SemCache++: Semantics-Aware Caching for Efficient Multi-GPU Offloading

Nabeel Al-Saber, Milind Kulkarni

School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
nalsaber, milind@purdue.edu

Abstract

Offloading computations to multiple GPUs is not an easy task. It requires decomposing data, distributing computations and handling communication manually. GPU libraries have made it easy to offload computations to multiple GPUs by hiding this complexity inside library calls. Such encapsulation prevents the reuse of the data between successive kernel invocations resulting in redundant communication.

In this work, 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 sub-matrices. Experimental results show that our system can eliminate redundant communication and deliver significant performance improvements over multi-GPU libraries like CUBLASXT.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent Programming—Parallel Programming; D.4.2 [*Operating Systems*]: Storage Management—Distributed Memories

Keywords Multi-GPU offloading, GPGPU, Communication optimization

1. Introduction

Graphics processing units (GPUs) offer massive, highly-efficient parallelism, making them an attractive target for computation-intensive applications. Due to the difficulty of programming GPUs, a practical option for leveraging their capabilities is to *offload* computation using libraries. For example, there are many GPU implementations of linear algebra libraries [4, 6–8], which outperform CPU implementations of popular libraries such as BLAS and LAPACK by taking advantage of the GPU's parallel hardware.

This GPU library-based offloading approach has some drawbacks. Notably, moving data back and forth between the CPU and the GPU incurs significant expense. If successive library calls operate on the same data, the data should be moved to the GPU just once, rather than separately for each call, while data should only be transferred back to the CPU if a computation requires it. Such optimization is in tension with the encapsulation objectives of library-based offloading: if a programmer has to manually manage communication between the CPU and GPU, she can no longer port her program to a heterogeneous system without modification.

To help tackle this problem, over the past several years there have been several proposals to introduce automatic memory management between the CPU and single GPU, freeing the programmer from the burden of managing data movement [1, 5]; in fact, the newest version of CUDA [6] offers *Unified Memory (UM)*, which dynamically tracks data movement minimizing communication.

In recent years, *Multi-GPU* systems are becoming increasingly popular. Unfortunately, handling multi-GPU systems is substantially harder than managing a single GPU, as now computation and data need to be distributed across multiple GPUs. To simplify multi-GPU offloading, libraries such as CUBLASXT [6], MAGMA [8], CULA [4] and FLAME [7], completely encapsulate communication in their library calls: prior to invoking a method, data is transferred to the GPU(s), and upon completion, data is transferred back. Such encapsulation introduces significant overheads, as much of this data movement is redundant. However, without encapsulation, managing data movement between kernels is quite difficult in multi-GPU systems.

While there have been several attempts at developing multi-GPU frameworks that can optimize communication more thoroughly, they are not well-suited to developing library replacements. StarPU [2] requires adopting a new programming model. While, StarSs [3] requires annotating every CPU data access, including those outside the offloaded library call. The burden of rewriting an application or annotating large numbers of data accesses makes these models hard to adopt for large applications.

What is needed is an *automatic* approach to managing data movement between the CPU and multiple GPUs that can *dynamically* determine whether data movement is necessary and most importantly provide a drop-in replacements for computational libraries for heterogeneous computing without adopting a new programming model.

In our previous work we proposed SemCache [1], a runtime system to automatically manage and optimize communication between the CPU and a single GPU using caching.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the author/owner(s).

PPoPP'15, February 7–11, 2015, San Francisco, CA, USA
ACM 978-1-4503-3205-7/15/02
http://dx.doi.org/10.1145/2688500.2688527

In this work we propose SemCache++, an extension of SemCache that makes the following contributions:

- The design and implementation of SemCache++, a generic multi-GPU cache that automatically manages communication between CPU and multiple GPUs at variable granularity. SemCache++ enables multi-GPU caching to avoid communication.
- SemCache++ exploits all devices (CPUs and GPUs) in parallel, and uses CUDA streams to allow overlapping of communication and computation.
- A SemCache++-enabled multi-GPU BLAS library that provides a drop-in replacement for existing BLAS libraries.
- Experimental results showing that SemCache++ can dramatically reduce redundant communication, and deliver significant performance improvements over CUBLASXT, NVIDIA's tuned multi-GPU BLAS library.

2. SemCache++

SemCache++ manages communication by tracking the locations of the submatrices, identifying whether it is on the CPU, or on one or more GPUs, or shared between the CPU and GPUs. Data is not eagerly communicated, but instead it is only transferred if it is needed by a computation. Because the data remains distributed after a library call completes, when a future library call is issued, the subtasks of that call can be dispatched to appropriate GPUs to reduce communication.

As in SemCache, SemCache++ determines when a CPU reads or writes data through the use of page protection. Note that while data on the GPUs is tracked at the granularity of decomposed inputs (submatrices), data on the CPU is tracked at the granularity of entire matrices. Thus, SemCache++ uses a two-level directory structure to track data. The first-level entry points to a set of *translation records* for the matrix. When a matrix is decomposed into submatrices and distributed across the GPUs, each submatrix is assigned a record in this second level. A translation record serves several purposes. First, it translates between the location of data on the CPU and the corresponding location on the GPUs, facilitating data movement between devices. Second, it keeps track of the coherence state of the data (*i.e.*, where valid copies of the submatrix reside).

When a task is launched to execute on a GPU, it uses SemCache++ directives to identify which submatrices are needed for the computation. If the data is already being tracked by the first level, SemCache++ checks the status of the required submatrices in the second level. If the data does not exist on the target GPU, communication is performed.

3. Experimental Evaluation

To evaluate SemCache++, we built multi-GPU implementation of BLAS library interfaces provided by CUBLAS (NVIDIA's single-GPU linear algebra libraries) using SemCache++ directives. Our experiments were performed on a server with AMD Opteron Processors and 32GB memory connected via PCIe 2.0 to two NVIDIA Kepler K20 GPUs.

We used a simple microbenchmark that performs two matrix multiplies and a DAXPY: $D = AB + AC$. Note that the two matrix multiplies share one of their operands (A), and the DAXPY operates on the results. As a baseline, we used CUDA 6's unified memory along with CUBLAS to implement a communication-optimized single-GPU version of the microbenchmark. We compared this baseline to SemCache++, CUBLASXT and StarPU using one and two GPUs. Unlike SemCache++ and CUBLASXT, StarPU implementation requires rewriting the benchmark using their programming model. CUBLASXT supports multi-GPU computation by

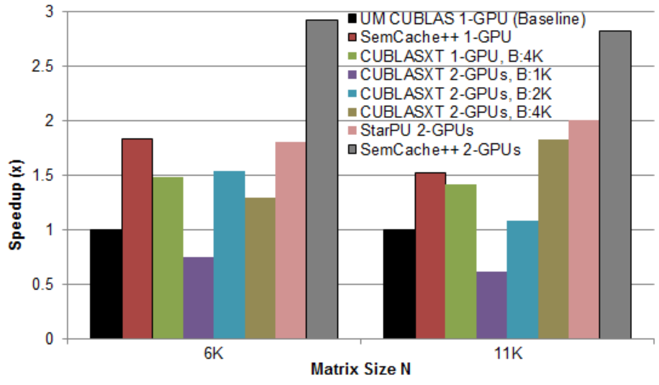

Figure 1. Speedup of microbenchmark for different matrix sizes, normalized to UM CUBLAS 1-GPU)

overlapping communication with computation. Its performance is dependent on setting the block size for this pipelined schedule. Hence, we evaluated different block sizes for CUBLASXT.

Figure 1 shows the results of the microbenchmark experiment, looking at two different matrix sizes. We see that even on a single GPU, both SemCache++ and CUBLASXT are faster than the baseline because they overlap communication with computation. SemCache++ is faster than CUBLASXT because it is able to minimize communication. The A matrix is cached on both GPUs, as are the results of the DGEMMs. Hence, the DAXPY can be performed with no additional communication. In contrast, CUBLASXT, which does not leave the DGEMM results on the GPUs, must communicate the results of the DGEMMs *back* to the GPUs to perform the DAXPY.

When scaling to two GPUs, we find that SemCache++'s advantage increases: it is nearly $3\times$ faster than the baseline, and 30-50% faster than CUBLASXT and StarPU. StarPU is slightly faster than CUBLASXT because it avoids redundant communication. However, synchronization was used to produce correct results which made it slower than SemCache++.

References

[1] N. AlSaber and M. Kulkarni. Semcache: Semantics-aware caching for efficient gpu offloading. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, 2013.

[2] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. In *Euro-Par 2009 Parallel Processing*. 2009.

[3] E. Ayguadé, R. M. Badia, F. D. Igual, J. Labarta, R. Mayo, and E. S. Quintana-Ortí. An extension of the starss programming model for platforms with multiple gpus. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, 2009.

[4] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis. Cula: hybrid gpu accelerated linear algebra routines. *SPIE Defense and Security Symposium (DSS)*.

[5] T. B. Jablin, J. A. Jablin, P. Prabhu, F. Liu, and D. I. August. Dynamically managed data for cpu-gpu architectures. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, CGO '12, 2012.

[6] NVIDIA. Cuda. `http://developer.nvidia.com/cuda-toolkit`.

[7] G. Quintana-Ortí, F. D. Igual, E. S. Quintana-Ortí, and R. A. van de Geijn. Solving dense linear systems on platforms with multiple hardware accelerators. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '09, 2009.

[8] P. D. S. Tomov, R. Nath and J. Dongarra. Magma version 0.2 user guide.