Accelerating Scientific Workflow Applications with GPUs

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

***Abstract*—This work analyzes the performance increases gained from enabling Swift applications to utilize the GPU through the GeMTC Framework. By identifying computationally intensive portions of Swift applications, we can easily turn these code blocks into GeMTC microkernels. Users can then call these microkernels throughout the lifetime of their Swift application. The GeMTC API handles task overlap and data movement, providing transparent GPU acceleration for the user. This work highlights preliminary performance results from the scientific application MDProxy. This application determines the energy of particles in a modeled universe as they move around in space.**

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

I. Background Information

GeMTC (GPU enabled Many-Task Computing) [1], is a CUDA-based framework which provides efficient support for Many-Task Computing [2] workloads on accelerators. [3] The GeMTC framework has been integrated into Swift/T [4], a parallel programming framework from Argonne National Laboratory and the University of Chicago, providing GPU functionality for the Swift language. [5]

A microkernel is a traditional CUDA kernel that is modified to run in the GeMTC framework. A CUDA kernel is a user-defined function that runs on a NVIDIA GPU.

II. MDProxy Architecture

In Figure 1 the call stack architecture is shown for MDProxy through Swift and GeMTC. The user writes a Swift script that will build an array of potential particles and calls GeMTC MDProxy with this array as a parameter. Each call to MDProxy creates it's own universe of particles and ships the universe to the GPU. Finally the MDProxy application consists of three functions 1) initialize the universe, 2) run the computation, and 3) update the result.

III. Testing Environment

In this work we conduct our evaluation on a GTX 670 GPU with 7 Streaming Multiprocessors (SMXs). In addition this GPU contains 84 Warps(utilized as workers), 1344 CUDA Cores, and 2GB of DDR5 RAM. CPU results are tested on a 6 core 3Ghz AMD CPU.

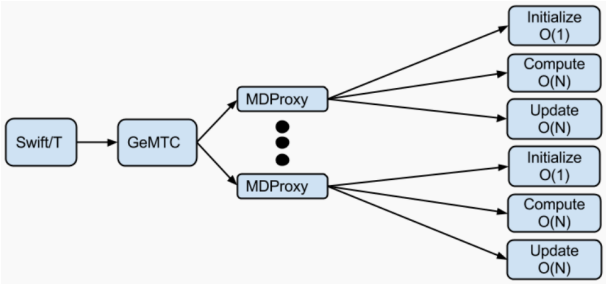

Fig. 1. Call stack architecture for the MDProxy implementation.

IV. MDProxy Evaluation

In Figure 2 the GeMTC MDProxy micro-kernel is evaluated with 2,688 particles and scaled up to 900 steps of computation from within the application. In addition, a comparison is drawn against a threaded CPU implementation of MDProxy with the GPU version showing a 10x speedup. Finally, Figure 3 shows how tasks per second are calculated based on varying the number of particles per universe. This work achieves almost 12k tasks per second for workloads with 500 particles in a given universe.

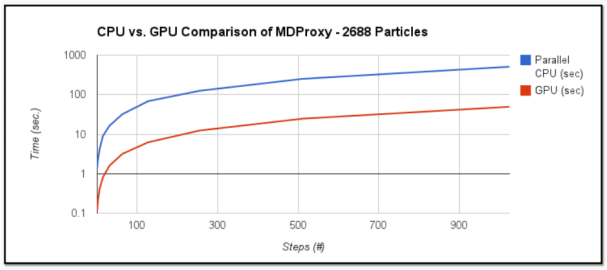

Fig. 2. MDProxy evaluation of the GPU vs. CPU implementations.

V. Conclusions

In conclusion this work aimed to evaluate a real scientific application. While the MD code evaluated here is not a production code it provides a skeleton of data movement and compute that is performed by MD Codes. MDProxy highlights the GeMTC potential by launching small compute universes on each compute element in the GPU. Finally, our preliminary

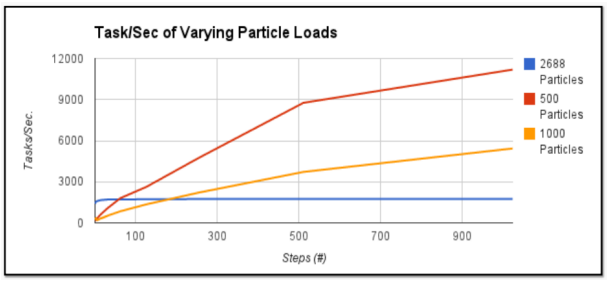

Fig. 3. Tasks per second achieved during MDProxy.

results show that GeMTC 10x faster than a threaded CPU implementation.

VI. Future Work

Future work will aim to improve the MD algorithm, this will provide for a more through analysis of the potential improved performance that GeMTC can provide. We will also extend our evaluation by testing MDProxy at Multi-Node scale. MDProxy provides an array of input parameters that affect the computation, further investigation will aim to find optimal task sizes. Finally, we aim to compare performance against other GeMTC-enabled accelerators and develop high level abstractions for the Swift/T + GeMTC stack while Expanding the library of GeMTC microkernels.

References

[1] S. J. Krieder and I. Raicu, "Towards the support for many-task computing on many-core computing platforms," Doctoral Showcase, IEEE/ACM Supercomputing/SC, 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] 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.

[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] 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.