Hybrid Dataflow Programming on Blue Waters

Scott J. Krieder*, Justin M. Wozniak†, Timothy Armstrong‡,
Michael Wilde†‡, Daniel S. Katz§, Ian T. Foster,†‡§ Ioan Raicu*†
*Department of Computer Science, Illinois Institute of Technology
†MCS Division, Argonne National Laboratory
‡Department of Computer Science, University of Chicago
§Computation Institute, University of Chicago & Argonne National Laboratory

***Abstract*—This work presents the analysis of hybrid dataflow programming over XK7 nodes of Blue Waters using a novel CUDA framework GeMTC. GeMTC is an execution model and runtime system that enables accelerators to be programmed with many concurrent and independent tasks of potentially short or variable duration. With GeMTC, a broad class of such "many-task" applications can leverage the increasing number of accelerated and hybrid high-end computing systems. GeMTC overcomes the obstacles to using GPUs in a many-task manner by scheduling and launching independent tasks on hardware designed for SIMD-style vector processing. We demonstrate the use of a high-level MTC programming model (the Swift parallel dataflow language) to run tasks on many accelerators and thus provide a high-productivity programming model for the growing number of supercomputers that are accelerator-enabled. While still in an experimental stage, GeMTC can already support tasks of fine (subsecond) granularity and execute concurrent heterogeneous tasks on 86,000 independent GPU warps spanning 2.7M GPU threads on Blue Waters.**

I. Introduction

This work explores methods for, and potential benefits of, applying the increasingly abundant and economical general-purpose graphics processing units (GPGPU) to a broader class of applications. It extends the utility of GPGPU from the class of heavily vectorizable applications to irregularly-structured many-task applications. Such applications are increasingly common, stemming from both problem-solving approaches (i.e., parameter sweeps, simulated annealing or branch-and-bound optimizations, uncertainty quantification) and application domains (climate modeling, rational materials design, molecular dynamics, bioinformatics).

In many-task computing (MTC) [1], tasks may be of short (even subsecond) duration or highly variable (ranging from milliseconds to minutes). Their dependency and data passing characteristics may range from many similar tasks to complex, and possibly dynamically determined, dependency patterns. Tasks typically run to completion: they follow the simple input-process-output model of procedures, rather than retaining state as in web services or MPI processes.

II. GeMTC Architecture

Fig. 1 shows a high-level diagram of GeMTC [2] driven by tasks generated by the Swift [3] parallel functional dataflow language (described in Section IV). GeMTC launches a daemon on the GPU that enables independent tasks to be multiplexed onto warp-level GPU workers. A work queue in GPU memory is populated from calls to a C-based API, and GPU workers pick up and execute these tasks. After a worker has completed a computation, the results are placed on an outgoing result queue and returned to the caller.

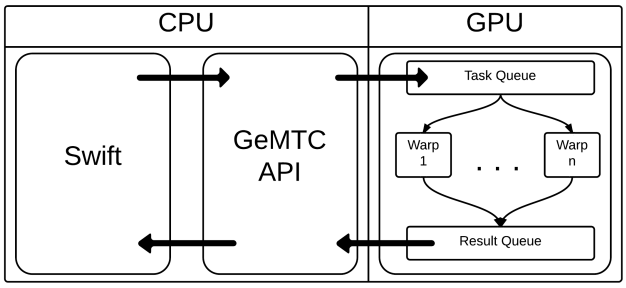

Fig. 1. Flow of a task in GeMTC driven by Swift.

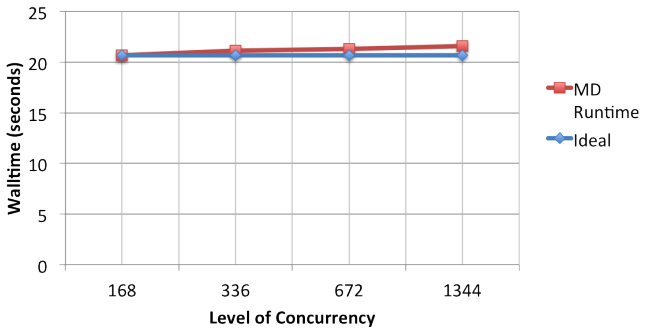

Fig. 2. GeMTC and MDLite scaling over 1344 workers on Blue Waters.

III. Performance Evaluation

Fig. 2 is a multinode scaling experiment where the number of simulations is set equal to the number of workers. At each data point there are two times as many workers as the previous, so we run twice as much work. In an ideal system without any overheads we would expect a flat line demonstrating the ability to conduct the same amount of work at each step. Even after 8 nodes we achieve 96% utilization. Future work aims to evaluate our system at even larger scales on Blue Waters.

Fig. 3 highlights throughput rates for Swift and GeMTC on a Kepler K20X. In this benchmark the maximum number of available GeMTC workers is enabled (168). Fig. 4

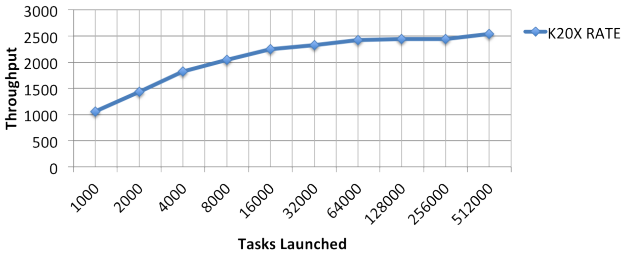

Fig. 3. GeMTC + Swift throughput on a XK7 node of Blue Waters.

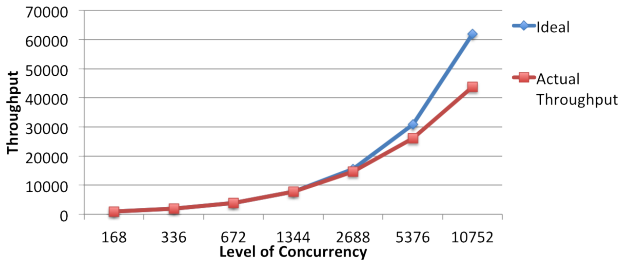

Fig. 4. GeMTC + Swift Throughput over 10,000 GPU workers.

demonstrates that Swift + GeMTC is capable of driving a high throughput of fine grain GPU work over many nodes on Blue Waters. As shown in Fig. 4 we obtain ∼70% of ideal throughput with 10,000-way concurrency.

Fig. 5 demonstrates an upper bound of GeMTC by launching efficiency workloads on multiple GPU nodes with only a single active GeMTC worker per GPU. We next enable 168 GeMTC warp workers per GPU (the maximum) and evaluate the efficiency of workflows with varied task granularities up to 86k individually operating GPU workers of Blue Waters. After adding 167 additional workers per GPU we do require longer lasting tasks to achieve high efficiency. We attribute this drop in performance to greater worker contention on the device queues and the fact that Swift must now drive 168 times the amount of work per node. In Fig. 6 we observe that tasks exceeding ∼1 second achieve high efficiency up to scales of 40K workers. Although we have not yet identified the cause for this drop in performance, we expect that the performance degradation at extreme levels of concurrency comes from the loading of shared libraries from the remote parallel filesystem. In future work we will continue to improve

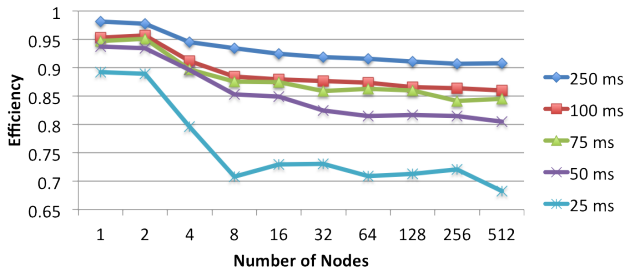

Fig. 5. GeMTC + Swift efficiency up to 512 nodes, 1 GeMTC worker.

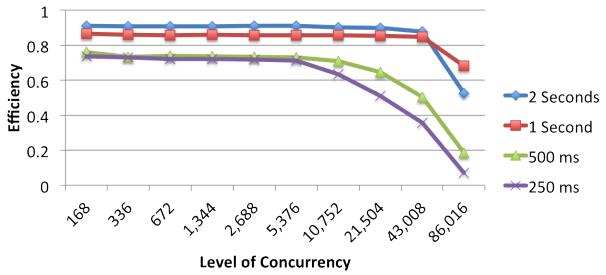

Fig. 6. Efficiency for workloads with varied task granularities up to 86K independent warps of Blue Waters. 168 active workers/GPU.

systemwide performance by reducing the reliance on dynamic loadable shared libraries and through larger scale evaluation on all 4K XK7 nodes of Blue Waters.

While we observe a drop in performance moving from 1 worker to 168 workers, we achieve 168x the amount of work with only 5x increase in time. In addition, these numbers improve even more when the time for computing vs. data transfer increases. In future work we will continue to improve systemwide performance and evaluation at even larger scale.

IV. Conclusions and Future Work

We have presented GeMTC, a framework for enabling MTC workloads to run efficiently on the XK7 nodes of Blue Waters. GeMTC encompasses the entire GPU running as a single GPU application similar to a daemon that receives and schedules work from the host through a C-API. GeMTC simplifies the programming model of the GPU by allowing GPUs to be treated as a collection of independent SIMD workers, enabling a MIMD view of the device. Future work also includes performance evaluation of diverse application kernels (e.g., data-pipelining, detecting cancer-related genes, glass modeling, and protein structure simulation); analysis of the ability of such kernels to effectively utilize concurrent warps; enabling of virtual warps which can both subdivide and span physical warps; support for other accelerators such as the Xeon Phi; and continued performance refinement.

V. Acknowledgements

We gratefully acknowledge support from NSF OCI-0725070 (Blue Waters), NSF ACI-1148443 (Swift), DOE DE-SC0005380 (ExM), and a generous donations from the Starr Foundation to the Illinois Institute of Technology. Work by Katz was supported by NSF while working at the Foundation. Any opinion, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

[1] I. Raicu, *Many-task computing: bridging the gap between high-throughput computing and high-performance computing*. ProQuest, 2009.

[2] S. K. et al., "Design and Evaluation of the GeMTC Framework for GPU-enabled Many-Task Computing," in *HPDC*. ACM, 2014.

[3] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, pp. 633–652, 2011.