Towards Efficient Many-Task Computing on Accelerators in High-End Computing Systems

Scott J. Krieder*, Benjamin Grimmer*, Dustin Shahidehpour*, Jeffrey Johnson*
Justin M. Wozniak†, Michael Wilde†‡, Ioan Raicu*†
*Department of Computer Science, Illinois Institute of Technology
†MCS Division, Argonne National Laboratory
‡Computation Institute, Univeristy of Chicago

***Abstract*—Current software and hardware limitations prevent Many-Task Computing (MTC) workloads from leveraging hardware accelerators boasting ManyCore Computing architectures. This work aims to address the programmability gap between MTC and accelerators, through the innovative CUDA middleware GeMTC. By working at the warp level, GeMTC enables heterogeneous task scheduling and 10x number of workers compared to CUDA. In order to span multiple accelerators across nodes, we have adopted the Swift parallel programming system, which can both support ne grained millisecond tasks and extreme scale supercomputers at 100K-cores.**

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

I. Introduction

II. GeMTC

III. Swift/T

Implicitly Parallel Scripting Language Data-ow driven scheduling of parallel tasks Distributed executor eliminates centralized bottlenecks Optimizing compiler detects errors, improves efciency Scales to 100k cores Portable to most MPI-based clusters Syntax similar to C, Java

IV. Preliminary Results

V. Conclusions and Future Work

Designed GeMTC Framework Improved Memory Management Integrated GeMTC + Swift/T Evaluated Synthetic Benchmarks

Abstract for other Accelerators Evaluate Real Applications Improve Current Performance

VI. Problem Statement

This work aims to provide an integration between data-flow driven parallel programming systems (e.g. Many-Task Computing - MTC) and hardware accelerators [1] (e.g. NVIDIA GPUs, AMD GPUs, and the Intel MIC). 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.[2] 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. [3] The scientific community has adopted Swift 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 is a parallel programming system that fits the MTC model, and 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 MIC) to further accelerate a wide range of applications, on a growing portion of high-end systems.

VII. Description of Novel Approach

The most recent version of Swift, namely Swift/T, supports function calls.[4] By plugging our middleware into the Swift/T accelerator API(which we are currently pursuing a collaboration to develop) we plan to have Swift call C wrapper functions to CUDA kernels/applications directly. Currently CUDA developers may only have a maximum of 16 kernels running concurrently, one kernel per streaming multiprocessor (SM). The problem is that all kernels have to start and end at the same time, causing extreme inefficiencies in heterogeneous workloads. By working at the warp level we trade local memory for concurrency and we expect to be able to run up to 96 concurrent kernels. Our work will develop a middleware that will allow independent kernels (MIMD style) to be launched and managed on many-core architectures that traditionally only support SIMD. [5]

We are also currently evaluating a real biochemistry application, namely the Open Protein Simulator (OOPS), 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.[6] 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. We already have preliminary results in the costs associated with managing and launching concurrent kernels on NVIDIA FERMI GPUs, through the Swift system. We expect our results to be applicable to many HPC resources where GPUs are now common. For example the June 2012 Top 500 machines that are GPU enabled include Tianhe-1A, Jaguar, and Nebulae. Currently, Swift can only utilize the general processors on

these machines to execute workloads and the GPUs are left idle. We will also explore many more applications from different domains, such as medicine, economics, astronomy, bioinformatics, physics, and many more.

VIII. Contributions

We plan to continue to push the performance envelope by enabling many MTC applications and systems to leverage the growing number of accelerated high-end computing systems. We also expect this work to enable other classes of applications to leverage accelerators, such as MapReduce and ensemble MPI. We also hope to influence future accelerator architectures by highlighting the need for hardware support for MIMD workloads.

References

[1] 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.

[2] 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.

[3] 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.

[4] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/t: Large-scale application composition via distributedmemory data flow processing," in *Proc. CCGrid*, vol. 13.

[5] S. J. Krieder, B. Grimmer, and I. Raicu, "Early experiences in running many-task computing workloads on gpgpus," XSEDE Poster Session, 2012.

[6] 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.