Numerical Modeling of Extreme Mass-Ratio Inspirals Accelerated by OpenCL

Justin McKennon
Undergraduate Electrical Engineering Major
University of Massachusetts Dartmouth

Research done in collaboration with
and under the supervision of Dr. Gaurav Khanna
Physics Department, University of Massachusetts Dartmouth

A special thanks to the NSF,
the UMass Dartmouth Math Department,
and NASA for their contributions, guidance and funding assistance on this project.

May 5, 2010

1 Abstract

Justin McKennon

Since the early 2000's, scientists, companies and researchers alike have all become dependent on computers. In the past, to increase the speed and performance of a processor, designers would simply add more transistors to accomplish this. Recently however, the computer industry hit a "frequency wall." Modern processor speeds have topped out at roughly 3 GHz due to the massive increase in power consumption and heat dissipation that comes with the adding of more and more transistors to a design. This has forced the computer industry to pursue what is called a "many-core" design. By packing several (sometimes up to eight(8)) processing cores on a single computer, one can effectively increase the performance and speed of it. This technique is widely used by the computer industry, but is quickly reaching the end of its feasibility. Adding more and more processors takes up more and more space, dissipates more and more heat, and uses more and more power. These factors play a tremendous role in how many cores on can effectively use in a single computer. Despite all of these limitations, science, math and technology continue to become more advanced and demand increased performance from computers and the computer industry.

2 Need for Parallelization

In the past few years, the idea of parallelization has been brought to the forefront of the computer industry. In theory, the idea is simple; do as many things at once as possible. Much difficulty has been found in the implementation of parallelization, however.

3 Realization of the Power of GPUs

Sitting right under the nose of computer scientists and designers throughout this whole process has been what could be the key to unlocking the secret of parallelization.

For several years, graphics processing units (GPUs) were solely used for gaming. GPUs are capable of performing hundreds of thousands of calculations per second, and run in a very efficient, parallelized style. When someone is playing an computer game, all in game movements, actions, physical effects (smoke, explosions etc) are all calculated, processed, and spat out in image form by the GPU seamlessly. Current high end GPUs contain hundreds of cores, making them ideal for parallel processing. This, leaves the question: how can one harness the power of these devices?

4 High Performance

Currently, there are several vendor and design specific programming languages that allow developers to use the GPU for non-gaming purposes. Languages such as CUDA, ATI Stream, and the Cell SDK all allow for the GPU to be programmed just as if it was the CPU.

Proper utilization of these programming languages allows for the user to achieve significant performance gains, often well over an order of magnitude.

5 Shortcomings

One of the main reasons why GPU programming isn't widely used is due to its difficulty. GPU programming languages are usually vendor and design specific. If a computer has an NVidia graphics card, CUDA must be used to program it. ATI graphics cards and the Cell Broadband Engine cannot be programmed with this language, severely inhibiting any sort of portability between different architectures.

Another major problem with current GPU programming languages is the lack of commonality between them. In standard CPU programming, most people who can program in C can also program in C++, Csharp etc due to the many similarities between the languages. With GPU programming, if one person can program in one language, in order to use a different type of architecture/device, the user would have to start from square one and learn all the nuances and techniques that are required to use the new language.

The biggest hurdle in GPU programming, is the requirement of advanced hardware knowledge. To be able to use any of the aforementioned GPU languages, the user must know their GPU inside and out. Memory management is far from straightforward and improper allocation of resources has a serious impact on any sort of performance that is to be gained from using the GPU. Debugging is also very difficult due to the lack of libraries that are able to be called by the GPU.

6 A New Frontier

6.1 Development

With the release of Mac OsX 10, aka "Snow Leopard," the boundaries of GPU and parallel computing were pushed considerably. In a collaborative effort between Apple and the Khronos group, OpenCL was developed and released. The idea behind OpenCL (Open Computing Language), was to develop a method for exploiting and utilizing the power of the GPU in a language that is hardware and device independent. As previously mentioned, GPU programming languages prior to the release of OpenCL had a severe issue with portability between architectures.Now, with OpenCL available, programming and NVidia GPU, an ATI GPU or any other GPU for that matter is the same. OpenCL takes care of the dirty work, and the only thing left to be done when importing to a new device is to ensure that proper memory allocation techniques are being used. This technology of architecture portability is so powerful that all major processor vendors (NVidia, ATI, etc) have adopted the standard.

6.2 Advantages

With its' inception, OpenCL will undoubtedly save thousands of hours of programming for those in the GPU field. Since OpenCL is derived from the language C, making it very easy for most modern programmers to pick up. In addition, one of the biggest pluses about OpenCL is how easy the memory management is. By simply reading the user manual that comes with the purchase of a GPU one will know more than enough to guarantee efficient running of an OpenCL program on their device from a memory allocation standpoint.

7 Numerical Relativity

Numerical Relativity is the study of strong sources of gravitational waves. These gravitational waves were predicted by Einstein's Theory of Relativity but have never been directly observed, due to technological deficiencies (until now).

7.1 Our Application

Our application deals with the evolution of gravitational waves generated by a compact object in a decaying orbit around a supermassive black hole (SMBH). These SMBH are often over 1,000,000 larger than our own sun, and often reside at the center of a galaxy, routinely devouring planets and stars. Due to the extreme difference between the mass of the object spiraling in to the black hole and the black hole itself, this decay can be referred to as an EMRI (Extreme Mass-Ratio Inspiral). Due to this extreme mass ratio, the objecting undergoing the EMRI can be referred to as a structureless object allowing this problem to be addressed under black hole perturbation theory.

7.2 Teukolsky Equation

$$
\begin{aligned}
&-\left[\frac{(r^2+a^2)^2}{\Delta}-a^2\sin^2\theta\right]\partial_{tt}\Psi-\frac{4Mar}{\Delta}\partial_{t\phi}\Psi\\
&-2s\left[r-\frac{M(r^2-a^2)}{\Delta}+ia\cos\theta\right]\partial_t\Psi\\
&+\Delta^{-s}\partial_r\left(\Delta^{s+1}\partial_r\Psi\right)+\frac{1}{\sin\theta}\partial_\theta\left(\sin\theta\partial_\theta\Psi\right)+\\
&\left[\frac{1}{\sin^2\theta}-\frac{a^2}{\Delta}\right]\partial_{\phi\phi}\Psi+2s\left[\frac{a(r-M)}{\Delta}+\frac{i\cos\theta}{\sin^2\theta}\right]\partial_\phi\Psi\\
&-\left(s^2\cot^2\theta-s\right)\Psi=-4\pi(r^2+a^2\cos^2\theta)T,
\end{aligned}
$$

Figure 1: Teukolsky Equation

Our application deals with the solving for the term T in the above Figure 1. This item is known as the source term, and is much too long to reproduce here. It is sufficient to say that it would be near impossible to solve for this term by hand let alone perform the iterations needed to get any relevant information since the term is time dependent. Even for a CPU this task is not easy. By utilizing OpenCL however, we were able to greatly shorten the amount of time taken to calculate the term.

7.3 Results

7.3.1 CBE

The first chart, Figure 2,shows the results of running the calculation in OpenCL on the Cell Broadband Engine. OpenCL(source) is with the parallelization kernel not precompiled (left to be compiled when it is called in the program), and OpenCL binary is with the kernel precompiled.

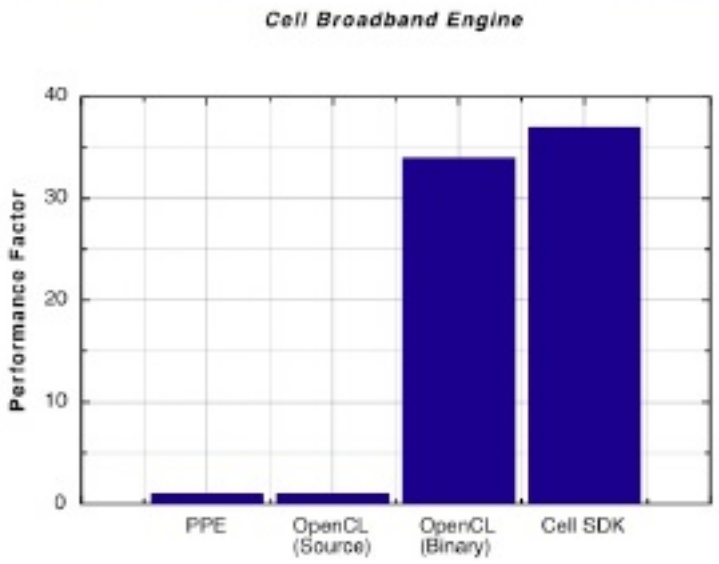

Figure 2: Baseline is the PPE on the Cell

As shown, OpenCL performed admirably against the calculation being done in the Cell Broadband Engine's native language. Both implementations yielded impressive results, over a 20x performance gain.

7.3.2 Tesla

In Figure 3, the same code was run on an NVidia Tesla C1060 GPU. The baseline CPU is an AMD quad core 2.5 Ghz processor, a very powerful computer.

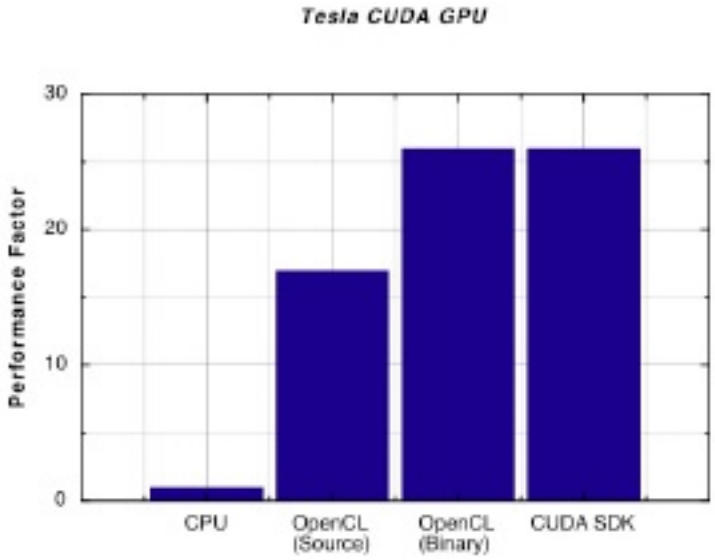

Figure 3: Nvidia Tesla GPU Results

As illustrated in figure 3, OpenCL performed identically to CUDA on the NVidia Tesla GPU. This result is especially remarkable because of OpenCL's short lifespan. CUDA has been around for over 5 years and OpenCL has yet to reach its' first birthday. To be so new and produce such powerful results is no easy matter. This is an incredible example of OpenCL's power and potential. With results like this, future parallelizations will almost certainly be done in OpenCL.

7.3.3 Relative Performance

In figure 4, a shining example of OpenCL's portability is shown.

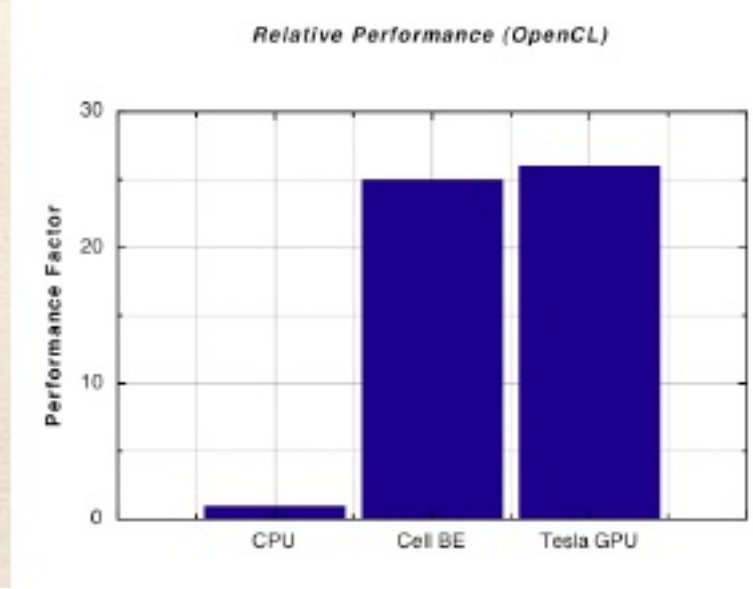

Figure 4: Comparative Results on Different Architectures

Regardless of which architecture the calculation was done on, OpenCL performed exceptionally. Despite the vastly different architecture layouts and powerful competition, OpenCL was able to produce remarkable results when the parallelization kernel was compiled before the program was run.

8 Conclusion

Regardless of the environment, OpenCL provided excellent performance gains over traditional serial computing on the CPU. With further updates and releases, OpenCL is almost surely going to surpass the performance of CUDA and the Cell SDK entirely. As the language becomes more and more refined, the results will certainly become more and more impressive. This trend helps fuel the belief that in the future, the majority of processing will be done in this fashion with a language similar to or with OpenCL.

9 Acknowledgements

Dr. Khanna would like to thank Glenn Volkema for his assistance, insight and work throughout this project. He also would like to acknowledge research support from the NSF, IBM, Sony and Nvidia. I would like to thank the University of Massachusetts Math Dept, the professors of the CSUMS program, the NSF, and Massachusetts Space Grant Consortium for their help and assistance throughout this project.

10 References

Websites: [1] IBMs Cell site: http://www.research.ibm.com/cell [2] Sony PS3: http://www.us.playstation.c
[3] RoadRunner: http://www.lanl.gov/roadrunner/ [4] Nvidias CUDA: http://www.nvidia.com/cuda/
[5] NSFs LIGO: http://www.ligo.caltech.edu/ and several others (GEO, Virgo, TAMA, AIGO). [6] ESA/NASA LISA: http://lisa.nasa.gov/ [7] Cell Broadband Engine Architecture and its first implementation a performance view: http://www.ibm.com/developerworks/power/library/pac
[8] OpenCL: http://www.khronos.org/opencl/

Papers: [9] L. Burko, G. Khanna, Accurate time-domain gravitational waveforms for extreme-mass-ratio binaries, Europhysics Letters 78, 2007, 60005; P. Sundararajan, G. Khanna, S. Hughes, Towards adiabatic waveforms for inspiral into Kerr black holes: I. A new model of the source for the time domain perturbation equation, Phys. Rev. D 76, 2007, 104005; P. Sundararajan, G. Khanna, S. Hughes, S. Drasco, Towards adiabatic waveforms for inspiral into Kerr black holes: II. Dynamical sources and generic orbits, Phys. Rev. D 78, 2008, 024022. [10] V. Agarwal, L.-K. Liu, D. Bader, Financial modelling on the cell broadband engine, IEEE International Parallel and Distributed Processing Symposium, Miami, Florida, 2008. [11] M. Christen et al., Accelerating Stencil-Based Computations by Increased Temporal Locality on Modern Multi- and Many-Core Architectures, New Frontiers in High-performance and Hardware-aware Computing (HipHaC), Lake Como, Italy, 2008. [12] Scherl et al., Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture, Nuclear Science Symposium / Medical Imaging Conference, Honolulu, Hawaii, 2007. [13] J. Breitbart, G. Khanna, An exploration of CUDA and CBEA for a gravitational wave data-analysis application (Einstein@Home), accepted in International Conference on Parallel Processing and Applied Mathematics, Wroclaw, Poland, 2009. [14] S. Teukolsky, Perturbations of a rotating black hole, Astrophys. J. 185, 1973, 635. [15] W. Krivan, P. Laguna, P. Papadopoulos, and N. Andersson, Dynamics of perturbations of rotating black holes, Phys. Rev. D 56, 1997, 3395.