BarraCUDA Project: A Homebrew Supercomputer

Matt Redmond

1 February 2009

Contents

1 Abstract

The BarraCUDA (Beginner's Analytic Research and Rendering Architecture over C Unified Device Architecture) project will ultimately entail a small-scale homebrew supercomputer designed around the NVIDIA CUDA architecture, as well as a set of simulation programs coded to take advantage of this CUDA architecture. The project is meant to serve as an introductory platform for student-researchers to develop parallel-programming skills, and design programs that showcase concepts in modern science research through real-time visualization. The initial seed for this projected was planted by Dr. Kayvon Fatahalian, a graphics researcher at Stanford University. The hardware aspect of the BarraCUDA project will be completed over a different time frame, but before the hardware is finished, I intend to develop and (on a less powerful computer) test a comprehensive and accurate three dimensional simulation of electrodynamics. This project will explore the vast world of scientific computing through the lens of a physicist, and should provide ample opportunities for me to learn new techniques in computational physics as well as numerical analysis.

2 Introduction

Through my Applied Science Research and Computer Science classes, I've gradually become aware of the growing importance of high performance computational power in the fields of physical, chemical, and biological research. A small sample of these problems can be found at http://www.nvidia.com/object/cudahome.html, but three particularly interesting ones (that seems to form a representative sample of the types of problems that one can approach with supercomputing resources) are [1], [2], and [3]. Because of my experiences over the summer, I've become quite interested in learning more about neutron scattering and visual representations of physical modeling problems. Last summer, I got a chance to work in Caltech's materials physics lab on the DANSE project, using distributed computational analysis to model the theoretical results of neutron scattering experiments before actually conducting them. It was here where I was exposed to the elegant concept and power of a Beowulf cluster, and I immediately realized that our high school could make good use of such a powerful tool (on a smaller scale) for instructional purposes, in the demonstration/visualization of scientific concepts as well as the programming process.

I have begun to design the simulation on a CUDA-based supercomputer because CUDA represents the new direction that high performance computing is moving in: instead of waiting for a single fast central processing unit to perform heavy calculations sequentially, these calculations are farmed out in parallel to hundreds of individual graphics processing unit cores for computation. The actual performance increase (across diverse applications) is reported by various sources [4] to be anything from 5x to 2600x faster than a CPU-based system. Furthermore, the CUDA architecture is alive and well-supported by the community, with various development tools available, and many new applications released frequently. CUDA supports the API OpenMP, a well-known parallel processing API and widely supported in the scientific computing community. I'd like to gain further experience (extending my summer research) working with OpenMP on hardware that can unlock all of its power. CUDA is supported on several chipsets, but the two most suitable consumer-grade architectures for this project are the NVIDIA GeForce GTX295 and the NVIDIA Tesla C1060.

Completing this project will not only provide me with a working knowledge in supercomputer design and construction, but also an opportunity to learn the parallel programming skills that I'll need to be a successful physics researcher in the future. The bottom line here is that supercomputing as a field continues

to grow very rapidly, and most pure sciences are relying upon supercomputing to provide a basis for carrying out their research. It's particularly interesting to explore the underpinnings of graphics processor based supercomputing, because the onset of GPU computing provides a brand new avenue for massive scalability that simply isn't present in the CPU-based paradigm.

3 Theory

This section will be broken down into the computer science theory behind how the project works, and the physical science theory behind how systems of electrically charged particles interact. Currently (as of January 2010), the simulation is not tuned to run on the CUDA architecture.

3.1 Supercomputing Theory

A GPU-based supercomputer differs from standard CPU-based supercomputers in a number of ways: the most obvious one of these is the actual location a nd type of the numerical processors. A CPU-based cluster (I will interchangeably refer to supercomputers and cluster computers in this paper; they are equivalent for our purposes) uses processors housed directly on the node motherboard, which communicate directly with the system memory through a high-speed bus. This has the advantage of extremely fast memory read/write operations, but it has the disadvantage (currently) of a relatively small number of available cores. The current top-of-the-line Intel i7-920 runs 8 cores at 1600 MHz each [5]. The GPU, in comparison, resides (generally) in an interface bus. In modern computers, this bus is the PCI express bus, which provides an indirect way of communicating with the memory through a controller chip on the motherboard. The advantage of the GPU-based system is that it can run many more cores than a CPU-based system can (480 cores per card in the BarraCUDA system at 1242 MHz each) [6], but the disadvantage is that it has relatively slower memory read/write times. Most GPUs attempt to correct for this by including memory onboard the card (1796 MB in the BarraCUDA system per card), so the read/write times are reduced to those of a CPU-based system. As one can see, there are significant advantages to using a GPU-based setup.

The main advantage (drastically more cores) of the GPU-base system is only harnessable when a program is designed to UTILIZE all of those cores in parallel. Certain programs lend themselves well to this paradigm; others do not. An example of an easily parallelizable problem is one that can be broken down into a number of sub-tasks, each of which can be computed independently of the other tasks. The n-body electrodynamics problem is a perfect example of an easily parallelizable problem: computing the force on each particle at a given timestep only relies on the previous position of all of the other particles. If we store the point charge positions in an array that is globally accessible by all of the cores, then we can update each particle SIMULTANEOUSLY. This is a HUGE speedup over the sequential calculations enacted by a single-core CPU setup, and provides the motivation for a multiple-core setup in general.

3.2 Scientific Theory

The simulation that is currently in place relies upon Coulomb's Law to generate the electric field acting on each individual point charge. This efield generates a force that is similar to the gravitional force in that it is an inverse-square law, but different because the force vector is signed (instead of always attracting like

gravity, it can attract or repel). The force vector acting on a point charge is represented by $\vec{F} = q * \vec{E}$, where q is the charge of the particle and $\vec{E}$ is the electric field acting on the particle. The explicit way to compute $\vec{E}$ is to sum up the effects of each other particle in the simulation on the particle in question.

Couloumb's Law gives this sum as

$$\vec{E} = \frac{1}{4\pi\epsilon_0} \sum_{i=1}^{n} \frac{q_i(\vec{r} - \vec{r_i})}{|\vec{r} - \vec{r_i}|^3}$$

where $\vec{r}$ is the vector to the point that we are testing the electric field at, and $\vec{r_i}$ is the vector of the point we are testing against. As i ranges from 1 to n, we test the electric field against all other particles. This means that updating ONE particle's electric field requires us to check $n - 1$ other particle's states, where n is the total number of particles. Updating ALL of the particle's electric fields requires $n(n - 1)$ checks, making the n-body electrodynamic simulation a problem known as an $O(n^2)$ problem (it has time complexity asymptotically equal to some constant multiple of n^2). These problems scale notoriously badly on standard PC hardware, but with a supercomputer, they become approachable. There are ways to reduce this (by only considering particles within a certain distance of eachother to have an observable effect) which can reduce the runtime complexity, but these methods trade off a gain in speed with a reduction in accuracy. I have decided that accuracy is a more desirable characterstic of my simulation than speed is, so I check all $n(n-1)$ particles.

4 Accomplishments

5 The Next Steps

6 Citations

[1] CUDA-Based Incompressible Navier-Stokes Solver. Julien Thibault and Inanc Senocak.

<http://coen.boisestate.edu/senocak/files/BSU_CUDA_Res_v5.pdf>

[2] TeraFlop Computing on a Desktop PC with GPUs for 3D CDF. J. Tolke and M. Krafczyk.

<http://www.irmb.bau.tu-bs.de/UPLOADS/toelke/Publication/toelkeD3Q13.pdf>

[3] CUDA Acceleration of Molecular Dynamics. David Kirk and Wen-mei W. Hwu.

<http://www.ks.uiuc.edu/Research/gpu/files/lecture8casestudies.pdf>

[4] NVIDIA CUDA Project Repository. <http://www.nvidia.com/object/cuda_home.html#>

[5] Processor Specs

[6] More Specs

A Source Code