Programming For Today, Developing for Tomorrow.

Sarom S. Leang, Theresa L. Windus, Mark S. Gordon

The limitations of Dennard Scaling and the desire to continue Moore's observation has resulted in a paradigm shift in computing hardware from higher clock speeds on a single CPU towards multiple computing cores on a single chip. In addition, the interest to minimize power consumption has resulted in a highly heterogeneous parallel environment where multi-core CPUs are augmented with a number of special purpose devices (accelerators), such as general-purpose graphical processing units or many-core coprocessors. These accelerators offer peak performances that can be well above those of the host processor. Exploiting this new heterogeneous environment requires novel software development tools, programming, and algorithm designs and poses a challenge for both new and legacy application codes.

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 and in quantum chemistry methods, such as Hartree-Fock, density functional theory, second order perturbation theory, and coupled cluster theory. Several popular electronic structure codes, such as TeraChem [1], GAMESS [2], NWChem [3], and Q-Chem [4], 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 in the literature, 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. Very rarely, if ever, is the CPU code utilized in practice with zero compiler optimizations and run on a single CPU core.

Recently, the Intel Phi accelerator is gaining some interest, mostly from the claims that the Intel Phi requires minimal programming effort; however, few comparisons of the Intel Phi with CPU and or GPU performance have been reported.

It is certainly true that minimal programming effort is needed to migrate basic linear algebra subroutine (BLAS) operations onto the Intel Phi. At the highest level of abstraction, the automatic offloading model, the Intel Phi can be engaged by simply setting an environmental. Automatic offloading utilizes both the host CPU and any available Intel Phi accelerators connected to the host to perform computations. Additional environmental variables can be set to control the work division between the host and the Intel Phi accelerator(s). The caveat of the automatic offloading model is the minimal dimensional requirement imposed on matrix sizes for different BLAS operations (for DGEMM: M, N > 1280, K > 256) before offloading will take place.

Finer control of the Intel Phi can be leveraged by the developer through the use of the compiler-assisted model. The latter utilizes pragmas within the host code to control data movement, absent in the automatic offloading model, and computations on the Intel Phi. From our experience, the coding effort necessary to utilize the compiler-assisted offloading model on the Intel Phi is similar to producing GPU code that employs the cuBLAS accelerated math library.

Both Intel and NVIDIA provide optimized math libraries for use with their accelerators. Utilizing vendor supplied accelerated math libraries such as Intel MKL or NVIDIA cuBLAS eliminates the need for software developers to produce optimal code for endemic operations found in quantum chemistry calculations such as double precision general matrix multiply (DGEMM). Our research has shown that the NVIDIA GPU (Kepler K20) outperforms the Intel Phi (5110P) for both square and non-square matrix multiplications [5]. We have also observed, for both vendor supplied accelerated math libraries, a decrease in performance relative to the host for DGEMM operations involving non-square matrices. The disparity in DGEMM performance between square and non-square matrices appears to be more severe for the Intel Phi.

Our findings highlight the need for a specialized accelerated math library for quantum chemistry. Such a library should offer optimized implementation of BLAS operations involving non-square matrices with support for one or more (possibly heterogeneous) accelerators with or without workload sharing on the host.

References:

1. PetaChem. http://www.petachem.com

2. 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
3. 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
4. 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
5. Leang, S. S.; Rendell, A. P.; Gordon, M. S. *J. Chem. Theory Comput.* **2014**, 10, 908-912