Early Experiences in Running Many-Task Computing Workloads on GPGPUs

Scott J. Krieder*, Benjamin Grimmer*, Ioan Raicu*†
*Department of Computer Science, Illinois Institute of Technology
†MCS Division, Argonne National Laboratory

Abstract—**This work aims to enable Swift to efficiently use accelerators (such as NVIDIA GPUs) to further accelerate a wide range of applications. This work presents preliminary results in the costs associated with managing and launching concurrent kernels on NVIDIA Kepler GPUs. We expect our results to be applicable to several XSEDE resources, such as Forge, Keeneland, and Lonestar, where currently Swift can only use the general processors to execute workloads and the GPUs are left idle.**

Keywords-**Many-Task Computing, Swift, GPGPU, CUDA, OOPS**

I. Introduction

Many-task computing (MTC) [1] aims to bridge the gap between two computing paradigms, high throughput computing (HTC) and high-performance computing (HPC). MTC emphasizes using many computing resources over short periods of time to accomplish many computational tasks (i.e. including both dependent and independent tasks), where the primary metrics are measured in seconds. MTC denotes high-performance computations comprising multiple distinct activities, coupled via file system operations. Swift is a particular implementation of the MTC paradigm, and is a parallel programming system that has been successfully used in many large-scale computing applications across the TeraGrid and now XSEDE. [2] It has been adopted by the scientific community as a great way to increase productivity in running complex applications via a dataflow driven programming model, which intrinsically allows implicit parallelism to be harnessed based on data access patterns and dependencies. Swift has been shown to run well on tens of thousands of nodes with task graphs in the range of hundreds of thousands of tasks. This work aims to enable Swift to efficiently use accelerators (such as NVIDIA GPUs and Intel Xeon Phi [3]) to further accelerate a wide range of applications. This work evaluates a real biochemistry application, namely the the Open Protein Simulator (OOPS) [4], which builds on the Protein Library (PL). OOPS is multipurpose and allows extensions to perform various simulation tasks relevant for life scientists, such as protein folding or protein structure prediction. We have taken parts of this application and ported to NVIDIA GPUs via the CUDA programming language, in order to accelerate OOPS computations via Swift. This work presents preliminary results in the costs associated with managing and launching concurrent kernels on NVIDIA FERMI GPUs, through the Swift system. We expect that our results to be applicable to several XSEDE resources, such as Forge, Keeneland, and Lonestar, where currently Swift can only use the general processors to execute workloads and the GPUs are left idle.

II. Many-Task Computing

Many-task computing (MTC) aims to bridge the gap between two computing paradigms, high throughput computing (HTC) and high-performance computing (HPC). MTC emphasizes using many computing resources over short periods of time to accomplish many computational tasks (i.e. including both dependent and independent tasks), where the primary metrics are measured in seconds. MTC denotes high-performance computations comprising multiple distinct activities.

III. Swift

Swift is a particular implementation of the MTC paradigm, and is a parallel programming system that has been successfully used in many large-scale computing applications across the TeraGrid and now XSEDE. It has been adopted by the scientific community as a great way to increase productivity in running complex applications via a dataflow driven programming model, which intrinsically allows implicit parallelism to be harnessed based on data access patterns and dependencies.

IV. Accelerators and Coprocessors

There are currently three major players in the hardware accelerator market including NVIDIA GPUs, AMD GPUs [5], and the Intel Xeon Phi. Running CUDA on NVIDIA GPUs is one of the most mature GPGPU solutions and provides high raw computational performance, however this does require code ported to the CUDA platform. The Intel Xeon Phi suffers from a lack of availability, but once this device is highly available it should bring large improvements in regards to accelerator programmability due to the familiar x86 environment. Finally, AMD GPUs provide a high level of openness in regards to programmability. AMD supports open standards such as OpenCL but may see difficulty in adoption within the HPC markets due to performance.[6]

V. Scheduler Architecture

- Without a scheduler, jobs run on the GPU following the pattern copycompute-copy. - Copy-compute-copy produces inefficiencies. Some of which are shown in the red area above. - Our scheduler sits in-between Swift and GPU. Handles multiple inputs from Swift and condenses these into single

GPU calls. - Use of a scheduler allows for overlapping kernel execution - Our scheduler overlaps data transfers from last solution and next problem to increase efficiency.

VI. Evaluation

VII. Future Work

Future work aims to migrate our scheduler into the GPU, allowing a daemon SuperKernel to manage the MicroKernels, through the use of CUDA concurrent kernels. We also expect our performance to increase on the latest NVIDIA Kepler Architecture. [7]

By integrating our GPGPU MTC Scheduler into Swift we will be able to provide GPU support for MTC applications that utilize Swift. In addition this will apply the dataflow model to GPUs and provide implicit parallelism at the task level.

Finally, we believe that the Intel Xeon Phi will provide an array of added benefits for running MTC workloads. Future work will examine how Intel Xeon Phi performs for MTC workflows.

VIII. Conclusion

In conclusion we presented a scheduler which provides concurrent kernel execution from built in library of CUDA kernels. This scheduler overlaps memory transfers for increased performance. The scheduler supports concurrent kernels limited only by CUDA contraints up to 16 kernels. Finally, our scheudler eliminates Kernel execution overheads and enables workloads with coarse granular kernels.

References

[1] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford, "Toward loosely coupled programming on petascale systems," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 22.

[2] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, reliable, loosely coupled parallel computation," in *Services, 2007 IEEE Congress on*. IEEE, 2007, pp. 199–206.

[3] Intel. (2012, May) Intel many integrated core architecture - advanced. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html

[4] A. N. Adhikari, J. Peng, M. Wilde, J. Xu, K. F. Freed, and T. R. Sosnick, "Modeling large regions in proteins: Applications to loops, termini, and folding," *Protein Science*, vol. 21, no. 1, pp. 107–121, 2012.

[5] AMD. (2012, May) Opencl zone — amd. [Online]. Available: http://developer.amd.com/resources/heterogeneous-computing/opencl-zone/

[6] S. J. Krieder and I. Raicu, "An overview of current and future computing accelerator architectures," 1st Greater Chicago Area System Research Workshop Poster Session, 2012.

[7] NVIDIA. (2012, May) Nvidia kepler compute architecture — high performance computing — nvidia. [Online]. Available: NVIDIA Kepler - http://www.nvidia.com/object/nvidia-kepler.html