antes, I.; Böhme, M.; Dapprich, S.; Ehlers, A. W.; Jonas, V.; Neuhaus, A.; Otto, M.; Stegmann, R.; Veldkamp, A.; Vyboishchikov, S. F. In *Reviews in Computational Chemistry*, Vol. 8; Lipkowitz, K. B., Boyd, D. B., Eds.; VCH: New York, 1996.

(22) Weigend, F.; Köhn, A.; Hättig, C. *J. Chem. Phys.* **2002**, *116*, 3175.

(23) Shao, Y. *Phys. Chem. Chem. Phys.* **2006**, *8*, 3172.

(24) *CUBLAS Library 1.0*; NVIDIA: Santa Clara, CA; http://developer.download.nvidia.com/compute/cuda/1_0/CUBLAS_Library_1.0.pdf. Accessed September 30, 2009.

(25) Bohannon, J. *Science* **2005**, *308*, 310.

(26) *Clean Energy Project*; Harvard University: Cambridge, MA; http://cleanenergy.harvard.edu. Accessed September 30, 2009.

(27) Dunning, T., Jr. *J. Chem. Phys.* **1989**, *90*, 1007.

(28) *SciGPU-GEMM* , *v0.8*; sciGPU.org, Harvard University: Cambridge, MA; http://scigpu.org/content/scigpu-gemm-v08-release. Accessed September 30, 2009.

CT900543Q