Evaluating one-sided programming models for GPU cluster computations

Jeff R. Hammond and A. Eugene DePrince III , Argonne National Laboratory
{jhammond,deprince}@mcs.anl.gov

Abstract — The Global Array toolkit (GA) [1] is a powerful framework for implementing algorithms with irregular communication patterns, such as those of quantum chemistry. On the other hand, accelerators such as GPUs have shown great potential for important kernels in quantum chemistry, for example, atomic integral generation [2] and dense linear algebra in correlated methods [3].

Integration of the global address space (GAS) programming model and associated one-sided protocols with GPU programming paradigms such as CUDA has the potential to revolutionize quantum chemistry by allowing the efficient use of very large clusters of heterogeneous nodes, such as the future multi-petaflop installation expected at Oak Ridge National Laboratory in 2012.

This paper reports on our preliminary investigations of the technical challenges and performance opportunities associated with cluster-GPU computation using the simplest approximation to quantum chemistry applications: parallel matrix-matrix multiplication (MMM). We focus on the role of asynchronous execution of network communication, device-to-host transfer, and kernel launch to understand the extent of latency-hiding that can be achieved for dense algorithms on large matrices.

I. Introduction

The hardware roadmap for exascale [4] includes many challenges, but two of the most obvious are (1) significant thread-level parallelism on a node and (2) significant node-level parallelism across a network with topology-dependent latency and bandwidth. The best approximation to the thousand-fold on-node parallelism today is a GPU, which possesses significant raw computing power with a constrained execution model (currently SIMD). Regardless of the network technology used to achieve exascale, hiding latency and dealing with decreasing bandwidth except across very limited distance will be critical. While much success has been achieved with single-GPU applications and with massively-parallel supercomputers with traditional processors (e.g. Cray XT5 and Blue Gene/P), successful application demonstrations using both accelerator processing and significant node-level parallelism are quite limited. The obvious reason is that applications must manage both network communication and node-level interprocessor communication, which is both challenging from a programming model perspective but can render latency-sensitive applications (e.g. molecular dynamics) effectively useless.

Quantum chemistry presents a tremendous opportunity for combined GPU and cluster parallelism due to (1) latency-insensitive algorithms, (2) relatively large and regular task granularity, and (3) large aggregate memory requirements. A good example of a method with such characteristics is the "gold-standard" coupled-cluster method CCSD(T) [5], [6]. This method has been demonstrative to efficiently utilize the supercomputers [7], [8], [9], [10], [11] with thousands to hundreds of thousands of processors at better than 50% efficiency because the majority of the floating-point computation is dense matrix-matrix multiplication (`SGEMM`[1]). The CCSD(T) method is realized in two stages: the iterative CCSD step, which is communication-intensive, and the non-iterative (T) step, which is more floating-point intensive but is weakly-coupled. The work required to extend the latter step to utilize attached processors such as GPUs is nominal, especially within the already massively-parallel chemistry package NWChem [12]. Less obvious is the difficulty and payoff associated with running the CCSD iterations on an accelerator due to the smaller task size and increased data-flow. Unlike the matrix-matrix multiplications (MMMs) performed in (T), which are all local, the CCSD iterations resemble, in an algorithmic sense, a few dozen parallel MMMs on matrices varying in size from megabytes to terabytes.

This paper explores how one might implement iterative algorithms in quantum chemistry with large clusters of GPUs by implementing parallel MMM using the de facto standard quantum chemistry run-time system, Global Arrays, and NVIDIA's GPU API, CUDA. We focus on attaining the greatest level of asynchronicity possible by exploiting non-blocking one-sided data-transfer calls in both APIs, hiding latency by overlapping communication and computation, by increasing absolute performance by delegating tasks to both the GPU and CPU, which poses a challenge due to their disparate performance. Preliminary results demonstrate that the opportunity for overlap within the hybrid processing environment is significant because of the obvious lack of contention for resources when the CPU is managing communication and only the GPU computes. More importantly, programming parallel MMM using GA and CUDA is nearly identical to the CPU-only case, indicating that the one-sided programming model of GA is naturally extendible to future accelerator-based supercomputers. Comparison of the CPU and GPU implementations of parallel MMM shows more than an order-of-magnitude advantage in performance for the GPU version despite the cost of data movement within the node.

[1]In this paper we consider only single-precision due to the deficient double-precision performance of the current generation of GPU hardware.

II. Methodology

To understand how GPU cluster computation functions in practice, simple benchmarks were performed on memory transfer rate and SGEMM performance. Since direct replacement of BLAS3 operations with their CUBLAS equivalents is desirable and, for the matrix sizes of quantum chemistry, quite feasible, we compare CPU and GPU SGEMM performance *with transfer time included*. Finally, we consider parallel MMM using different implementations on both CPU and GPU.

Memory transfer performance: Although memory transfer performance seems like a mundane issue and not worthy of investigation, in the context of Global Arrays (GA), it acquires new meaning. Both CUDA and ARMCI [13], the GA one-sided communication layer, require their own version of registered memory for maximum bandwidth. ARMCI registration is especially critical for Infiniband and Cray, the relevant interconnects for current and future GPU supercomputers. Unfortunately, as of the CUDA 3.0 driver, there is still no support for registering previously allocated memory with the GPU, hence, for a combined GA-CUDA application, one must choose between using registered memory from CUDA or from ARMCI *but not both*. Our measurements attempt to determine which option maximizes performance in various contexts.

SGEMM performance: While single-GPU applications are frequently built under the assumption that their entire state fits into device memory, quantum chemistry calculations have significant storage requirement which quickly exceed the 10s of gigabytes available on a single node, hence interesting calculations warranting a parallel computer necessarily involve inter-node communication and thus intra-node transfer if GPUs are to be used. Because of the complexity of the CCSD equations, it is extremely difficult to reuse data already stored on the GPU except for non-bottleneck procedures best performed on the CPU due to their small size. Enforcing strict data-locality to eliminate intra-node transfers would likely to a significant load-imbalance, especially at scale, so we consider only the case where large objects must be copied on and off the GPU for every kernel launch.

Parallel matrix-matrix multiplication performance: The final evaluation of GPU-cluster programmability with GA+CUDA in this paper is a simple implementation of parallel matrix-matrix multiplication. The algorithm is similar to the SUMMA [14] and SRUMMA [15] algorithms implemented in GA many years ago, with appropriate modifications related to transferring tiles to and from the GPU.

III. Preliminary Results

Hardware Details: Unless otherwise noted, all performance data reported here was obtained on the *lincoln* machine at NCSA, which has Intel Xeon E5345 CPUs, NVIDIA Tesla T10 GPUs with Infiniband SDR interconnect.

Memory transfer performance: Figure 1 demonstrates the role of registered CUDA memory on transfer bandwidth. The "in" (host-to-device) performance is diminished between 10^5 and 10^6 bytes using unregistered memory (allocated with posix_memalign and page-alignment), while "out" (device-to-host) bandwidth is approximately 40% worse using unregistered memory for all transfer sizes. The impact of unregistered memory on ARMCI performance is significantly less for, but becomes pathological at large scale [16]. A second issue is that CUDA asynchronous memory transfer operations can only be applied to registered memory segments. Thus, one must compromise between true one-sided and bandwidth-optimal communication within the node via CUDA and scalable one-sided performance between nodes via ARMCI. This represents an unpalatable choice for HPC chemistry applications.

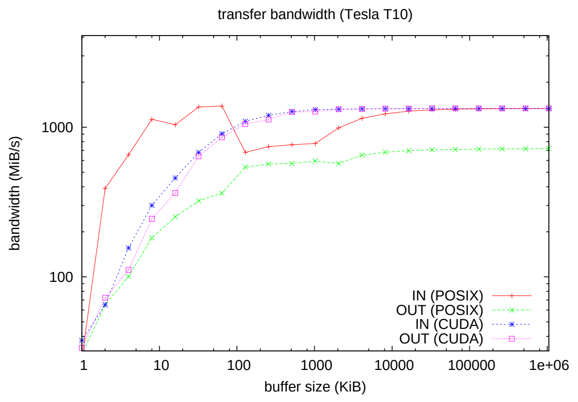

Fig. 1. Transfer bandwidth on Lincoln.

SGEMM performance: The performance of multi-threaded Intel MKL 11.1 SGEMM versus CUBLAS 2.3 for a variety of CPUs and GPUs is shown in Figure 2. While the Tesla T10 vastly outperforms the Intel Xeon present in Lincoln for all but the smallest matrices, more recent Intel processors increase the cross-over point to above dimension 1000, which limits the drop-in utility of GPUs significantly. The GPU performance drops significantly for irregular matrix sizes (*rank* mod 64 $\neq$ 0), which can be address by padding the global array manually.

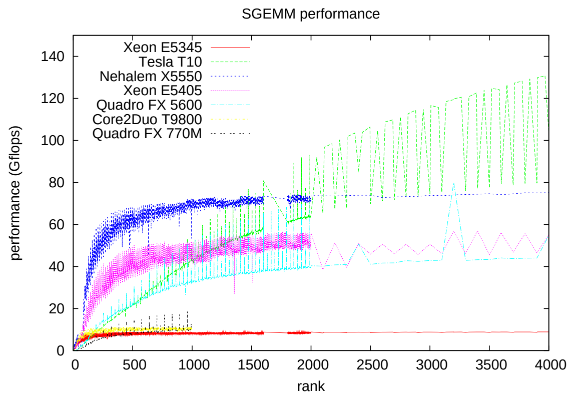

Fig. 2. SGEMM performance on a variety of processors.

Parallel matrix-matrix multiplication performance: Figures 3 and 4 show the scaling over a limited range of CPU or GPU nodes and the effect of increasing asynchronicity in the GPU implementation. Specifically, in the fully asynchronous implementation in Figure 4, we rearranged GA and CUDA communication calls to maximize overlap of both with CUBLAS asynchronous execution.

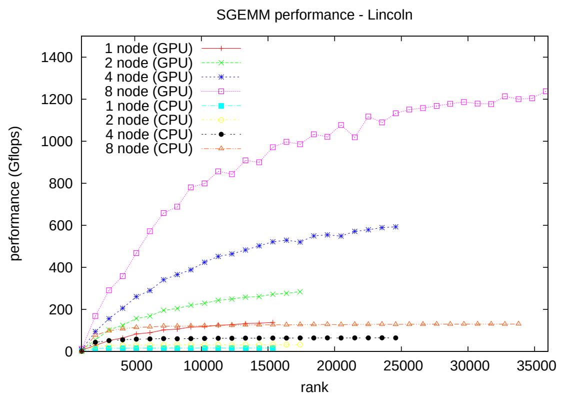

Fig. 3. Comparison of parallel MMM for various node counts using CPUs or GPUs.

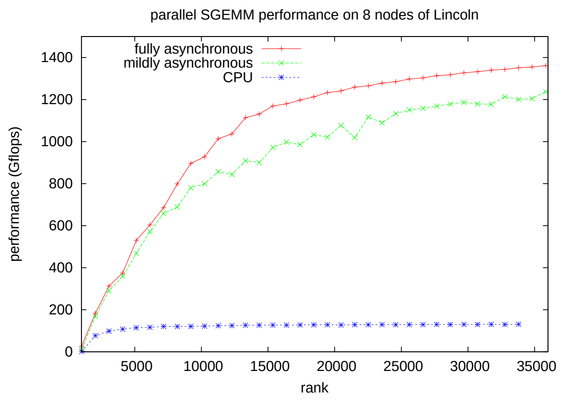

Fig. 4. Comparison of two different implementations of parallel MMM on 8 nodes.

IV. Conclusions and Future Work

Our preliminary results demonstrate that significant performance improvements can be achieved by simple drop-in replacement of CPU BLAS3 calls with their GPU equivalent within the Global Arrays programming model despite unresolved conflicts between the intra- and internode communication APIs (CUDA and ARMCI) regarding registered memory. This indicates the likely success of porting large-scale quantum chemistry calculations such as CCSD to large GPU-based clusters, especially if future versions of CUDA implement the necessary features to cooperate with RDMA interconnects.

In the near future, we will implement the complete version of SRUMMA (i.e. add prefetching to our existing implementation), utilize both the CPU and the GPU simultaneously, and add automated padding and GPU-oriented dimensioning (e.g. tiles and total dimension should be multiples of 64). All of these developments will be contributed to the Global Arrays toolkit for subsequent distribution.

Acknowledgments

We acknowledge an Argonne Director's Postdoctoral Fellowship, an Argonne Computational Postdoctoral Fellowship and the Argonne Laboratory Directed Research and Development (LDRD) program for support. This research was supported in part by the National Science Foundation through TeraGrid resources provided by NCSA (Lincoln) under grant number TG-MCB080110N.

References

[1] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global arrays: a portable "shared-memory" programming model for distributed memory computers," in *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 340–ff.
[2] T. J. M. Ivan S. Ufimtsev and, "Quantum chemistry on graphical processing units. 1. strategies for two-electron integral evaluation," *J. Chem. Theo. Comp.*, vol. 4, no. 2, pp. 222–231, 2008.
[3] L. Vogt, R. Olivares-Amaya, S. Kermes, Y. Shao, C. Amador-Bedolla, and A. Aspuru-Guzik, "Accelerating resolution-of-the-identity second-order MøllerPlesset quantum chemistry calculations with graphical processing units," *J. Phys. Chem. A*, vol. 112, no. 10, pp. 2049–2057, 2008.
[4] A. Geist, "Paving the roadmap to exascale."
[5] K. Raghavachari, G. W. Trucks, J. A. Pople, and M. Head-Gordon, "A fifth-order perturbation comparison of electron correlation theories," *Chem. Phys. Lett.*, vol. 157, pp. 479–483, May 1989.
[6] K. Raghavachari, J. A. Pople, E. S. Replogle, and M. Head-Gordon, "Fifth order moeller-plesset perturbation theory: comparison of existing correlation methods and implementation of new methods correct to fifth order," *J. Phys. Chem.*, vol. 94, pp. 5579–5586, 1990.
[7] L. Pollack, T. L. Windus, and W. A. de Jong, "Thermodynamic properties of the C5, C6, and C8 n-alkanes from ab initio electronic structure theory," *J. Phys. Chem. A*, vol. 109, no. 31, pp. 6934–6938, 2005.
[8] E. Aprà, R. J. Harrison, W. A. Shelton, V. Tipparaju, and A. Vazquez-Mayagoitia, "Computational chemistry at the petascale: Are we there yet?" *J. Phys.: Conf. Ser.*, vol. 180, p. 012027 (6pp).
[9] E. Aprà, A. P. Rendell, R. J. Harrison, V. Tipparaju, W. A. de Jong, and S. S. Xantheas, "Liquid water: obtaining the right answer for the right reasons," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–7.
[10] K. Kowalski, J. R. Hammond, W. A. de Jong, and A. J. Sadlej, "Coupled cluster calculations for static and dynamic polarizabilities of C_{60}," *J. Chem. Phys.*, vol. 129, no. 22, pp. 226 101–226 103, 2008.
[11] K. Kowalski, S. Krishnamoorthy, O. Villa, J. R. Hammond, and N. Govind, "Active-space completely-renormalized equation-of-motion coupled-cluster formalism: Excited-state studies of green fluorescent protein, free-base porphyrin, and oligoporphyrin dimer," *J. Chem. Phys.*, vol. 132, p. 154103, 2010.
[12] E. J. Bylaska et al., "NWChem, a computational chemistry package for parallel computers, version 5.1.1," 2009.
[13] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. K. Panda, "High performance remote memory access communication: The armci approach," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 233–253, 2006.
[14] R. A. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.
[15] M. Krishnan and J. Nieplocha, "Srumma: A matrix multiplication algorithm suitable for clusters and scalable shared memory systems," *Int. Par. and Dist. Proc. Symp.*, vol. 1, p. 70b, 2004.
[16] S. Krishnamoorthy, S. Shende, J. R. Hammond, N. A. Romero, and A. D. Malony, "Nwchem workload characterization using the tau performance system," p. 10, 2010.