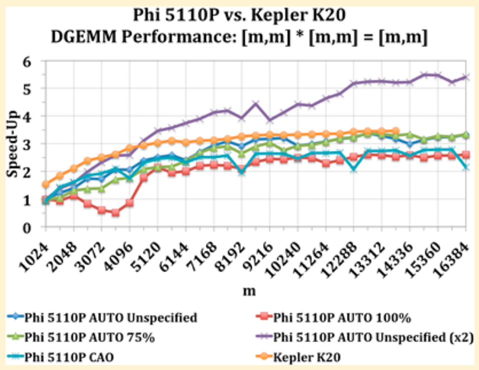
Quantum Chemical Calculations Using Accelerators: Migrating Matrix Operations to the NVIDIA Kepler GPU and the Intel Xeon Phi

Sarom S. Leang,[‡] Alistair P. Rendell,[†] and Mark S. Gordon*[,‡]

[†]Research School of Computer Science, Australian National University, Acton ACT 0200, Australia
[‡]Department of Chemistry and Ames Laboratory, Iowa State University, Ames, Iowa 50011-3111, United States

ABSTRACT: Increasingly, modern computer systems comprise a multicore general-purpose processor augmented with a number of special purpose devices or accelerators connected via an external interface such as a PCI bus. The NVIDIA Kepler Graphical Processing Unit (GPU) and the Intel Phi are two examples of such accelerators. Accelerators offer peak performances that can be well above those of the host processor. How to exploit this heterogeneous environment for legacy application codes is not, however, straightforward. This paper considers how matrix operations in typical quantum chemical calculations can be migrated to the GPU and Phi systems. Double precision general matrix multiply operations are endemic in electronic structure calculations, especially methods that include electron correlation, such as density functional theory, second order perturbation theory, and coupled cluster theory. The use of approaches that automatically determine whether to use the host or an accelerator, based on problem size, is explored, with computations that are occurring on the accelerator and/or the host. For data-transfers over PCI-e, the GPU provides the best overall performance for data sizes up to 4096 MB with consistent upload and download rates between 5–5.6 GB/s and 5.4–6.3 GB/s, respectively. The GPU outperforms the Phi for both square and nonsquare matrix multiplications.



1. INTRODUCTION

The use of accelerators to enhance scientific computing goes back at least to the early 1980s, when the FP730 floating point accelerator was introduced for the VAX 11/730 computers that had only recently become popular in academic departments. In the last several years, the most popular accelerator has been the NVIDIA graphical processing unit (GPU). This accelerator has been employed successfully in molecular dynamics simulations[1] and in electronic structure theory methods, such as Hartree–Fock (HF),[2] density functional theory (DFT),[3] second order perturbation theory (MP2),[4] and coupled cluster (CC) theory.[5] The use of the NVIDIA GPU accelerator has been largely successful, but a large investment in novel programming is required to gain a significant improvement over purely CPU computations. Several popular electronic structure codes, such as TeraChem,[6] GAMESS,[7] NWChem,[8] and Q-Chem,[9] to name a few, offer GPU-enabled codes. While hybrid GPU/CPU speedups of up to 2 orders of magnitude relative to purely CPU computations have been reported,[10] these often fail to compare GPU acceleration relative to the very best and most efficient CPU code running on all cores of the host CPU.

Recently, the Intel Phi accelerator has gained some interest, in part because the Phi is purported to require little or no new programming effort; however, there have been few reports in which the performance of the Phi accelerator is compared with native CPU performance or with GPU speedups.

The focus of this work is to compare the performance of the Intel Phi 5110P with the NVIDIA K20 GPU for matrix operations using compiler directives and vendor supplied accelerated math libraries. In particular, double precision general matrix multiply (DGEMM) operations are used for benchmark comparisons. These are a core functionality requirement for electronic structure theory computations, especially for methods that incorporate electron correlation corrections at some level. Also, as data must be communicated from the host to the accelerator in order to perform the DGEMM, the data-transfer performance of the Peripheral Component Interconnect (PCI)-e is evaluated.

2. COMPUTATIONAL APPROACH

The need for DGEMMs is ubiquitous in quantum chemistry. All methods that include electron correlation, as well as analytic second derivatives, require a four index integral transformation from the atomic orbital (AO) to the molecular orbital (MO) basis. This is best performed as a set of four DGEMM operations. For CC and MP2, DGEMMs also occur in the contraction of integrals and amplitudes.[11]

Typically, migrating matrix multiplications onto accelerators involves the transfer of matrix data from the host to the accelerator, computing the matrix product on the accelerator, and transferring the matrix product from the accelerator back to the host. The benefit of migrating matrix multiplications onto accelerators is evaluated by comparing the total time needed

Received: December 7, 2013
Published: February 17, 2014

with and without the accelerator. This provides the speedup relative to the host.

The data-transfer performance of the Phi 5110P and the NVIDIA K20 over PCI-e was evaluated by measuring the time taken to transfer matrices with data sizes ranging from 2 to 4098 MB (in increments of powers of two) from the host to the accelerator (upload) and from the accelerator to the host (download). For each data size, an untimed "warm-up" transfer was first completed followed by three consecutive transfers that were timed. The average transfer time was used to calculate the data-transfer rate.

To evaluate the Phi 5110 and Kepler K20 performance, the matrix product $[m, k] \times [k, n] = [m, n]$ was calculated. The dimensions of these matrices are usually related to the size of the AO basis or the number of electrons/number of occupied MOs. Square matrices are associated with full AO to MO transformations. For correlation methods with a large basis set the occupied orbitals typically represent 5% or less of the total number of MOs.

Square matrix multiplications ($m = k = n$) were evaluated for values of m ranging from 1024 to 16384. Three sizes of nonsquare $[m, k]$ matrices were used: 32, 768, and 1458 MB. For each size, the row to column ratio, m:k, was varied from 1:1 (square matrix) to 1:110 reflecting the ratio of occupied to total MOs, in such a manner that the calculation fits within the memory limits of the K20 GPU.

2.1. Hardware. The host consists of two 2.0 GHz 8-core Intel Xeon E5-2650 processors, configured with 128 GB of memory and with a theoretical peak double-precision performance of 256 GFLOPS.[12]

The K20 GPU features a streaming multiprocessor (SMX) and is configured with 5 GB of GDDR5 memory with a theoretical peak memory bandwidth of 208 GB/s.[13] Each of the 13 SMX operates at 705 MHz and contains 64 double-precision units, each of which can execute a fused multiply-add operation. The K20 possesses a theoretical peak double-precision performance of 1173 GFLOPS (4.58× theoretical peak double-precision performance speed-up relative to the host). Host memory can be moved or transferred to disk by the operating system. Pinned memory helps improve memory access between the host and device by preventing the host memory from being swapped.

The Phi 5110P coprocessor features 60 64-bit x86-based Intel cores operating at 1.053 GHz.[14] Each core supports four hardware threads and contains a 512-bit SIMD instruction set. Each core is capable of executing 16 double-precision fused multiply-add operations per clock cycle. The Phi has a theoretical peak double-precision performance of 1010 GFLOPS (3.94× theoretical peak double-precision performance speed-up relative to the host), is configured with 8 GB of GDDR5 memory, and has a theoretical peak memory bandwidth of 320 GB/s. Using the MKL math library, one can employ either automatic or compiler-assisted offloading. Automatic offloading (AUTO) is controlled using an environmental variable and no code modifications are needed. The minimum dimensional requirement for AUTO matrix multiplication is m, $n > 2048$ and $k > 256$.[15] With AUTO, the computational work division between the host and Phi coprocessor is determined at runtime. AUTO does not allow the user to control the movement of data between the host and coprocessor.

In compiler-assisted offloading (CAO), pragmas are inserted into the host code to manage data movement and computation offload onto the coprocessor. In the current study, pragmas are used to isolate communication between the host and the coprocessor in order to evaluate data-transfer performance over PCI-e. Pragmas are also used to isolate and control the computational offloading between the host and the coprocessor so that all of the computations are offloaded onto the coprocessor. The latter allows for a direct comparison of the DGEMM performance between the Phi coprocessor and the K20 GPU. In the current work, implementing the CAO model for offloading DGEMM routines involved the same amount of effort and code modifications necessary to utilize the NVIDIA cuBLAS library on the GPU.

The runtime performance of the Phi can also be fined tuned through the use of environmental variables to control, for example, the binding of threads to physical processing cores (thread affinity).[16] To distinguish environmental variables that may be shared by the host and the coprocessor a prefix can be assigned (e.g., MIC_ENV_PREFIX=MIC). By default, four hardware threads are assigned to each core on the Phi with one of the cores being excluded when using the offload runtime. An offloaded computation on the Phi can make use of up to 236 threads. The MIC_KMP_AFFINITY environmental variable controls the binding of threads to cores. The default affinity setting, *scatter*, uses a round-robin assignment of threads among available cores. The *compact* affinity setting assigns threads to consecutive cores while the *balance* affinity setting divides threads up evenly into the available cores but keeping consecutive threads locally close. At 236 threads, all 59 available cores are saturated with 4 threads and the *compact* and *balance* affinity settings offer the same thread binding.

2.2. Software. The host version of the code was compiled with the Intel Fortran compiler and made use of the DGEMM routine from the threaded version of Intel MKL. In addition, *-O3* level optimizations and *-align array64byte* array alignment flags were used. All runs on the host employed 16 OMP threads with the KMP_AFFINITY environment variable unset.

The K20 GPU version of the performance code was built using the Portland Group Inc. (PGI) CUDA Fortran compiler version 13.5 with *-O3* level optimizations. In addition, the DGEMM routine from the NVIDIA cuBLAS math library was used. To improve data transfer performance between the host and the GPU, pinned host memory was used.

The Phi version of the code used the Intel Many-core Platform Software Stack (MPSS) 2.1.6720-13 and was built with the same compiler, optimization, and array alignment flags used on the host. For DGEMMs on the Phi, the coprocessor-optimized MKL DGEMM routine was used. For all Phi runs, the MIC_KMP_AFFINITY environmental variable for the coprocessor was set to *compact* as this was found to offer better overall performance relative to the default affinity setting of *scatter*. To improve data transfer performance on the Phi, the environmental variable MIC_USE_2MB_BUFFERS=2 M was set to turn on the use of 2 MB page buffers.

3. NUMERICAL RESULTS

3.1. Data-Transfer Performance over PCI-e. Figure 1 shows the data-transfer performance over PCI-e for the Phi 5110P and the K20 GPU. Overall, the K20 offers the better data-transfer performance over PCI-e for both upload and download. The Phi exhibits a data-transfer rate of 3−6.5 GB/s over the range of data sizes transferred. The K20 has a more consistent data-transfer rate of 5−6.3 GB/s over the same range. The K20 download rate is about 0.5 GB/s better than

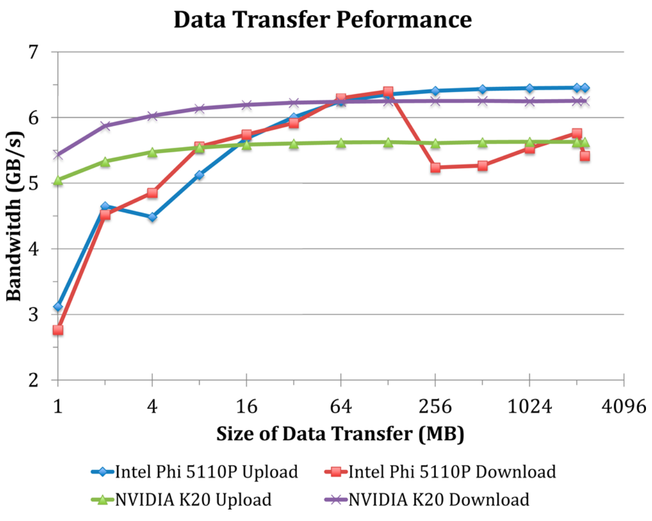


Figure 1. Data transfer performance on the Intel Xeon Phi 5110P coprocessor and NVIDIA Kepler K20 GPU for upload (host-to-device) and download (device-to-host).

the upload rate. The Phi exhibits similar data-transfer rates for both upload and download for data sizes up to 128 MB. Above 128 MB, the Phi download rate decreases by up to 1 GB/s. For 8−4096 MB data sizes, the Phi produces a data-transfer performance that is similar to that of the K20. For data sizes below 8 MB, the data-transfer performance for the Phi is poor; this poor performance might be attributed to the lack of pinned host memory support in the CAO model. (In CUDA Fortran, the use of pinned memory only requires adding the "pinned" variable attribute to the array declaration.)

3.2. DGEMM Performance. Figure 2 demonstrates the performance of the Phi and K20 GPU for *square matrix* multiplication relative to executing on the host. For the Phi coprocessor, four different AUTO scenarios are shown for the computational work division:

- 25% on the host and 75% on one coprocessor (AUTO 75%)
- 0% on the host and 100% on one coprocessor (AUTO 100%)
- determined at runtime using one coprocessor (AUTO Unspecified)
- determined at runtime using two coprocessors (AUTO Unspecified (×2)).

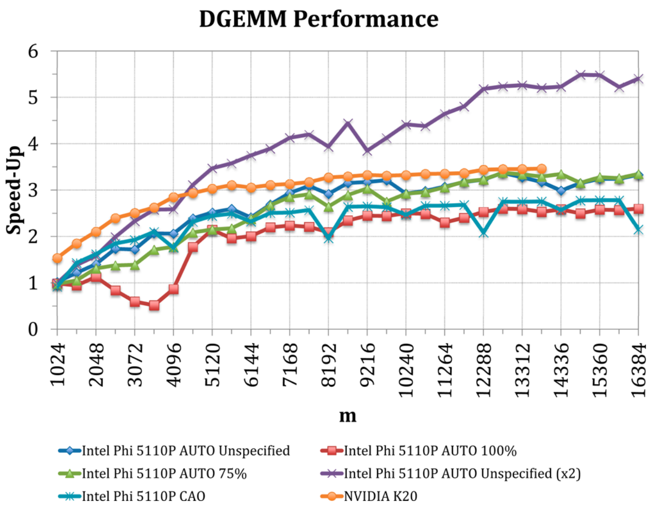


Figure 2. DGEMM performance on the Intel Xeon Phi 5110P coprocessor and NVIDIA Kepler K20 GPU for $[m, m] \times [m, m] = [m, m]$ using various $[m, m]$ matrix sizes.

For the CAO performance evaluation depicted in Figure 2, all of the computational work was executed on a single coprocessor. For comparison with the latter, the K20 performance evaluation executed all computational work on a single GPU.

Relative to running on the host, the GPU offers the best overall performance for migrating square matrix multiplication to a single accelerator. With a single accelerator, the Phi AUTO Unspecified model offers a peak speedup of 2.6−3.4× relative to the host for DGEMM square matrix operations, depending on the work division. The fact that similar performances are exhibited by AUTO 75% suggest that a 1:3 work division ratio between the host and Phi was used at runtime in the Unspecified case for most of the square matrices tested. This is in line with analysis of the offload report for AUTO Unspecified, which reveals a Phi work division percentage range 38−77%. For all AUTO scenarios evaluated with a single Phi, the speedup relative to the host does not exceed 2× until the value of m is larger than 4096. For $m < 4096$, AUTO Unspecified offers the best Phi performance.

Using two Phi coprocessors, AUTO Unspecified (×2) starts to show performance gains even for relatively small matrix dimensions, achieving a peak speedup of 5.4× (raw performance of 1420 GFLOP/s) relative to the host for DGEMM square matrix operations with $m > \sim 12\,000$. Analysis of the offload report at peak speedup indicates a computational work division of 13% on the host and ~43.5% on each coprocessor.

CAO (Figure 2) provides a peak speed-up of 2.8× relative to the host for square matrix DGEMM operations. The CAO Phi performance closely matches the AUTO 100% case. This is not unexpected since all DGEMM operations are executed on the coprocessor in both cases. Interestingly the CAO performance drops at values of m coinciding with matrix sizes of large powers of two.

Completely migrating matrix multiplication onto the GPU yields a square matrix DGEMM speedup of 1.5−3.5× relative to the host. The GPU outperforms CAO Phi for all square matrix sizes evaluated. AUTO Unspecified, which utilizes both the host and one Phi, exhibits a comparable square matrix DGEMM performance to the GPU for $m > 7168$. For $m < 4096$, the GPU outperforms the Phi.

Figures 3, 4, and 5 display the performance of the Phi and K20 GPU for migrating *nonsquare matrix* multiplications. The choices for matrix sizes and dimensions are strongly driven by quantum chemistry end use scenarios. The first matrix is taken to be the orbital coefficient matrix, the second matrix to contain integrals that can be processed in batches, and the summing index, k, taken as the number of atomic orbitals. Three different sizes (32, 768, and 1458 MB) for the first matrix were used. The ratio of rows to columns in the first matrix, m:k, increased from 1:1 to 1:110 as might occur if the transformation is to only a subset of the molecular orbitals. For the second matrix the number of rows is also k, but the number of columns, n, is allowed to vary freely. This reflects the fact that it is often chosen based on available memory and the idea of computing as many integrals as possible for transformation. Since the

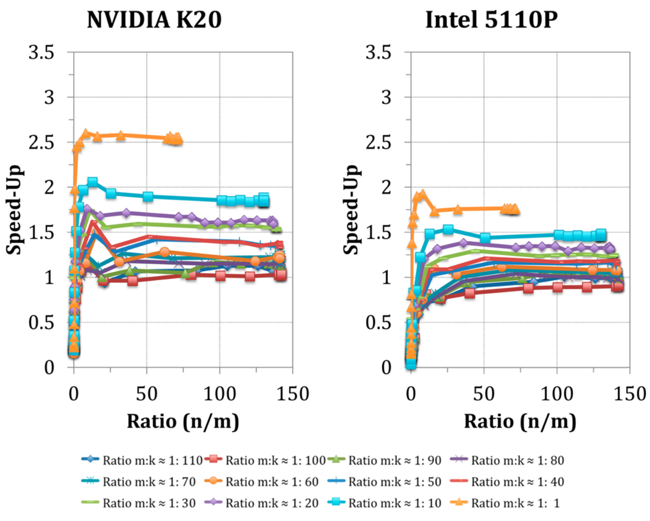


Figure 3. DGEMM performance on the Intel Xeon Phi 5110P coprocessor and NVIDIA Kepler K20 GPU for [*m*, *k*] × [*k*, *n*] = [*m*, *n*] using a 32 MB [*m*, *k*] matrix.

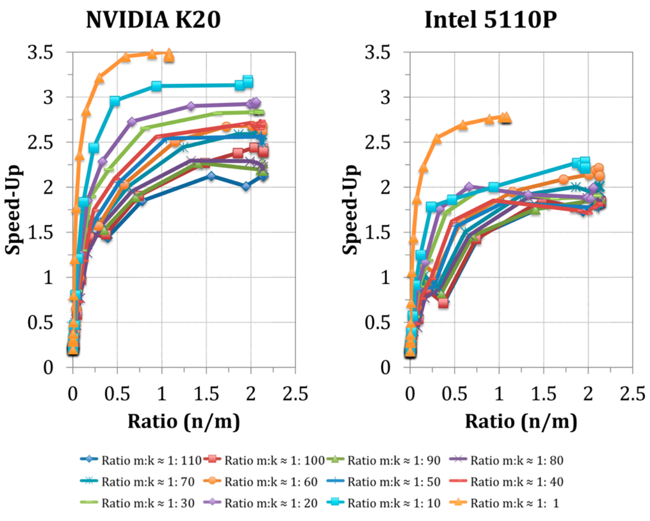


Figure 5. DGEMM performance on the Intel Xeon Phi 5110P coprocessor and NVIDIA Kepler K20 GPU for [*m*, *k*] × [*k*, *n*] = [*m*, *n*] using a 1458 MB [*m*, *k*] matrix.

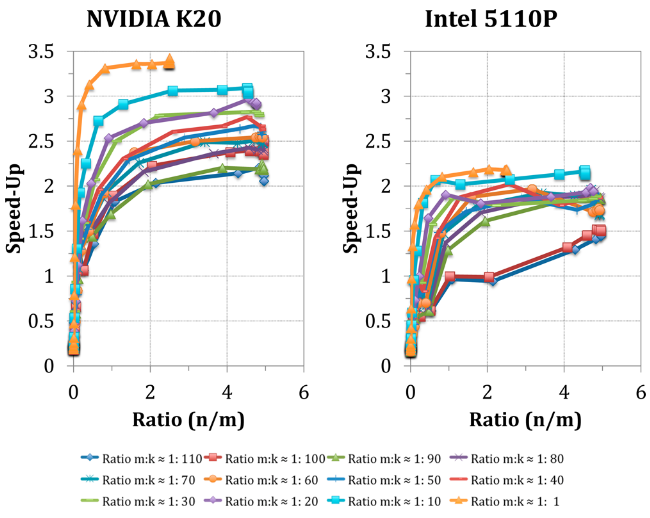


Figure 4. DGEMM performance on the Intel Xeon Phi 5110P coprocessor and NVIDIA Kepler K20 GPU for [*m*, *k*] × [*k*, *n*] = [*m*, *n*] using a 768 MB [*m*, *k*] matrix.

available memory is usually large, *n* is usually greater than both *m* and *k*. In all cases the Phi CAO model was used for comparison with the GPU and all DGEMM operations were executed on a single accelerator (the host was left idle).

Considering first the results for the *m*:*k* ratio fixed at 1:1, for *n*:*m* ratios <1 performance degrades quickly with little or no benefit resulting from use of an accelerator. This is most clearly evident in Figure 5. Fortunately however, as mentioned above, this situation is unlikely; for the more important case where the ratio of *n*:*m* is larger than one, performance is relatively flat. Now, for a fixed *n*:*m* ratio, performance consistently decreases with increasing *m*:*k* ratios; that is, for a given [*m*, *k*] matrix size and for *any* *n*:*m* ratio, an *m*:*k* ratio of 1:1 (square) provides an upper-bound to the performance that can be achieved on an accelerator for migrating matrix multiplication.

Relative to running on the host, the GPU provides the best overall performance for matrix multiplication involving nonsquare matrices. For a 32 MB [*m*, *k*] matrix (Figure 3), the Phi provides a speedup of ≤1.5× for all nonsquare [*m*, *k*] matrices tested, while for *m*:*k* ratios between 1:70 and 1:110, the Phi offers no performance improvement over running on the host. In contrast, the GPU provides a 1−1.5× speedup relative to the host for matrices with *m*:*k* ratios between 1:40 and 1:110. For *m*:*k* ratios less than 1:40, the GPU provides a speedup of 1.5−2×.

For Olestra with a cc-pVTZ basis set (occupied MO:AO ratio = 453:9142) or Vitamin B12 using an aug-cc-pVQZ basis set (occupied MO:AO ratio = 359:11557), migrating AO-to-MO transformations onto accelerators may offer little to no performance improvement relative to running on the host. For both accelerators, the square matrix performance observed for *m* = 2048 (2048 × 2048 double precision matrix = 32 MB) in Figure 2 can be improved by increasing the column dimension of the [*k*, *n*] matrix, as is evident from the performance observed for *n*/*m* values above 1 for *m*:*k* ∼ 1:1 in Figure 5. The latter is more pronounced for the GPU.

For a larger 768 MB [*m*, *k*] matrix (Figure 4), the performance is improved on both the Phi and GPU. The Phi exhibits a peak speedup range of 1.5−2.2×. Above *n*/*m* = 2, the Phi provides a speedup >1.5× for *m*:*k* ratios between 1:1 and 1:90. The GPU exhibits a speedup >2× when *n*/*m* ≥ 2. For BPTi using an aug-cc-pVQZ basis set (occupied MO:AO ratio =1748:56933), the GPU has a potential speedup of 2.8×, while the Phi only provides a potential speedup of 1.9×, relative to the host.

The performance for both accelerators continues to improve as the [*m*,*k*] matrix size increases. For a matrix size of 1458 MB (Figure 5), the Phi provides a speedup of 1.5−2× for *n*/*m* = 1. The GPU continues to outperform the Phi with speedups of 1.8−3.1× for *n*/*m* = 1. For *n*/*m* > 1, the performance for the Phi and GPU does not change significantly. The GPU appears to offer a better overall performance than the Phi for accelerating matrix multiplications involving nonsquare matrices that may be encountered when studying large molecular systems using large basis sets.

4. CONCLUDING REMARKS

The performance of migrating matrix multiplications to the Phi 5110P and K20 GPU was evaluated. For data-transfers over PCI-e, the GPU provides the best overall performance for data sizes up to 4096 MB with consistent upload and download rates between 5–5.6 GB/s and 5.4–6.3 GB/s, respectively. The Phi suffers from poor data-transfer performance for data sizes smaller than 8 MB, possibly due to the lack of pinned memory support in the CAO model.

Both the Phi and the GPU offered at least 70% of the theoretical peak double-precision performance speed-up relative to the host for DGEMM operations offloaded on to a single accelerator.

For square matrix multiplications, the GPU outperforms both AUTO and CAO using a single Phi. AUTO Unspecified, which uses both the host and Phi coprocessor for the computation, gives the closest performance to the GPU, but only for $m > 7168$. Compared to CAO on the Phi, the GPU does not appear to exhibit performance drops for square matrix sizes with large powers of two.

For migrating nonsquare matrix multiplications, the GPU outperforms the Phi for the matrix sizes and dimensions tested. The GPU provides a greater speedup relative to the host for a wider range of row to column ratios than the CAO approach on the Phi. Observed for both accelerators is a decrease in performance when the row to column ratio of the $[m, k]$ matrix decreases. However, the performance on both accelerators is observed to improve for a given row to column ratio as the size of the $[m, k]$ matrix increases.

Except for AUTO 75%, AUTO Unspecified and AUTO Unspecified (2×), none of the tests described here made use of the 16 cores on the host for overlapping computations. Therefore, the presented performance analyses may offer a lower bound to the potential speedups that can be achieved. On the other hand, the AO Unspecified case may represent the best upper bound to speedups that can be obtained when using a single Phi coprocessor for migrating matrix multiplications. Because matrix product operations are ubiquitous in many areas of science, the analyses presented here are broadly applicable.

A purported advantage of the Phi coprocessor is the ability to readily exploit its additional computational power without the need to develop a new algorithm or introduce a new code base. The AUTO model does allow users to immediately harness the additional computing power of the Phi by setting a few environmental variables. However, based on the analyses presented here, the Phi AUTO model underperforms relative to an under-utilized K20 GPU. That is, the best performing AUTO case evaluated with a single coprocessor made use of overlapping communication and overlapping computation with the host, while the GPU used blocking communication and blocking computation and left the 16 cores on the host idle.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: mark@si.msg.chem.iastate.edu.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This material is based upon work supported in part by a National Science Foundation SI^2 grant and in part by the Air Force Office of Scientific Research under AFOSR Award No. FA9550-12-1-0476. The computations performed for this work were done on the Iowa State University Cyence computer, provided by a National Science Foundation MRI award.

■ REFERENCES

(1) Gotz, A. W.; Williamson, M. J.; Xu, D.; Poole, D.; Grand, S. L.; Walker, R. C. *J. Chem. Theory Comput.* **2012**, *8*, 1542–1555.

(2) Asadchev, A.; Gordon, M. S. *J. Chem. Theory Comput.* **2012**, *8*, 4166–4176.

(3) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230–1236.

(4) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A.* **2008**, *112*, 2049–2057.

(5) De Prince, E. A., III; Hammond, J. R. *J. Chem. Theory Comput.* **2011**, *7*, 1287–1295.

(6) PetaChem. http://www.petachem.com (accessed Nov. 4, 2013).

(7) Gordon, M. S.; Schmidt, M. W. In *Theory and Applications of Computational Chemistry: The First Forty Years*; Dykstra, C. E., Frenking, G., Kim, K. S., Scuseria, G. E., Eds.; Elsevier: Amsterdam, 2005; Chapter 41, pp 1167–1189.

(8) Valiev, M.; Bylaska, E. J.; Govind, N.; Kowalski, K.; Straatsma, T. P.; Van Dam, H. J. J.; Wang, D.; Nieplocha, J.; Apra, E.; Windus, T. L.; de Jong, W. A. _*Comput. Phys. Commun.* **2010**, *181*, 1477–1489.

(9) Shao, Y.; Molnar, L. F.; Jung, Y.; Kussmann, J.; Ochsenfeld, C.; Brown, S. T.; Gilbert, A. T. B.; Slipchenko, L. V.; Levchenko, S. V.; O'Neill, D. P.; DiStasio, R. A., Jr; Lochan, R. C.; Wang, T.; Beran, G. J. O.; Besley, N. A.; Herbert, J. M.; Lin, C. Y.; Van Voorhis, T.; Chien, S. H.; Sodt, A.; Steele, R. P.; Rassolov, V. A.; Maslen, P. E.; Korambath, P. P.; Adamson, R. D.; Austin, B.; Baker, J.; Byrd, E. F. C.; Dachsel, H.; Doerksen, R. J.; Dreuw, A.; Dunietz, B. D.; Dutoi, A. D.; Furlani, T. R.; Gwaltney, S. R.; Heyden, A.; Hirata, S.; Hsu, C.-P.; Kedziora, G.; Khalliulin, R. Z.; Klunzinger, P.; Lee, A. M.; Lee, M. S.; Liang, W.; Lotan, I.; Nair, N.; Peters, B.; Proynov, E. I.; Pieniazek, P. A.; Rhee, Y. M.; Ritchie, J.; Rosta, E.; Sherrill, C. D.; Simmonett, A. C.; Subotnik, J. E.; Woodcock, H. L., III; Zhang, W.; Bell, A. T.; Chakraborty, A. K.; Chipman, D. M.; Keil, F. J.; Warshel, A.; Hehre, W. J.; Schaefer, H. F., III; Kong, J.; Krylov, A. I.; Gill, P. M. W.; Head-Gordon, M. *Phys. Chem. Chem. Phys.* **2006**, *8*, 3172–3191.

(10) Ufimtsev, I. S.; Martínez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222–231.

(11) Szabo, A.; Ostlund, N. S. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*; Macmillan: New York, 1982.

(12) ARK | Intel Xeon Processor E5-2650. http://ark.intel.com/products/64590 (accessed Nov. 2, 2013).

(13) NVIDIA GK110 Whitepaper. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf (accessed Nov. 2, 2013).

(14) ARK | Intel Xeon Phi Coprocessor 5110P. http://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core (accessed Nov. 2, 2013).

(15) Intel MKL Automatic Offload enabled functions for Intel Xeon Phi coprocessors. http://software.intel.com/en-us/articles/intel-mkl-automatic-offload-enabled-functions-for-intel-xeon-phi-coprocessors (accessed Nov. 2, 2013).

(16) OpenMP* Thread Affinity Control. http://software.intel.com/en-us/articles/openmp-thread-affinity-control (accessed Nov. 2, 2013).