Using GPU's to Accelerate Stencil-based Computation Kernels for the Development of Large Scale Scientific Applications on Heterogeneous Systems

Jian Tao
Center for Computation & Technology, Louisiana State University, Baton Rouge, Louisiana, USA
jtao@cct.lsu.edu

Marek Blazewicz
Applications Department, Poznań Supercomputing and Networking Center, Poznań, Poland
marqs@man.poznan.pl

Steven R. Brandt
Center for Computation & Technology, Louisiana State University, Baton Rouge, Louisiana, USA
sbrandt@cct.lsu.edu

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel programming; D.3.3 [*Language Constructs and Features*]: Frameworks

General Terms Algorithms, Design, Languages

Keywords GPGPU Programming, Computational Framework, HPC, Stencil Computation

Abstract

We present CaCUDA - a GPGPU kernel abstraction and a parallel programming framework for developing highly efficient large scale scientific applications using stencil computations on hybrid CPU/GPU architectures. CaCUDA is built upon the Cactus computational toolkit, an open source problem solving environment designed for scientists and engineers. Due to the flexibility and extensibility of the Cactus toolkit, the addition of a GPGPU programming framework required no changes to the Cactus infrastructure, guaranteeing that existing features and modules will continue to work without modification. CaCUDA was tested and benchmarked using a 3D CFD code based on a finite difference discretization of Navier-Stokes equations.

1. Introduction

Heterogeneous systems are becoming more common in the field of High Performance Computing (HPC). Three out of five of the fastest computers in the world use GPGPUs to achieve their performance[5], and more than 34 of the top 500 systems are GPU-based. However, even using tools like CUDA and OpenCL it is a non-trivial task to obtain optimal performance on the GPU, and it is even more difficult to achieve sustained performance at scale on hybrid supercomputers. The CaCUDA programming framework leverages the highly scalable Cactus framework [3], making use of its component infrastructure and parallel programming abstractions to design and implement a tool for creating stencil-based computation kernels. By using automatic code generation from a set of highly optimized code templates, CaCUDA frees scientific application developers not only from lower level programming issues such as parameter parsing and I/O, but more importantly, from the parallelization and optimization details of GPGPU programming. Our design assigns one GPU to one MPI process and is able to benefit from the Cactus grid abstractions without requiring any changes to the distributed grid structure, grid geometry, and inter-process communication. CaCUDA extends Cactus by adding a code generation system that automates the management of storage on the GPU, synchronization among threads, communication between CPU and GPU, and optimization on GPU. Everything except the kernel stencil computation itself can be handled in the CaCUDA programming framework automatically via a kernel descriptor and a code generator.

PPoPP'12, February 25–29, 2012, New Orleans, Louisiana, USA.
ACM 978-1-4503-1160-1/12/02.

2. Cactus Computational Framework

Cactus [3] is a problem-solving platform that was designed and implemented by an international team of computational scientists led by Seidel, Suen et al., to free numerical relativists from lower level parallel programming as well as hardware concerns [4]. After years of development, Cactus has evolved into a generic, open-source framework for developing large scale parallel scientific applications based on structured meshes. Currently, there are at least 30 worldwide research groups using Cactus. The name *Cactus* is also a metaphor for its design. A Cactus application consists of a core piece of infrastructure called the *flesh* and user modules called the *thorns*. The flesh provides a framework for defining and parsing parameters, for scheduling work, interoperation between C, C++, F77, and F90, as well as interaction with other thorns. Thorns are described using a domain specific language (DSL) called the *Cactus Configuration Language (CCL)* [1]. The information in the CCL files includes the name of the implementation, the definition of functions and parameters, the schedule of the routines, and whether they require synchronization after execution, etc.

3. CaCUDA Programming Framework

In order to facilitate programming in the CaCUDA environment, we defined the term *Kernel Abstraction*, which consists of three major components: *Kernel Descriptor*, *Computation Templates*, and *Code Generator*. The definition of a kernel may be divided into three separate tasks:

- *Declaring the Kernel:* The *cacuda.ccl* configuration file is used to describe the data dependencies, namely the *grid functions* and *parameters* required by the kernel and stencil. This decla-

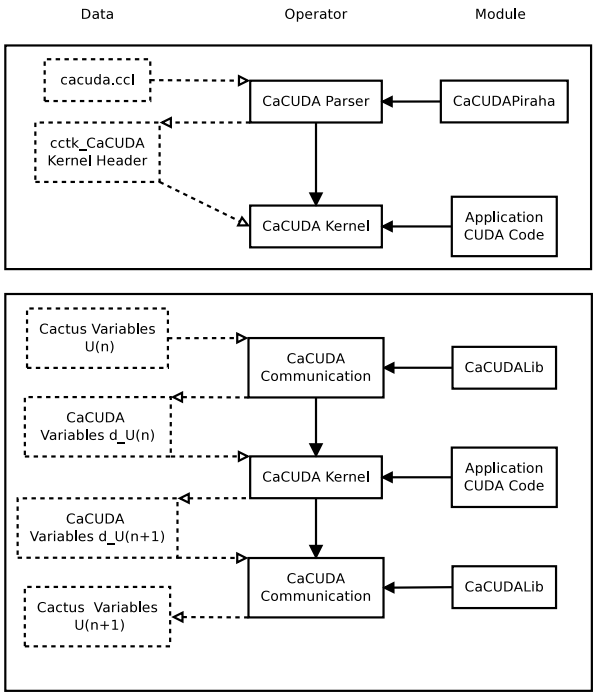

Figure 1. The workflow of a CaCUDA-based application: The upper box shows the generation of the CaCUDA kernel headers at the code compilation stage, while the lower box shows how the variables are evolved to the next time step.

ration file is then used to generate a kernel frame (macros) that performs automatic data fetching, caching and synchronization with the host.

- *Writing the Kernel:* Using kernel-specific auto-generated macros the programmer writes a set of stencil equations for one grid point only, accessing neighboring grid points by specifying the relative index in each direction. The code must be written to avoid *read-after-write* and *write-after-read* hazards, and thus requires some basic knowledge of parallel programming.
- *Scheduling the Kernel:* Insertion of the newly generated kernel into the Cactus schedule tree.

The CaCUDA kernel abstraction makes it easy to write and execute GPGPU kernels, and makes it possible to optimize the kernel without changing the kernel code itself. The whole optimization process is handled by swapping the templates or adjusting the kernel parameters. Furthermore, our system is not limited to the GPGPU architecture. The templates could be easily adapted to run as sequential CPU or parallel OpenMP code. The CaCUDA kernel code can be integrated in a straightforward manner within existing thorns (modules) without touching the flesh (core infrastructure). The workflow of the CaCUDA compilation process as well as the kernel execution is shown in Figure 1.

4. Sample Application

In this work we've focused on a simple test case in Computational Fluid Dynamics (CFD), namely lid-driven cavity with computations performed in single precision. Additional details regarding the testing of the application can be found in our previous work [2]. We adopted the computational patterns proven to be most efficient in stencil computations. These patterns were further generalized to fit wider variety of numerical problems. Our test application achieved

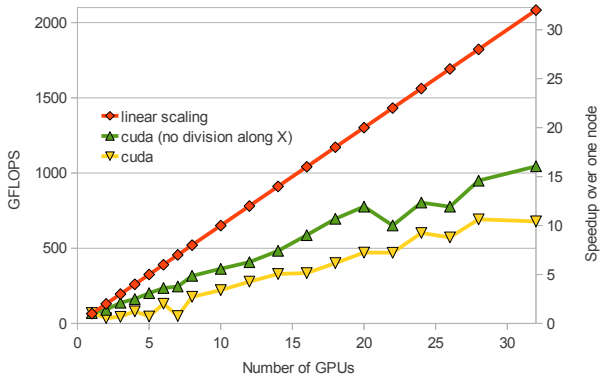

Figure 2. The graph presents the weak scaling test and shows the performance (in GFLOPS) and the speedup over a single GPU. The domain size on each GPU is 192^3. Two cases were considered: with and without domain decomposition along the X direction.

70GFLOPS on single NVIDIA Fermi GPU. However, due to communication overhead, the performance of computations conducted on 2 nodes is comparable to this on single node. The weak scaling test is shown in Figure 2.

5. Conclusions

We presented our work to design and implement a GPGPU kernel abstraction, which is suitable for developing highly efficient large scale scientific applications using stencil computations on hybrid CPU/GPU systems. By leveraging the MPI-based data parallelism implemented in Cactus, we have developed a tool which enables both MPI and GPU acceleration. The lid-driven cavity problem was implemented and benchmarked with CaCUDA, and the results presented. Our current efforts are focused on minimizing the costs of the data exchange between GPU and CPU and optimizing the boundary exchange.

6. Acknowledgments

This work was performed using the computational resources of LSU/LONI and was supported by the Center for Computation and Technology at LSU. This work was also supported by the UCoMS project under award number MNiSW(Polish Ministry of Science and Higher Education) Nr 469 1 N - USA/2009 in close collaboration with U.S. research institutions involved in the U.S. Department of Energy (DOE) funded grant under award number DE-FG02-04ER46136 and the Board of Regents, State of Louisiana, under contract no. DOE/LEQSF(2004-07).

References

[1] G. Allen, T. Goodale, F. Löffler, D. Rideout, E. Schnetter, and E. L. Seidel. Component Specification in the Cactus Framework: The Cactus Configuration Language. In *Grid2010: Proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, 2010. (arXiv:1009.1341).

[2] M. Blazewicz, S. R. Brandt, P. Diener, D. M. Koppelman, K. Kurowski, F. Lffler, E. Schnetter, and J. Tao. A massive data parallel computational framework on petascale/exascale hybrid computer systems, submitted. In *International Conference on Parallel Computing*, Ghent, Belgium, 2011.

[3] Cactus. URL http://www.cactuscode.org/.

[4] E. Seidel and W.-M. Suen. Numerical relativity as a tool for computational astrophysics. *J. Comp. Appl. Math.*, 109:493, 1999.

[5] Top 500. URL http://www.top500.org/. Top 500 Supercomputer Sites.