Accelerating Resolution-of-the-Identity Second-Order Møller−Plesset Quantum Chemistry Calculations with Graphical Processing Units†

Leslie Vogt,[‡,§] Roberto Olivares-Amaya,[‡,§] Sean Kermes,[§,#] Yihan Shao,[||] Carlos Amador-Bedolla,[§,⊥] and Alán Aspuru-Guzik*[,§]

Department of Chemistry and Chemical Biology, Harvard University, Cambridge, Massachusetts 02138, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12180, Q-Chem, Inc., 5001 Baum Blvd., Suite 690, Pittsburgh, Pennsylvania 15213, and Departamento de Química Teórica, Facultad de Química, Universidad Nacional Autónoma de México, México D. F. 04510

Received: September 24, 2007; In Final Form: November 9, 2007

The modification of a general purpose code for quantum mechanical calculations of molecular properties (Q-Chem) to use a graphical processing unit (GPU) is reported. A 4.3x speedup of the resolution-of-the-identity second-order Møller−Plesset perturbation theory (RI-MP2) execution time is observed in single point energy calculations of linear alkanes. The code modification is accomplished using the compute unified basic linear algebra subprograms (CUBLAS) library for an NVIDIA Quadro FX 5600 graphics card. Furthermore, speedups of other matrix algebra based electronic structure calculations are anticipated as a result of using a similar approach.

1. Introduction

Optimizing computational chemistry codes for central processing units (CPUs) running both in serial and in parallel has been the main focus of software developers in the scientific-computing community, especially for massively parallel high-performance computing systems. However, the increasing demand for sophisticated graphics for video games, computer-aided design (CAD), animation, and other applications is driving the development of more and more powerful graphical processing units (GPUs), which take advantage of data parallelism to render graphics at high speeds. Although video cards have been traditionally used only for graphics-intensive applications, they have also been recently leveraged toward scientific-computing problems, such as finite-difference time-domain algorithms,[1] sorting algorithms for large databases,[2] *n*-body problems,[3] and quantum Monte Carlo methods for chemical applications.[4] In these cases, programmers were required to construct GPU algorithms using a limited set of operations originally intended for computer graphics applications; however, the recent release of graphics card manufacturer NVIDIA's compute unified device architecture (CUDA) development toolkit for some of their high-end graphics cards allows developers to code algorithms in a C-like language.[5] CUDA greatly eases the transition from using CPUs to general-purpose computing on GPUs (GPGPU), as evidenced by the application of CUDA-implemented algorithms to *n*-body problems in astrophysics[6] and two-electron integrals in electronic structure problems.[7] Additionally, recent abstracts have indicated speedups using CUDA-implemented algorithms for Coulomb integral evaluations[8] and molecular dynamics.[9]

Along with CUDA, NVIDIA also released compute unified basic linear algebra subprograms (CUBLAS) as a linear algebra library for cards that support CUDA.[10] In this work, we explore using GPGPU computing for electronic structure applications by executing matrix−matrix multiplication operations using CUBLAS. In particular, we focus on reducing computational time of the resolution-of-the-identity second-order Møller−Plesset perturbation theory (RI-MP2),[11−14] as implemented in Q-Chem 3.1.[15,16] One of the widely used correlation treatments for electronic structure calculations, MP2, evaluates two-electron repulsion integrals of the form[17]

$$(\mu\nu|\lambda\sigma) = \int\int \phi_\mu(r_1)\,\phi_\nu(r_1)\,r_{12}^{-1}\phi_\lambda(r_2)\,\phi_\sigma(r_2)\,\mathrm{d}r_1\,\mathrm{d}r_2$$

where μ, ν, λ, and σ are orbital basis function (ϕ) indices. The calculation of the energy (E) is dependent on

$$E^{(2)} = \sum_{ijab} \frac{(ia|jb)^2 + \frac{1}{2}[(ia|jb) - (ib|ja)]^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}$$

$$(ia|jb) = \sum_{\mu\nu\lambda\sigma} C_{\mu i} C_{\nu a} C_{\lambda b} C_{\sigma b} (\mu\nu|\lambda\sigma)$$

where i, j (a, b) are occupied (virtual) molecular orbitals that are Fock operator eigenfunctions with eigenvalues ϵ_i, ϵ_j (ϵ_a, ϵ_b), and C is the molecular orbital coefficient matrix.

In RI-MP2, the evaluation time is reduced compared to traditional MP2 calculations.[11,12] This technique involves approximating the costly four-center two-electron integrals with the use of two-center and three-center integrals. To evaluate the integral, products of orbital basis functions are represented as a linear combination of atom-centered auxiliary basis functions P

$$\rho_{\mu\nu}(r) = \mu(r)\,\nu(r) \approx \tilde{\rho}_{\mu\nu}(r) = \sum C_{\mu\nu,P} P(r)$$

† Part of the "William A. Lester, Jr., Festschrift".

* Corresponding author. E-mail: aspuru@chemistry.harvard.edu.
‡ These authors contributed equally to this work.
§ Harvard University.
# Rensselaer Polytechnic Institute.
|| Q-Chem, Inc.
⊥ Universidad Nacional Autónoma de México.

10.1021/jp0776762 CCC: $40.75
Published on Web 01/30/2008

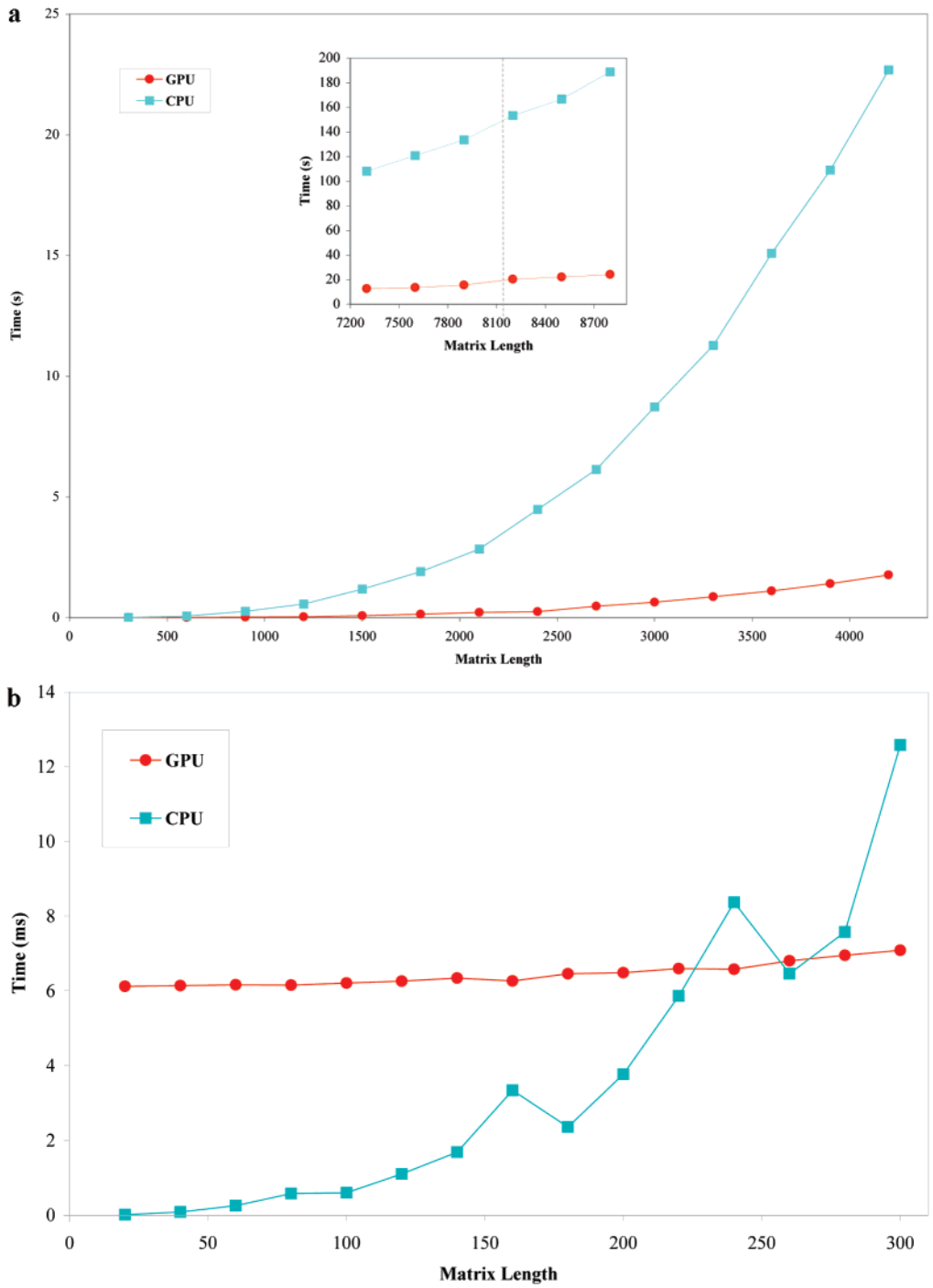

Figure 1. Average processing times for the multiplication of large matrices using a CPU and a GPU (as a coprocessor). Pairs of square matrices are multiplied on an AMD dual-core CPU and on an NVIDIA Quadro FX 5600 GPU. (a) For matrices as small as 750 elements per side, the GPU outperforms the CPU. For matrices with an area of a few million elements, a 13x speedup is obtained using the GPU. Inset: Matrices larger than 2^{13} elements on a side cannot be directly multiplied in the GPU due to memory limitations, but splitting the matrices into smaller pieces before GPU multiplication shows no appreciable difference in total time. Split-matrix timings are for the points to the right of the vertical line. (b) For small matrices, the data transfer overhead of the slow PCI bus makes the GPU slower than the CPU implementation.

When the Coulomb self-interaction of the residual density is minimized, the four-center integrals are approximated as

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

This step is an approximate insertion of the resolution-of-the-identity,

$$I = \sum_{m} (|m)(m| \approx \sum_{P,Q} |P)(P|Q)^{-1}(Q|$$

RI-MP2 is known to produce equilibrium geometries that rival density functional theory (DFT) except for transition metal compounds.[18] On the other hand, RI-MP2 is also known to capture long-range correlation effects, which are missing in many popular density functionals. So for many weakly bound systems, where DFT results might become questionable, RI-MP2 stands as essentially the least expensive alternative.[16,19]

In section 2, we present an overview of GPGPU computing and section 3 contains general results of matrix−matrix multiplication times for both CPU and GPU implementations.

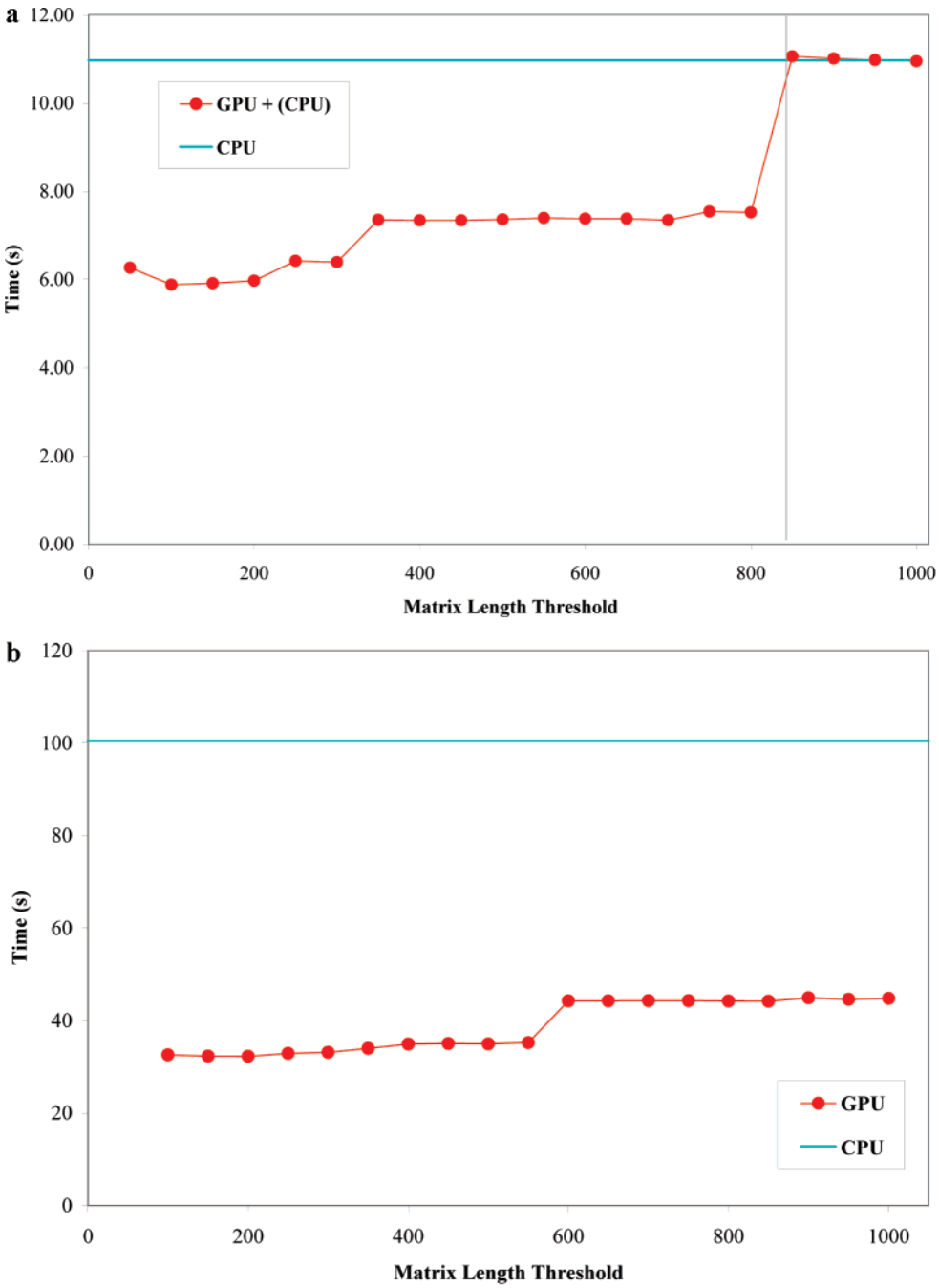

Figure 2. Total RI-MP2 calculation time for *n*-octane and *n*-tetradecane using a CPU and a GPU (as a coprocessor). If a matrix edge is smaller than the threshold, it is multiplied in the CPU; otherwise, it is multiplied in the GPU. For both cases the matrix grouping factor *S* is set to 5. (a) For *n*-octane, larger thresholds revert back to CPU timings because there are no matrices with edges larger than 850 (vertical line) when using the cc-pVDZ basis set. (b) *n*-Tetradecane calculations are 70% faster than the CPU in the 150−500 range of thresholds.

In section 4, we describe performance improvements achieved through the use of a GPU in RI-MP2 calculations of a series of alkanes, showing the effect of changing the number of electrons and the quality of the basis set used in the computation. Finally, in section 5, we conclude with our resulting RI-MP2 speedup and the potential impact that GPGPU computing can have on electronic structure calculations.

2. General-Purpose Computing on Graphical Processing Units

Graphical processors are able to outperform CPUs for certain applications because of intrinsic parallelization within the device. Multicore and parallel CPU architectures, though able to run many instructions simultaneously, require computational threads to be explicitly coded to make optimal use of the available resources. Whereas a single-core CPU can only execute a single instruction at a time (although several instructions may be in the pipeline), a GPU can execute a single instruction on many pieces of data at the same time, using a single instruction, multiple data (SIMD) paradigm. This inherent parallelization is a result of hardware architecture; graphics cards are composed of an array of multiprocessors, each of which has its own section of pipeline bandwidth. The graphics card used in this study has 16 multiprocessors, with each multiprocessor containing eight

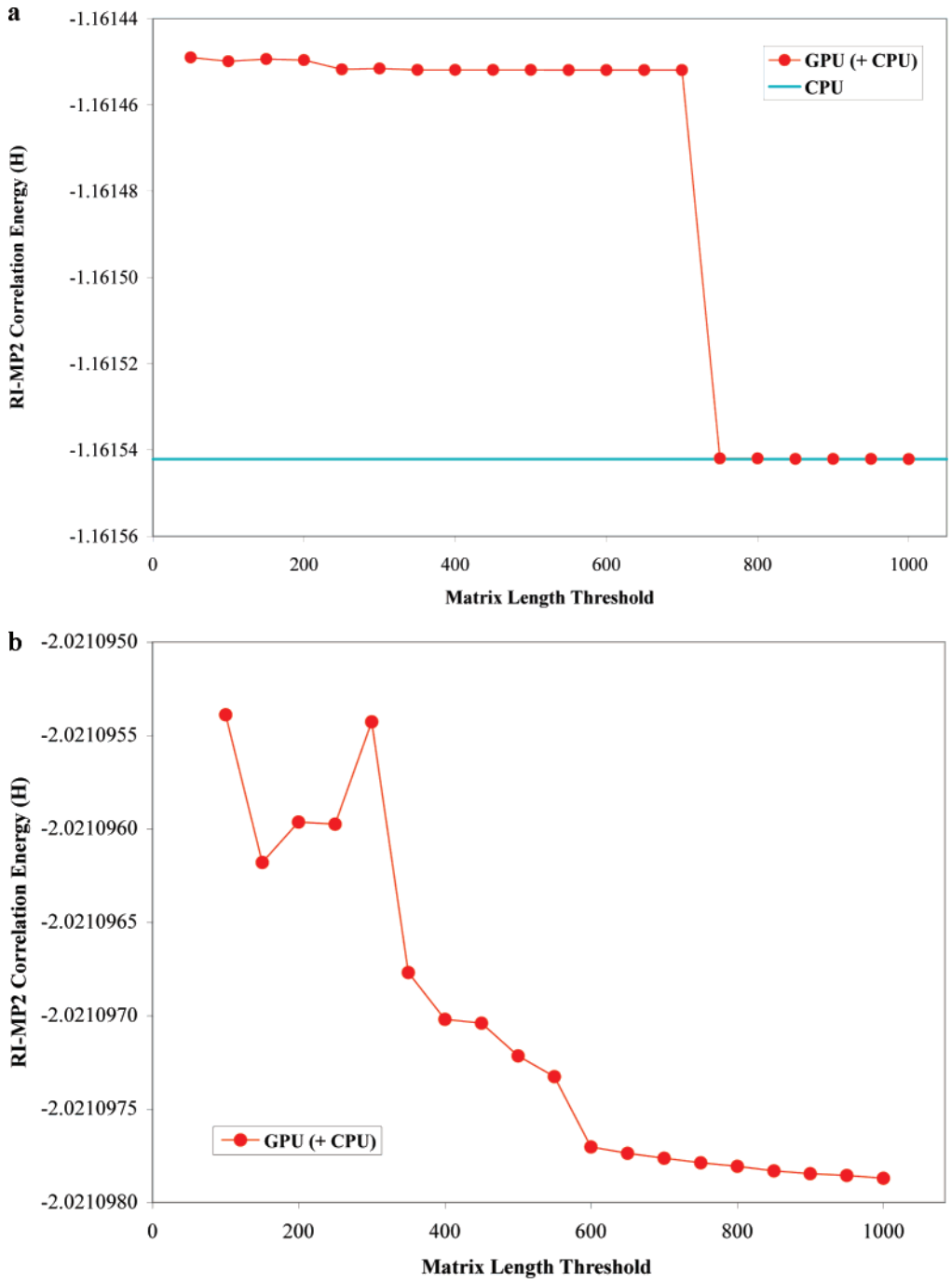

Figure 3. RI-MP2 correlation energy obtained for *n*-octane and *n*-tetradecane using the cc-pVDZ basis set on a CPU and a GPU (as a coprocessor). (a) For *n*-octane, the GPU-implemented algorithm returns an energy only 0.11 mhartree off of the value obtained using the CPU alone. (b) For *n*-tetradecane, the CPU-calculated energy is −2.02124 hartrees (not shown).

processors and able to handle up to 768 threads concurrently.[5] At any given clock cycle, each multiprocessor executes the same thread on all eight processors, although each processor operates on different data. The threads are periodically switched to minimize the chance that a thread is waiting for the appropriate data to be available. For problems that exhibit high levels of data parallelism, GPUs can provide considerable computational speedup because this hardware design allows multiple computational threads to execute quickly on a block of data that is reused. This approach is ideal for rendering graphical data, but some scientific-computing applications can also be adapted for use on these powerful video cards.

Until recently, a major hurdle in developing general-purpose applications for GPUs was the difficulty of programming for them. The only access to the device was either through graphics packages like OpenGL or by using a special assembly language made for the card. Graphics packages provide the wrong abstraction for nongraphical applications, making programs written with them difficult to understand, maintain and use. Writing assembly code directly for the device is a less than ideal solution because of limits on the number of instructions the card is able to process at a given time and the expertise required to write code for a particular GPU. However, due to the computational potential of GPUs for general computation,

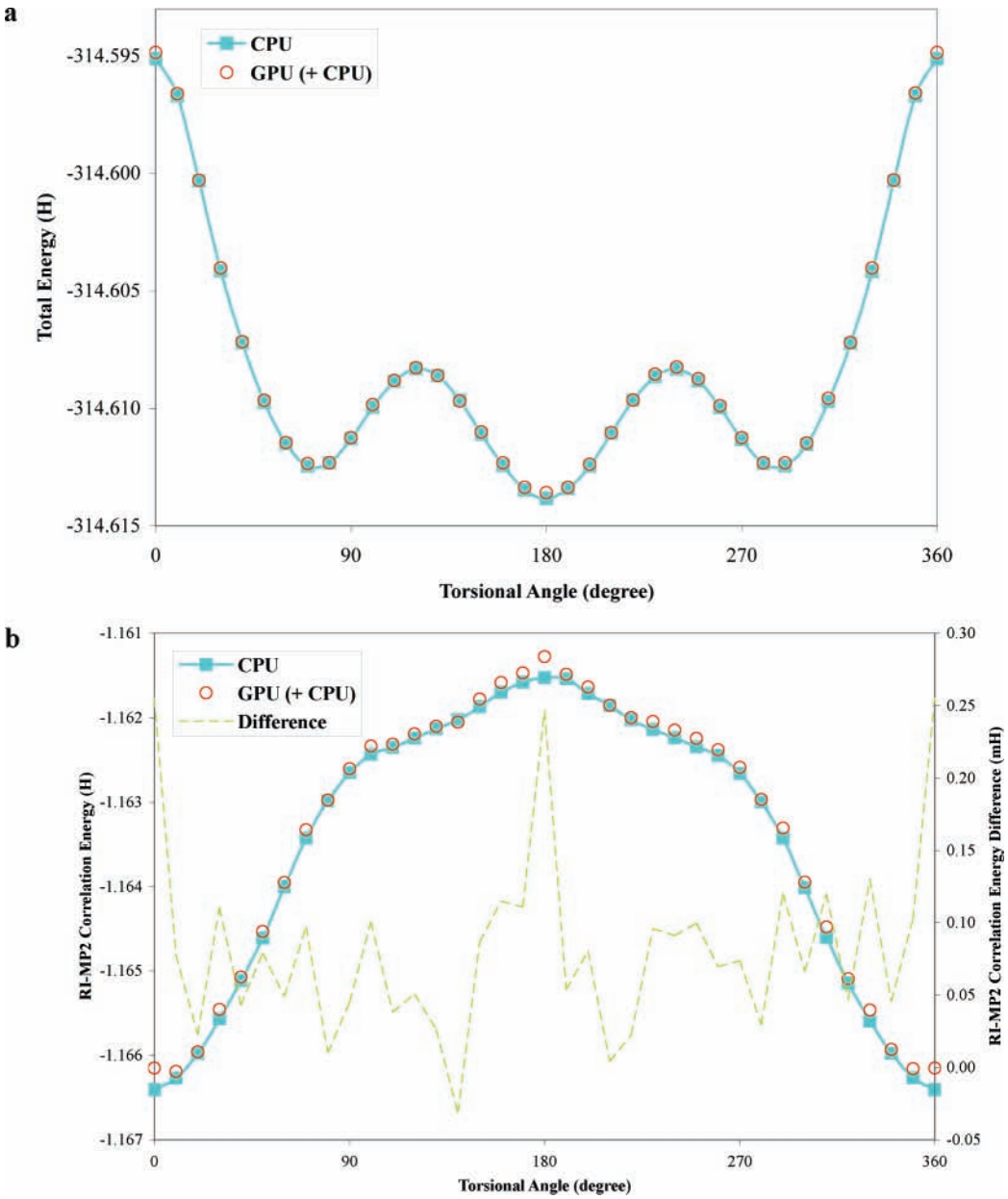

Figure 4. Total and RI-MP2 correlation energy obtained for *n*-octane conformers using the cc-pVDZ basis set on a CPU and a GPU (as a coprocessor) over a range of central bond torsional angles. (a) Total energy from single point calculations for the series of conformers. (b) RI-MP2 correlation energy for the series of conformers shows that the average random error introduced by the GPU calculations is 0.08 mhartree (rmsd: 0.1 mhartree, MAD: 0.05 mhartree).

programmers were interested in implementing linear algebra routines on GPUs, even before the release of the CUDA toolkit.[20,21] Operations such as vector products and matrix−matrix multiplication are easily parallelizable, have high levels of data reuse, and are important building blocks for other applications. With the release of the CUBLAS library, migrating code written in C and Fortran to GPUs is now considerably easier.

Despite decades of sustained progress in state-of-the-art of quantum chemistry methodology, speeding up calculations and thereby increasing the size of tractable molecules is an ongoing activity. The use of GPUs provides an important opportunity to gain further speedups when linear algebra operations are heavily used, as in RI-MP2. For this study, we focus on the effect of carrying out matrix−matrix multiplication using a GPU because this operation is one of the more time-consuming routines for CPUs to perform. The best known matrix−matrix multiplication algorithm scales as $O(n^{2.3})$ in computational time with matrix edge length.[22] The CUBLAS matrix−matrix multiplication function scales as $O(n^3)$,[10] nevertheless the scaling prefactor is found to be considerably smaller than for comparable CPU algorithms, as discussed in section 3.

3. Computational Setup and Matrix Multiplication Comparison

The hardware setup consists of a single NVIDIA Quadro FX 5600 GPU, an AMD Dual Core Opteron 170 processor, a LanParty NForce 4 motherboard, and two gigabytes of RAM. The operating system used is Ubuntu 7.04 with a Linux 2.6 kernel and the Intel Fortran Compiler v10.0. To characterize CUBLAS function performance, we obtained benchmarks of the speed of matrix−matrix multiplication using both the host

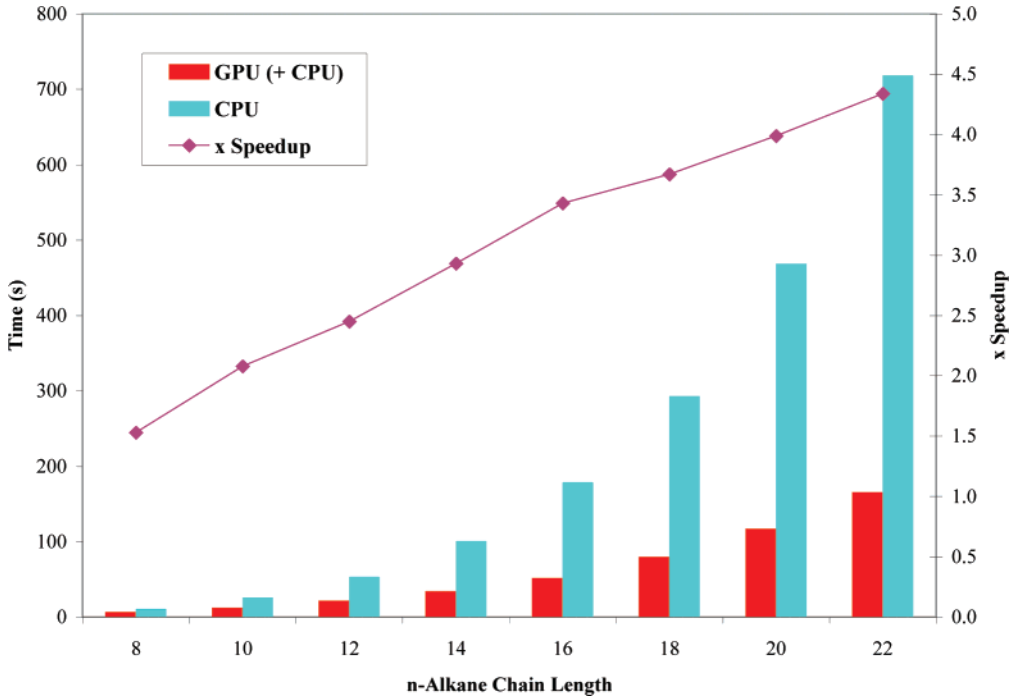

Figure 5. Total processing times for the calculation of RI-MP2 single point energies for a series of linear alkanes using the cc-pVDZ basis set on a CPU and a GPU (as a coprocessor). The series of linear alkanes with even numbers of carbon atoms from octane (C_8H_{18}) to doeicosane ($C_{22}H_{46}$) is investigated. Speedups of 1.5x to 4.3x (average 2.7x) are achieved throughout the series with an average RI-MP2 correlation energy error of 0.3 mhartree.

CPU (regular BLAS function) and the GPU (CUBLAS function). Figure 1a shows the average ($N = 20$) computational times to multiply a pair of square matrices with 300−4000 elements on a side. To multiply a pair of 300 by 300 matrices, both the CPU and the GPU took only a hundredth of a second. As the size of the matrices increases, the benefit of the GPU becomes apparent. Large matrices can be multiplied about thirteen times faster on the GPU than on the host CPU, a significant gain for a moderate programming effort. We separately analyzed the times of preprocessing, actual matrix multiplication and postprocessing for the CPU and the GPU. The preprocessing and postprocessing times are due mostly to data transfer between the motherboard and the GPU memory. The scaling with matrix size is the same for a given stage in both units, but the prefactor is reduced 20-fold for the GPU in the case of the multiplication step. This reduction in evaluation time overwhelms the modest increase in pre- and postprocessing time for the GPU and the advantage of using the GPU for this function increases for larger matrices.

A challenge to effectively using GPUs for linear algebra occurs when matrices become very large. The GPU on-board memory is a finite resource, which can restrict the number of matrix elements that can be passed to the card. Although the Quadro FX 5600 has 1.5 gigabytes of on-board RAM, the memory required to multiply very large matrices can exceed this limit, causing the device to crash. When we attempted to multiply matrices larger than 8192 elements on a side, the memory on the device was exhausted and the CUBLAS library process stopped. To get around this memory limitation, the large matrices can be split into pieces before being sent to the GPU, multiplied there, and then put back together using the CPU. The inset of Figure 1a shows the result of using this method on a hardware system consisting of a single NVIDIA Quadro FX 5600 GPU, an AMD Dual Core Athlon X2 processor, an Asus M2N32-SLI Deluxe motherboard, and 4 gigabytes of RAM. Data points for matrices smaller than 8192 elements on a side are multiplied using the standard library call, and points for larger matrices use the method described. Though a little performance is lost transferring data in the bus, multiplying these very large matrices in multiple passes on the GPU is still significantly faster than using just the CPU.

In electronic structure applications, however, many of the matrices that need to be multiplied are much smaller than a thousand elements on a side. Figure 1b shows average ($N = 20$) computational times for multiplying a pair of square matrices that have only 20−300 elements on a side. The major bottleneck for using the GPU is revealed by this figure—the PCI bus latency. Data communication between the GPU and the CPU is conducted via a NVIDIA CK804 PCI x16 Bridge (Clock 66 MHz, Width 32 bits), which is considerably slower than typical memory access for a CPU. The time it takes to transfer data over the bus is long enough to make using a GPU for matrix multiplication on matrices smaller than two hundred elements on a side inefficient. Our approach, therefore, is to set a lower bound on when to use the graphics card processors, similar to Yasuda's use of a threshold parameter to control integral evaluations.[7] If a resultant matrix has an area smaller than the square of a cutoff threshold, it is multiplied using the CPU and all matrices larger than the cutoff are multiplied using the GPU. Because electronic structure calculations use rectangular matrices, the optimal cutoff threshold is not obvious, as discussed in the next section.

4. Speedups of RI-MP2 Calculations on a Series of Alkanes

Within Q-Chem 3.1, the RI-MP2 correlation energy is evaluated in five steps. The steps are listed below with the following abbreviations: atomic basis functions, auxiliary basis functions, occupied and virtual orbitals have the same notation

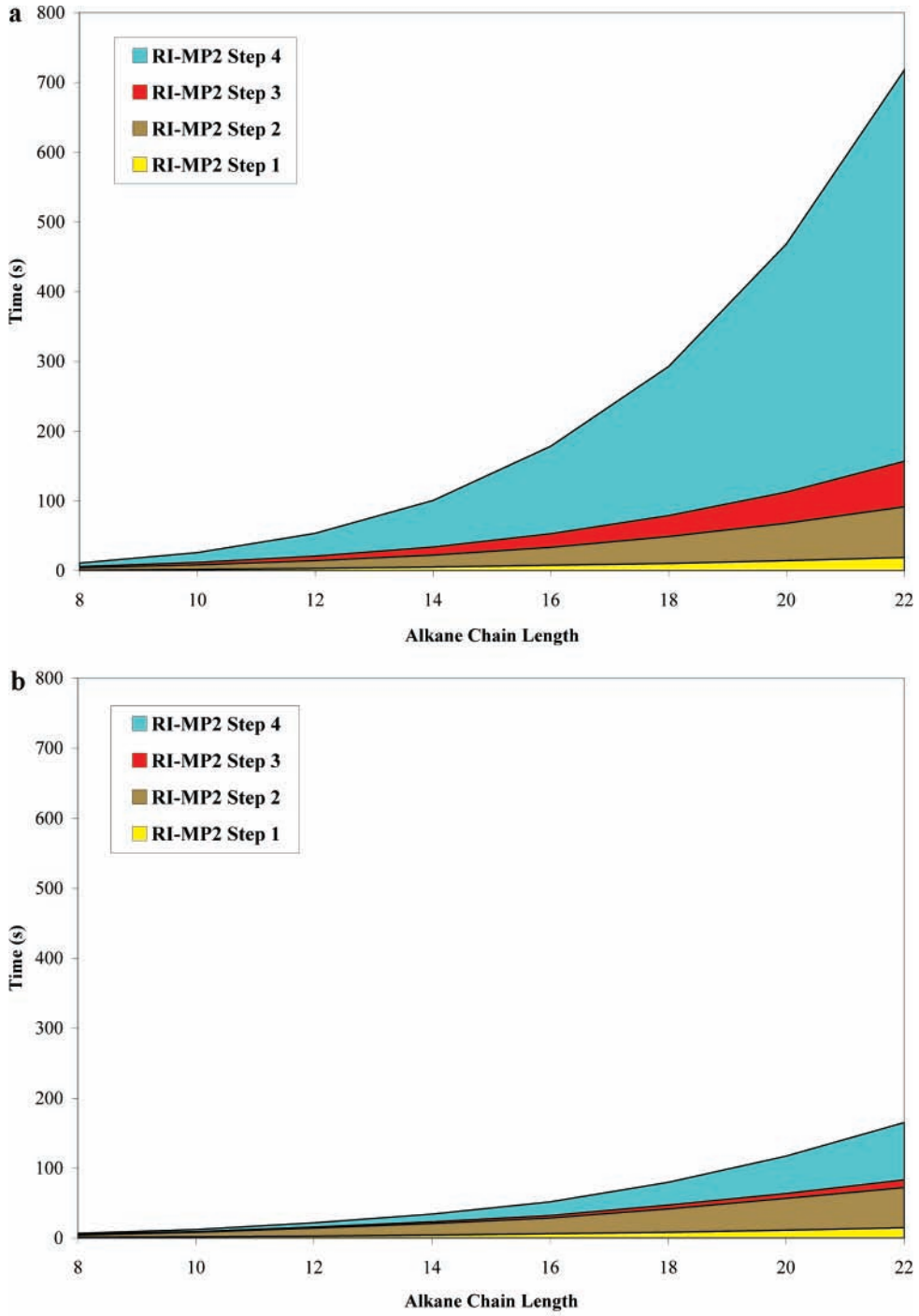

Figure 6. Computational time for the each step of the RI-MP2 calculation is plotted for a series of linear alkanes. The time required for step 5 is less than 1 s, which is smaller than the visible scale of the graphs. (a) The breakdown of processing times for each step on the CPU shows that the matrix−matrix multiplication is 78% of the total calculation time. (b) Using the GPU as a coprocessor, the cost of step 4 is reduced to only 50% of the total computational time.

as in section 1, N (M) is the number of atomic (auxiliary) basis functions, O (V) is the number of occupied (virtual) orbitals, and α, β, γ, and η are prefactors for the estimated computational cost.

Step 1: Evaluate $(P|Q)$, which are two-electron repulsion integrals between two auxiliary basis functions and form the square root of its inverse matrix, $(P|Q)^{-1/2}$. The estimated computational cost of this step is $\alpha M^2 + \beta M^3$.

Step 2: Evaluate $(\mu\nu|P)$, which are two-electron repulsion integrals between a pair of normal atomic basis functions; and an auxiliary basis function then transform the three-center integrals into $(ia|P)$. The estimated computational cost of this step is $\gamma N^2M + 2N^2MO + 2NMOV$.

Step 3: Form $B_{ia,Q}$ via

$$B_{ia,Q} = \sum_{P}^{AUX} (ia|P)(\mathrm{P}|Q)^{-1/2}$$

The estimated computational cost of this step is $2M^2OV$.

Step 4: Form $(\widetilde{ia|jb})$ via

$$(\widetilde{ia|jb}) \approx \sum_{Q}^{AUX} B_{ia,Q} B_{jb,Q}$$

The estimated computational cost of this step is MO^2V^2.

Step 5: Evaluate RI-MP2 energy using

$$E^{(2)}_{\mathrm{RI-MP2}} = \sum_{ijab} \frac{(\widetilde{ia|jb})^2 + \frac{1}{2}[(\widetilde{ia|jb}) - (\widetilde{ib|ja})]^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}$$

The estimated computational cost of this step is ηO^2V^2.

We observe that steps 1 and 5 contribute little to the total CPU time, whereas step 2 and 3, which both scale as $O(n^4)$ with the system size, have comparable costs. Step 4, which scales as $O(n^5)$, dominates the RI-MP2 energy evaluation, and becomes our primary concern in this study. In the original code for step 4 within Q-Chem 3.1, there is a loop over (i, j) pairs, and within the loop, a matrix of size MV is multiplied to the transpose of a matrix of the same size. In this work, we introduce a new parameter, S, which groups S occupied orbitals together during the computation and, within each loop, results in the total matrix size of $M(SV)$. On the basis of speedups obtained for *n*-octane (which uses the smallest matrices of the series), S is set to five for all *n*-alkane calculations. This simple code modification stacks matrices that would otherwise be smaller than the cutoff threshold together, allowing the GPU to be used more effectively.

Figure 2 shows the speedup in RI-MP2 calculations obtained simply by modifying the code as described above and using the GPU as a coprocessor to execute the matrix−matrix multiplication in step 4. Results are shown for *n*-octane (Figure 2a) and for *n*-tetradecane (Figure 2b), both treated with a cc-pVDZ basis set.[23] Total computational time is plotted against the threshold used to send matrices to the GPU with CPU time for each calculation shown as a reference. As seen in Figure 2a, the value of the threshold must be set smaller than the maximum matrix size to ensure that matrices are sent to the GPU (around 850 for *n*-octane with the cc-pVDZ basis and $S = 5$). For larger systems, such as *n*-tetradecane (Figure 2b), the maximum matrix size is greater than 1000, so a speedup is obtained for all threshold values. For the *n*-alkane series comparison, a threshold of 350 is used. However, it should be noted that the value of S and the cutoff threshold can be optimized to minimize the impact of bus latency discussed in section 3. Although systems as large as $C_{22}H_{46}$ with the cc-pVDZ basis set did not reach the limit of the graphics card memory, calculations as small as $C_{14}H_{34}$ with the cc-pVTZ basis set[23] cause the device to crash. Code modifications are currently underway to incorporate the method described in section 3 into Q-Chem 3.1 to treat larger systems with more accurate basis sets.

A price to pay for the speedup achieved by using the GPU is some loss of precision (see Figure 3). Current GPUs only support 32-bit single precision floating point numbers instead of the 64-bit double precision numbers used by modern CPUs; however, this is most likely a temporary setback because manufacturers have promised double precision support in future generations of GPUs. The precision degradation due to single precision is not of great concern with our RI-MP2 calculations as long as an appropriate cutoff threshold is used. In general, the energies obtained using single precision on the graphics card were only slightly different from the ones found using double precision on the CPU. As seen in Figure 3, the difference of the RI-MP2 correlation energy using the GPU from that found using double precision using the CPU is on the order of 10^{-4} hartree for both *n*-octane and *n*-decane. Figure 4 shows the total and RI-MP2 correlation energy for a series of conformers of *n*-octane as the torsional angle of the central bond is rotated. The average error introduced in the RI-MP2 correlation energy is only 0.08 mhartree (Figure 4b), preserving the trend of the CPU calculation.

The general picture of the speedups obtained by combining the use of the GPU and the CPU can be seen in Figure 5, where we report calculations of single point energies with the RI-MP2 method for a series of linear alkanes using the cc-pVDZ basis set. Energies for alkanes with an even number of carbon atoms from octane (C_8H_{18}) to doeicosane ($C_{22}H_{46}$) are calculated. Speedups of 1.5x (35%) to 4.3x (77%), with an average of 2.7x (63%), are achieved throughout the series. This is a significant increase of efficiency in molecular calculations with no considerable expense of precision. Even with GPUs that only work with single precision arithmetic, the average error in RI-MP2 correlation energy is 0.3 mhartree (rmsd: 0.5 mhartree, MAD: 0.3 mhartree). This is only 6×10^{-5}% of the average total energy.

To show the effect of using the GPU for matrix−matrix multiplication in RI-MP2 calculations, Figure 6 plots the computational time required by each RI-MP2 step for the series of alkanes. For $C_{22}H_{46}$, a system with 178 electrons, the time required to evaluate step 4 of the RI-MP2 calculation on the CPU is 78% of the total (Figure 6a). Using the GPU (Figure 6b) reduces the time spent on this step to 50% of the total RI-MP2 time. The next largest contribution to the calculation becomes step 2, which increases from 10% to 35% of the RI-MP2 time when the GPU is used for doeicosane. This step involves the evaluation of three-center integrals and subsequent two-index transformations. For the evaluation of three-center integrals, in the future we can potentially adapt the approach developed by Yasuda for the evaluation of two-center integrals to approximate Coulomb integrals. We expect that by combining the two approaches, a more significant speedup can be obtained for RI-MP2 treatment of electron correlation with only a moderate programing effort.

5. Conclusions

In this article, we demonstrated that simply rerouting one linear algebra routine in the evaluation of RI-MP2 correlation from a CPU to a GPU achieved a speedup of 4.3x for the calculation of the single point energy of doeicosane. To the best of our knowledge, this is the first implementation of a CUBLAS library function in electronic structure codes. The C-like language of CUDA allows easy migration of code segments to implementations using a GPU. The resulting price/performance is very competitive; the parallelization offered by graphical processors will allow scientific calculations that are usually run on clusters and special-purpose supercomputers to be evaluated at a fraction of the cost. Efforts to reduce computational time even more by implementing the two-electron repulsion integrals, as well as other linear-algebra operations in the code (matrix-vector multiplication and diagonalization routines), are underway. The recent availability of the CUBLAS library is an encouraging development for electronic structure developers. This level of abstraction allowed us to encapsulate the linear algebra calls of Q-Chem in such a way that all matrix−matrix multiplications are carried out with CUBLAS with minimal code

modification. This encapsulation strategy allows for faster adoption of other novel technologies such as other types of linear algebra coprocessing units as they become available. The challenge left for the electronic structure community is to restructure and adapt the current algorithms to environments where the linear algebra operations can be carried out expeditiously aided by GPUs or other similar devices such as field-programmable gate arrays (FGPAs).

Acknowledgment. S.K. thanks the National Science Foundation for financial support under REU Site: DMR-0353937. A.A.-G., L.V., and R.O.-A. acknowledge support from the Faculty of Arts and Sciences (FAS) of Harvard University and the Henry Beck Fund at the Harvard Center for the Environment. R.O.-A. is thankful for the support of Fundación Mexico en Harvard A.C. and CONACyT. We also thank the Life Sciences Division of the FAS for system administration support, Fan Wu, Jay Best, Lalit Jain, and Sangwoo Shim for helpful discussions, and Kurt Keville and NVIDIA Corporation for hardware. C.A.-B. thanks DGAPA-UNAM for financial support under PAPIIT IN206507.

References and Notes

(1) Krakiwsky, S. E.; Turner, L. E.; Okoniewsky, M. M. *Proc. 2004 Int. Symp. Circuits Systems* **2004**, *5*, V265.

(2) Govindaraju, N.; Gray, J.; Kumar, R.; Manocha, D. *Proc. 2006 ACM SIGMOD Int. Conf. Management of Data*, June 27−29, 2006, Chicago, IL, U.S.A. **2006**.

(3) Hamada, T.; Iitaka, T. *ArXiv Astrophysics e-prints* 2007, astro-ph/073100.

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

(5) *NVIDIA CUDA Programming Guide 1.0*, www.nvidia.com.

(6) Belleman, R. G.; Bédorf, J.; Portegies Zwart, S. F. *ArXiv Astrophysics e-prints* **2007**, astro-ph/0707.0438v2.

(7) Yasuda. K. *J. Comput. Chem.* **2007**, *29*, 334.

(8) Martinez, T. J.; Patel, S. **2007**, http://cms.mcc.uiuc.edu/wiki/display/mccfiles/Findings+2007.

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

(10) *NVIDIA CUBLAS Library 0.8*, www.nvidia.com.

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

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

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

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

(15) Shao, Y.; *et al. Phys. Chem. Chem. Phys.* **2006**, *8*, 3172.

(16) Distasio, R. A.; Steele, R. P.; Rhee, Y. M.; Shao, Y.; Head-Gordon, M. J. *J. Comput. Chem.* **2007**, *28*, 839.

(17) Head-Gordon, M.; Pople, J. A.; Frisch, M. J. *Chem. Phys. Lett.* **1988**, *153*, 503.

(18) Frenking, G.; Antes, I.; Böhme, M.; Dapprich, S.; Neuhaus, A.; Otto, M.; Stegmann, R.; Veldkamp, A.; Vyboishchikov, S. F. *Reviews in Computational Chemistry*; VCH: New York, 1996; Vol. 8.

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

(20) Krüger, J.; Westermann, R. In *ACM SIGGRAPH 2005 Courses, July 31-August 04, 2005, Los Angeles, CA, U.S.A.*; Fujii, J., Ed.; SIGGRAPH 2005; ACM Press: New York, 2005; p 234, http://doi.acm.org/10.1145/1198555.1198795.

(21) Fatahalian, K.; Sugerman, J.; Hanrahan, P. *Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. on Graphics Hardware, August 29−30, 2004, Grenoble, France*; ACM Press: New York, 2004; p 133, http://doi.acm.org/10.1145/1058129.1058148.

(22) Coppersmith, D.; Winograd, S. *J. Symbol. Comput.* **1990**, 251.

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