GPUs and Einstein's Equations

Timothy Dewey
tdewey@math.umd.edu
Advisor: Dr. Manuel Tiglio
tiglio@umd.edu
Physics Department, CSCAMM*

October 12, 2010

Abstract

This document outlines the project proposed for the 2010-2011 AMSC 663/664 course series. The project is to write code that can be used to solve Einstein's equations using graphics processing units. We provide a brief introduction to the project and then provide an outline of the proposed implementation and validation. Furthermore, a timeline is provided for major milestones in the project.

1 Problem Introduction

The aim of this project is to write code to solve Einstein's equations using spectral methods. This implementation should be optimized so we will be able to give meaningful performance data that can, in turn, be used to make a determination about the utility of using graphics processing units (GPUs) for solving large scale problems. There seems to be a good evidence that we can make significant time and power improvements by solving Einstien's equations on GPUs instead of CPUs.

2 Approach/Implementation

Einstein's equations are a system of hyperbolic PDE's which are first order in space and time. In order to solve these equations, we will use a pseudo-spectral collocation method (we will just say spectral method from now on). The code will be implemented first in MATLAB, then in

*Center for Scientific Computation and Mathematical Modeling

C/C++, and finally in Compute Unified Device Architecture (CUDA) for GPUs. This code must be general enough to handle the complicated system of Einstein's equations.

In order to realize exponential convergence with the spectral method, we will use the collocation points as the roots of orthogonal polynomials (e.g., Legendre or Chebyshev). We will use the Chebyshev polynomials with the Gauss-Lobatto points (see [1]). These points will determine a differentiation matrix $\mathbf{D}$ that will allow us to approximate the derivative of a function $u(x)$ at the collocation points $\mathbf{x} = \{x_0, \ldots, x_{N-1}\}$ by simply performing a matrix vector multiplication $\mathbf{D}u(\mathbf{x})$. This computation is fast and requires little memory for a small number of collocation points N. Typically, we have $N < 30$. The main cost in solving Eistein's equations is computing the right hand side (RHS) of the equations. This computation is very complex, and it will need to be optimized for a GPU.

2.1 Computing Resources and Tools

This project will require a use of sophisticated computational tools. It will, of course, require GPU computing which is an increasingly important skill in scientific computing. The code will be run on a small GPU cluster. Also, we will using the profiling tools such as gprof and Nsight. For full optimization, we will make use of basic linear algebra subprograms (BLAS). The final code should be efficient, stable, and readable. In addition to using various debugging tools, the code will be clearly documented.

This project will require a mature understanding of both the numerical methods and the computational techniques used to solve large scale PDEs. Completing this project should clearly demonstrate the ability to perform at a high level in scientific computing.

2.2 Challenges

A major challenge in the project will be to manage the expensive memory accesses for the GPU. It is important to keep the GPU busy with the data it needs since computations are cheap relative to memory accesses. We will use spectral methods to reduce the memory required to solve these equations. The other major challenge in the GPU implementation will be communication between the host processor and the GPU. If communication between the CPU host and GPU is needed at every time step, then the overall computation could easily be dominated by this communication. For this reason, we aim to implement all the code required to step the system of PDEs an arbitrary number of steps forward in time on one GPU. If we are successful in completing

this task, then the communication required between the host CPU and GPU will be minimal.

Optimization is a nontrivial task and this will be a large portion of the project. Profiling results from the CPU implementation will suggest the portions of code that have the greatest potential for speedup in the GPU implementation. We will port these portions of the code to CUDA first in order to test the time spent running this code in the GPU. These data points will be useful in the overall optimization of the code. The results of the profiling and optimization will be presented in the final project report.

3 Databases/Validation

In order to validate the implementation, we will rely on computing the solutions to problems with known answers. An analytic solution to Einstein's equations for a spherically symmetric black hole is known (see [2]), and it can serve as a known solution for 1, 2, and 3 dimensions. This analytic solution will be the known solution to which we compare our computed solution at each stage of the process. Furthermore, there will be certain functions in the implementation that can be validated independently. For example, a function that computes a differentiation matrix can be validated on functions with known analytic derivatives. A series of test functions will be written to validate portions of the code throughout the implementation.

4 Testing

A series of tests will be performed to show the performance of the code on the CPU and the GPU. In order to compare the performance of the CPU and GPU, one should consider time, energy, development cost, system cost, and perhaps some other variables. These considerations are beyond the scope of this project, but we should get at least some insight into whether or not GPUs are a viable means to save time and money on the computations need to simulate black holes for the LIGO (Laser Interferometer Gravitational-Wave Observatory) project and other similar projects. Furthermore, we will generate performance data that can be used to calculate the cost of a simulation given a cost function.

5 Project Schedule/Milestones

The major milestones of the project will be the MATLAB, C, and CUDA versions of the spectral method solver for Einstein's equations. In

addition to these 3 languages, we hope to have time to modify the code to handle 2 and 3 dimensions. If the 1-d case is handled properly, then adding more dimensions should be much less work than implementing and testing the 1-d case. The bulk of the work for this project should be spent properly architecting, testing, and optimizing the 1-d case.

In addition to the implementations, there will be informal written plans for how the C and CUDA code should be implemented. This will include data structures and functions. These informal plans should be forward-thinking so the code will be easily modified to handle higher dimensions.

Timeline:
December 1, 1-d Matlab code verified on test problem.
December 1, Written plan for C code.
February 10, 1-d C code verified on test data.
February 15, Written plan for CUDA.
March 15, 1-d CUDA code verified on test data.
April 15, Optimized 1-d CUDA code.
May 1, Complete writeup of results.
May, Time permitting, 2-d and/or 3-d version.

6 Deliverables

• MATLAB, C, and CUDA code to solve Einstein's equations using spectral methods.
• Complete code documentation.
• Write-up explaining results of code profiling and performance testing.

References

[1] L. Trefethen. "Spectral Methods in MATLAB." SIAM, 2000.

[2] G. Calabrese, L. Lehner, M. Tiglio. "Constraint-preserving boundary conditions in numerical relativity." arXiv:gr-qc/0111003v1. 2 November 2001.