DOE-PURDUE-08629

SLEEC: Semantics-Rich Libraries for Effective Exascale Computation

Final Technical Report

Period Covered: September 2012—August 2017
Award #: DE-SC0008629

April 2018

Milind Kulkarni

School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47906

Contents

1 Introduction and Summary

SLEEC (Semantics-rich Libraries for Effective Exascale Computation) was a project funded by the Department of Energy X-Stack Program, award number DE-SC0008629. The initial project period was September 2012–August 2015. The project was renewed for an additional year, expiring August 2016. Finally, the project received a no-cost extension, leading to a final expiry date of August 2017.

Modern applications, especially those intended to run at exascale, are not written from scratch. Instead, they are built by stitching together various carefully-written, hand-tuned libraries. Correctly composing these libraries is difficult, but traditional compilers are unable to effectively analyze and transform across abstraction layers. Domain specific compilers integrate semantic knowledge into compilers, allowing them to transform applications that use particular domain-specific languages, or domain libraries. But they do not help when new domains are developed, or applications span multiple domains. SLEEC aims to fix these problems. To do so, we are building generic compiler and runtime infrastructures that are semantics-aware but not domain-specific. By performing optimizations related to the semantics of a domain library, the same infrastructure can be made generic and apply across multiple domains.

SLEEC made contributions in three broad areas:

1. **Optimized Communication Between CPU and GPU**. To support the increasing use of GPUs and other accelerators, programmers often adopt an *offloading* approach, where some computations are delegated to the accelerator. Critical to making this offloading approach effective is determining which *data* needs to be moved to the accelerator. Our strategy is to take a domain-aware approach, where by understanding the semantics of the operations being performed on the accelerator, we can perform exactly the computations needed [2, 1]. We also leveraged similar ideas to accelerate GPU-offloaded tree computations [3].

2. **Domain-aware scheduling and computation optimization**. In addition to optimizing communication, we investigated strategies for exploiting domain semantics to optimize computational science applications. Our approach led to domain-agnostic (but domain *aware*) optimization strategies for an in-house computational mechanics code [6] that we extended to Peridigm, a peridynamics code from Sandia, and LULESH [7, 8].

3. **Application optimization**. Finally, we leveraged insights from our optimization strategies to develop optimized applications. These include an optimized, multi-scale peridynamics algorithm [5] and a domain-aware, optimized partitioning algorithm [4].

The following sections provide brief overviews of these efforts.

2 Communication Optimization

2.1 SemCache

Recently, GPU libraries have made it easy to improve application performance by offloading computation to the GPU. However, using such libraries introduces the complexity of manually handling explicit data movements between GPU and CPU memory spaces. Unfortunately, when using these libraries with complex applications, it is very difficult to optimize CPU-GPU communication between multiple kernel invocations to avoid redundant communication.

In this paper, we introduce SemCache, a semantics-aware GPU cache that automatically manages CPU-GPU communication and dynamically optimizes communication by eliminating redundant transfers using caching. It's key feature is the use of library semantics to determine the appropriate caching granularity for a given offloaded library (e.g., matrices in BLAS). We applied SemCache to BLAS libraries to provide a GPU drop-in replacement library which handles communications and optimizations automatically. Our caching technique is efficient; it only tracks matrices instead of tracking every memory access at fine granularity. Experimental results show that our system can dramatically reduce redundant communication for real-world computational science application and deliver significant performance improvements, beating GPU-based implementations like CULA and CUBLAS.

This work appeared in AlSaber and Kulkarni, ICS 2013 [2].

2.2 SemCache++

Offloading computations to multiple GPUs is not an easy task. It requires decomposing data, distributing computations and handling communication manually. GPU drop-in libraries (which require no program rewrite) have made it easy to offload computations to multiple GPUs by hiding this complexity inside library calls. Such encapsulation prevents the reuse of data between successive kernel invocations resulting in redundant communication. This limitation exists in multi-GPU libraries like CUBLASXT.

In this paper, we introduce SemCache++, a semantics-aware GPU cache that automatically manages communication between the CPU and multiple GPUs in addition to optimizing communication by eliminating redundant transfers using caching. SemCache++ is used to build the first multi-GPU drop-in replacement library that (a) uses the virtual memory to automatically manage and optimize multi-GPU communication and (b) requires no program rewriting or annotations. Our caching technique is efficient; it uses a two level caching directory to track matrices and submatrices. Experimental results show that our system can eliminate redundant communication and deliver performance improvements over multi-GPU libraries like StarPU and CUBLASXT.

This work appeared in AlSaber and Kulkarni, ICS 2015 [1].

2.3 Locality-aware GPU Execution

GPGPUs deliver high speedup for regular applications while remaining energy efficient. In recent years, there has been much focus on tuning irregular, task-parallel applications and/or the GPU architecture in order to achieve similar benefits for irregular applications running on GPUs. While most of the previous works have focused on minimizing the effect of control and memory divergence, which are prominent in irregular applications and which degrade the performance, there has been less attention paid to decreasing cache pressure and hence improving performance of applications given the small cache sizes on GPUs.

In this paper we tackle two problems. First we extract data parallelism from irregular task parallel applications, which we do by subdividing each task into sub tasks at the CPU side and sending these sub tasks to the GPU for execution. By doing so we take advantage of the massive parallelism provided by the GPU. Second, to mitigate the memory demands of many tasks that access irregular data structures, we schedule these subtasks in a way to minimize the memory footprint of each warp running on the GPU. We use our framework with 3 task-parallel algorithms and show that we can achieve significant speedups over optimized GPU code.

This work appeared in Hbeika and Kulkarni, LCPC 2016 [3]

3 Domain-aware optimization

3.1 Optimizing Multi-scale Computational Mechanics

An important emerging problem domain in computational science and engineering is the development of multi-scale computational methods for complex problems in mechanics that span multiple spatial and temporal scales. An attractive approach to solving these problems is recursive decomposition: the problem is broken up into a tree of loosely coupled sub-problems which can be solved independently and then coupled back together to obtain the desired solution. However, a particular problem can be solved in myriad ways by coupling the sub-problems together in different tree orders. As we argue in this paper, the space of possible orders is vast, the performance gap between an arbitrary order and the best order is potentially quite large, and the likelihood that a domain scientist can find the best order to solve a problem on a particular machine is vanishingly small.

In this paper, we present a system that uses domain-specific knowledge captured in computational libraries to optimize code written in a conventional language (C). The system generates efficient coupling orders to solve computational mechanics problems using recursive decomposition. Our system adopts the inspector-executor paradigm, where the problem is inspected and a novel heuristic finds an effective implementation based on domain properties evaluated by a cost model. The derived implementation is then executed by a parallel run-time system (Cilk) which achieves optimal parallel performance. We demonstrate that our cost model is highly correlated with actual application runtime, that our proposed technique outperforms non-decomposed and non-multiscale methods. The code generated by the heuristic also outperforms alternate scheduling strategies, as well as over 99% of randomly-generated recursive decompositions sampled from the space of possible solutions.

This work appeared in Liu et al., ICS 2013 [6]

3.2 Optimizing LULESH with CnC

Writing scientific applications for modern multicore machines is a challenging task. There are a myriad of hardware solutions available for many different target applications, each having their own advantages and trade-offs. An attractive approach is Concurrent Collections (CnC), which provides a programming model that separates the concerns of the application expert from the performance expert. CnC uses a data and control flow model paired with philosophies from previous data-flow programming models and tuple-space influences. By following the CnC programming paradigm, the runtime will seamlessly exploit available parallelism regardless of the platform; however, there are limitations to its effectiveness depending on the algorithm. In this paper, we explore ways to optimize the performance of the proxy application, Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH), written using Concurrent Collections. The LULESH algorithm is expressed as a minimally-constrained set of partially-ordered operations with explicit dependencies. However, performance is plagued by scheduling overhead and synchronization costs caused by the fine granularity of computation steps. In LULESH and similar stencil-codes, we show that an algorithmic CnC program can be tuned by coalescing CnC elements through step fusion and tiling to become a well-tuned and scalable application running on multi-core systems. With these

optimizations, we achieve up to 38x speedup over the original implementation with good scalability for up to 48 processor machines.

This work appeared in Liu and Kulkarni, WolfHPC 2015 [7].

3.3 Task and Data Coarsening in CnC

Programmers are faced with many challenges for obtaining performance on machines with increasingly capable, yet increasingly complex hardware. A trend towards task-parallel and asynchronous many-task programming models aim to alleviate the burden of parallel programming on a vast array of current and future platforms. One such model, Concurrent Collections (CnC), provides a programming paradigm that emphasizes the separation of the concernsdomain experts concentrate on their algorithms and correctness, whereas performance experts handle mapping and tuning to a target platform. Deep understanding of parallel constructs and behavior is not necessary to write parallel applications that will run on various multi-threaded and multi-core platforms when using the CnC model. However, performance can vary greatly depending on the granularity of tasks and data declared by the programmer. These program-specific decisions are not part of the CnC tuning capabilities and must be tuned in the program. We analyze the performance behavior based on tuning various elements in each collection for the LULESH application using CnC. We demonstrate the effects of different techniques to modify task and data granularity in CnC collections. Our fully tiled CnC implementation outperforms the OpenMP counterpart by 3× for 48 processors. Finally, we propose guidelines to emulate the techniques used to obtain high performance while improving programmability.

This work appeared in Liu and Kulkarni, LCPC 2016 [8]

4 Application optimization

4.1 Multi-scale Peridynamics

Peridynamics is a nonlocal extension of classical continuum mechanics that is well-suited for solving problems with discontinuities such as cracks. This paper extends the peridynamic formulation to decompose a problem domain into a number of smaller overlapping subdomains and to enable the use of different time steps in different subdomains. This approach allows regions of interest to be isolated and solved at a small time step for increased accuracy while the rest of the problem domain can be solved at a larger time step for greater computational efficiency. Performance of the proposed method in terms of stability, accuracy, and computational cost is examined and several numerical examples are presented to corroborate the findings.

This work appeared in Lindsay et al., CMAME 306:382–405 [5].

4.2 Domain-aware Multilevel Partitioning

Multiscale problems are often solved by decomposing the problem domain into multiple subdomains, solving them independently using different levels of spatial and temporal refinement, and coupling the subdomain solutions back to obtain the global solution. Most commonly, finite elements are used for spatial discretization, and finite difference time stepping is used for time integration. Given a finite element mesh for the global problem domain, the number of possible decompositions into subdomains and the possible choices for associated time steps is exponentially large, and the

computational costs associated with different decompositions can vary by orders of magnitude. The problem of finding an optimal decomposition and the associated time discretization that minimizes computational costs while maintaining accuracy is nontrivial. Existing mesh partitioning tools, such as METIS, overlook the constraints posed by multiscale methods and lead to suboptimal partitions with a high performance penalty. We present a multilevel mesh partitioning approach that exploits domainspecific knowledge of multiscale methods to produce nearly optimal mesh partitions and associated time steps automatically. Results show that for multiscale problems, our approach produces decompositions that outperform those produced by stateoftheart partitioners like METIS and even those that are manually constructed by domain experts.

This work appeared in Jamal et al., IJNME 112(1):58–85 [4]

5 References

[1] Nabeel Al-Saber and Milind Kulkarni. Semcache++: Semantics-aware caching for efficient multi-gpu offloading. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, pages 79–88, New York, NY, USA, 2015. ACM.

[2] Nabeel AlSaber and Milind Kulkarni. Semcache: semantics-aware caching for efficient gpu offloading. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, ICS '13, pages 421–432, New York, NY, USA, 2013. ACM.

[3] Jad Hbeika and Milind Kulkarni. Locality-aware task-parallel execution on gpus. In Chen Ding, John Criswell, and Peng Wu, editors, *Languages and Compilers for Parallel Computing*, pages 250–264, Cham, 2017. Springer International Publishing.

[4] M. Hasan Jamal, Arun Prakash, and Milind Kulkarni. Exploiting semantics of temporal multiscale methods to optimize multilevel mesh partitioning. *International Journal for Numerical Methods in Engineering*, 112(1):58–85.

[5] P. Lindsay, M.L. Parks, and A. Prakash. Enabling fast, stable and accurate peridynamic computations using multi-time-step integration. *Computer Methods in Applied Mechanics and Engineering*, 306:382 – 405, 2016.

[6] Chenyang Liu, Muhammad Hasan Jamal, Milind Kulkarni, Arun Prakash, and Vijay Pai. Exploiting domain knowledge to optimize parallel computational mechanics codes. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, ICS '13, pages 25–36, New York, NY, USA, 2013. ACM.

[7] Chenyang Liu and Milind Kulkarni. Optimizing the lulesh stencil code using concurrent collections. In *Proceedings of the 5th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, WOLFHPC '15, pages 5:1–5:10, New York, NY, USA, 2015. ACM.

[8] Chenyang Liu and Milind Kulkarni. Evaluating performance of task and data coarsening in concurrent collections. In Chen Ding, John Criswell, and Peng Wu, editors, *Languages and Compilers for Parallel Computing*, pages 331–345, Cham, 2017. Springer International Publishing.