

# Using GPUs to Accelerate Analysis of Kerr Tails

John P. Dickerson\*  
University of Maryland

## Abstract

In this paper, we explore the possibility of using the GPU to accelerate arithmetic-intensive computations in black hole theory. As solutions to many scientific problems involve heavily parallel operations – matrix-vector multiplication, matrix-matrix multiplication, FFT and IFFT – we feel that GPUs provide significant application to our work. By implementing these accelerations, we provide a case study in interfacing legacy Fortran code with the GPU via the FLAGON library. We enumerate the current successful speedups realized by this coupling alongside any failed trials resulting in increased computation time. Finally, we discuss our future plans and expectations regarding this project and the state of the GPU in general scientific computing.

**CR Categories:** J.2 [Computer Applications]: Physical Sciences and Engineering—Physics; D.1.3 [Software]: Programming Techniques—Concurrent Programming

**Keywords:** GPGPU, scientific computing, CUDA

## 1 Introduction

### 1.1 Scientific Computing and Black Hole Theory

The canonical work on late-time behavior of gravitational fields of black holes was published by Karl Schwarzschild in 1916 [Schwarzschild 1916]. As a star collapses through its gravitational radius to form a black hole, it leaves behind a gravitational field. Schwarzschild's work studies the dynamics of perturbations in this gravitational field using scalar fields. His work focuses on black holes without any angular momentum or charge (known as Schwarzschild black holes), and thus he only studies *static* gravitational fields. Schwarzschild determines that any perturbations of a static black hole's gravitational field are themselves spherical. Phrased differently, the geometry of these gravitational fields is guaranteed to be spherically symmetric. Richard Price's additional work in the area recognizes this geometric symmetry and applies spherical harmonics to compactly describe the decay of these field perturbations [Price 1972].

A newer area of research involves a similar problem, only applied to the gravitational fields of Kerr black holes [Kerr 1963]. Kerr black holes, unlike their Schwarzschild counterparts, have nontrivial angular momentum, meaning they are rotating. It is believed that the center of the Milky Way contains a quickly spinning black hole of this type, GRS 1915+105, with a spin rate of around 1000 rotations per second [Eckart et al. 2004]. Nontrivial angular momentum destroys gravitational fields' spherical symmetry and thus complicates the spherical harmonics-based analytical description with additional multipoles [Hod 2000] [Barack and Ori 1999] [Burko and Khanna 2003].

The analytical descriptions provided by two of these papers, Hod [Hod 2000] and Burko [Burko and Khanna 2003], differ in their predictions. Hod enumerates a list of equations that describe late-time field decay, where an appropriate equation is chosen based on the initial pure multipole structure of the gravitational field (at

time  $t = 0$ ). Contrary to this, Burko's work claims fields decay in a manner similar to that of a Schwarzschild black hole's late time decay.

Work done by Manuel Tiglio at the University of Maryland explores if and when either case is correct [Tiglio et al. 2007]. Code written to simulate the decay of a Kerr black hole's gravitational field shows that, for low initial multipole values, Hod's predictions match Burko's. However, for higher initial multipole values, their predictions diverge. Furthermore, Tiglio's work supports Burko's model in the case of Kerr-Schild coordinates and Hod's model in the case of Boyer-Lindquist coordinates being used.

The calculations required to simulate the evolution of these perturbations are computationally intensive, as seen in Tiglio's and others' work [Tiglio et al. 2007] [Burko and Khanna 2003]. Our goal is to leverage the power of consumer-level graphics cards to significantly decrease the execution time of these simulations.

### 1.2 Scientific Computing on GPUs

Initially, heavy interest in stand-alone graphics processors for gaming and multimedia pushed the price of this dedicated hardware into the consumer realm. As such, both the architecture and programming styles associated with GPUs heavily catered to problems in the graphics realm. Although many non-visual scientific computing problems were successfully mapped onto graphics card during this time [Manavski 2007] [Szerwinski and Guneysu 2008], the recent release of Nvidia Corporation's CUDA and the impending release of the Open Computing Language (OpenCL) framework are ushering in the era of general purpose GPU computing.

Prior to the release of CUDA-capable devices, research teams interested in leveraging the power of GPUs for general scientific computing had to first map their problem into the graphics realm. In general, this involved writing shaders to take advantage of the programmable portions of the GPU – vertex and fragment processors – and using onboard texture memory to load and store results. Nvidia's CUDA architecture takes strides toward opening the GPU to more general application via hardware reorganization. Its accompanying language, C for CUDA, attempts to further alleviate the inconvenience of remapping a problem by providing general high- and low-level APIs for accessing the parallel elements of the GPU [Halfhill 2008].

True to its roots in graphics, today's GPU is specialized for compute-intensive, highly parallelized computation [Gummaraju and Rosenblum 2005]. CUDA provides a useful abstraction of the graphics card as a device capable of executing millions of lightweight *threads* in parallel. These threads are organized into *blocks*, with each thread in the block having access to its own registers and the block's shared memory, as well as the capability to synchronize with neighboring threads. A *kernel* of code is executed on one or more of these blocks. As CUDA-capable devices allow generic DRAM access (both gather and scatter), each thread has access to the GPU's onboard and texture memory. David Luebke of Nvidia Research provides a more thorough explanation of the CUDA architecture, including memory access patterns and synchronization, in his talk from Supercomputing 2008 [Luebke 2008].

\*e-mail: spook@umiacs.umd.edu

## 2 Methods

Certain logistical problems exist with any new technology. In our case, the legacy serial code on which we based our hybrid CPU-GPU accelerated code is written in Fortran. As the high-level API available for CUDA devices is based on C, some interface was necessary for cross-language communication. We have had success using the relatively new FLAGON library, a Fortran 9x wrapper built by Ramani Duraiswami at the University of Maryland, to tie the two codebases together [N. Gumerov 2007].

### 2.1 Theoretical Basis

The original, serial code for our gravitational field perturbation evolution *evolves* a field over  $n$  iterations, representing time steps, using Euler explicit time stepping. We deal first with a simplified 2D case; upon success, we will port results to the 3D case. For this 2D case, our dataset  $u$  is dependent on four components

$$u = \begin{pmatrix} u_1(t, r, \theta) \\ \vdots \\ u_4(t, r, \theta) \end{pmatrix} \quad (1)$$

Here  $t$  represents the current time step, while  $r$  and  $\theta$  represent the distance and angle offset from our origin, a Kerr black hole.

We are interested in solving a first order PDE of the  $4 \times 4$  system

$$F = \frac{\partial u}{\partial t} = A^r(u) \frac{\partial u}{\partial r} + A^\theta(u) \frac{\partial u}{\partial \theta} + G(u) \quad (2)$$

We expand  $u$  by differentiating with Chebyshev polynomials for the  $\partial_r u$  case, differentiating with Legendre polynomials for the  $\partial_\theta u$  case, and finally including  $G(u)$ . In the 2D case, this  $G(u)$  is relatively trivial and will not be addressed. From here, we can use either fourth-order Runge-Kutte integration or standard explicit time marching to move forward.

The implementation described below uses explicit time marching

$$u_{new} = u_{old} \cdot F \cdot \Delta t \quad (3)$$

to advance one step. RK4 will be implemented in the future.

### 2.2 Implementation on the GPU

Since we are using the relatively simple time stepping method mentioned above, the majority of our calculation time is spent computing  $F$ . We choose to attack  $F$  in parts, computing

$$F = A^r u_{old} \quad (4)$$

$$F = F + A^\theta u_{old} \quad (5)$$

$$F = F + G(u_{old}) \quad (6)$$

As with many spectral methods, our two differentiation calls are computationally-intensive, as opposed to data-intensive. It is with this knowledge that we begin mapping them to the GPU. At their roots, both the Chebyshev- and Legendre-based differentiation methods revolve around matrix-vector multiplication. General matrix-vector and matrix-matrix multiplication have been studied extensively on both the CPU and GPU. Extremely optimized routines exist on the CPU in the BLAS library, as well as on the GPU in libraries such as CUBLAS [Nvidia 2008a] and CUDPP [Harris et al. 2007].

The FLAGON library provides a handle for executing CUBLAS-based general matrix-vector multiplication on the GPU from legacy

Fortran code. We first tried naively calling these functions, essentially replacing any calls to CPU-based serial BLAS code with calls to GPU-based CUBLAS functions. Although correct results were returned, the “simple” act of transferring multiple small input and output arrays back and forth between the host and the GPU device horribly overshadowed any performance gains realized by moving to parallel.

From here, a series of modifications were made to our approach. The matrices  $A^r$  and  $A^\theta$  remain constant, and as such can be moved to either global or constant memory on the device. Furthermore, the various elements of  $u$  can be moved to and from the GPU exactly once per derivative per time step, in one lump. Although the same number of calls to CUBLAS functions are made, this strategy severely reduces the setup cost for each call.

## 3 Performance Analysis

Our performance analysis is split into three parts: the local speedup from the GPU, the overall speedup for our entire project, and the effect of the GPU on the error of our results.

For our specific case, we noticed a speedup of approximately four times for spectral differentiation (Figure 1). This is a low speedup compared to similar projects. Our speedups are choked by the relatively small matrices on which we perform the matrix-vector multiplications necessary for differentiation. Were larger matrices needed, we would begin to see the extremely high cost of memory management (moving our data and results to and from the device) amortized.

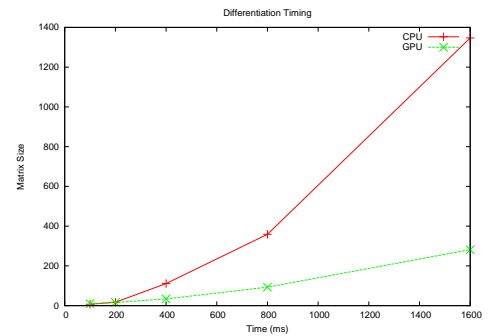


Figure 1: Local Execution Time

Overall, our entire project spends roughly 10% of its execution time in the serial spectral differentiation routines. Because of this, the speedup realized by executing our spectral differentiation code on the GPU is reduced significantly. Our overall speedup (for the largest tested input matrix size) is 1.074, as seen in Figure 2. For lower values of  $n$ , the heavy cost of data transfer further reduces this increase.

Ahmdal’s Law provides an explanation for this phenomenon.

$$s = \frac{1}{(1 - P) + \frac{P}{S}} \quad (7)$$

$$= \frac{1}{(1 - 0.10) + \frac{0.10}{4.0}} \quad (8)$$

$$\approx 1.081 \quad (9)$$

Setting  $P = 0.10$  and  $S = 4.0$ , we expect to see a performance increase of approximately 1.081. Our real performance gain of 1.074 matches this prediction.

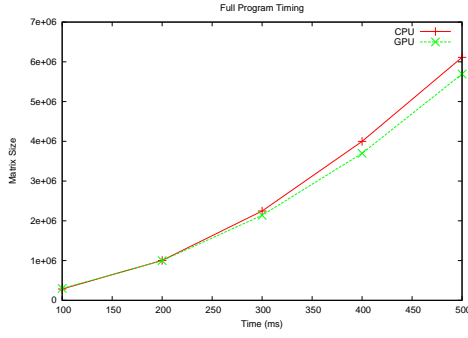


Figure 2: Global Execution Time

When moving any scientific computation from one architecture to another, it is important to observe and analyze possible sources of computational error. This holds especially true with computation on graphics cards. The cost of moving from single- to double-precision calculation on the GPU is significantly higher than on a CPU. This exaggerated cost is rooted in the field of graphics; often, the added acuity of double-precision calculation is not necessary for determining the color of a pixel on a display.

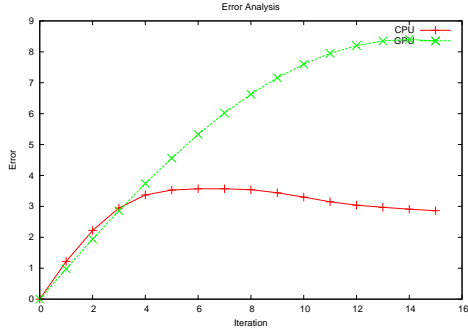


Figure 3: Error Analysis

We investigated the effect that this double to single conversion had on our results, comparing our single-precision GPU implementation to the double-precision CPU standard. Indicated in Figure 3, the error resulting from the GPU code stabilizes at slightly higher than twice the error of the CPU code. This is to be expected.

## 4 Future Work and Conclusions

The work presented here represents only the beginning stages of a more intensive project. Many improvements can be made to further our research. Our general course of action is listed below.

### 4.1 Spectral Derivative Optimization

Widely accepted libraries like CUBLAS, CUDPP, and CUFFT provide quick, efficient, and heavily tested implementations of common functions. To provide easy application to many domains, they must be extremely general. As we can afford to lose this generality, we would like to take the optimized implementations found in these libraries and see if any further speedup is available. More specifically, since we use the elements in  $A^r$  and  $A^\theta$  in multiple matrix-vector multiplications, there is the potential to leverage shared memory to lessen the total number of reads from the  $A$  matrices overall.

### 4.2 RK4 Integration

We would like to use fourth order Runge-Kutta integration to evolve our data. The RK4 iterative solver is compute-heavy and maps well to the GPU. We will be able to store the four matrices involved in RK4 integration as data in constant or global memory on the GPU, hopefully for a performance gain.

### 4.3 FLAGON Benchmarking

Part of the motivation behind this project was to provide a case study for the FLAGON library. FLAGON has proven itself to be a straightforward and fairly intuitive tool for combining a familiar language – Fortran – with a new technology. However, it would be interesting to get some measure of the overhead of interfacing via FLAGON.

By porting our basic code to the C programming language, we aim to provide such a benchmark. This will serve as a useful tool for weighting the potential slowdown of connecting CUDA to existing Fortran code instead of completely rewriting an existing program in C or C++.

### 4.4 Move from 2D to 3D

Our code can be used to solve a specific set of problems in the 2D case. Moving from polar to spherical coordinates will significantly broaden the application of this research.

In the 3D case, the gravitational field is composed of fifty-four components:

$$u_{3D} = \begin{pmatrix} u_1(t, r, \theta, \phi) \\ \vdots \\ u_{54}(t, r, \theta, \phi) \end{pmatrix} \quad (10)$$

Where  $t$  is the current time step, and  $r$ ,  $\theta$ , and  $\phi$  are in spherical coordinates. This added complexity (compared to four functions in 2D) is the reason for testing first on the lower dimensional case. Furthermore, the previously simple  $G(u)$  computation moves from a few lines of code to thousands of compute-intensive operations, and can no longer be viewed as trivial.

### 4.5 Multiple GPUs

We will certainly see a significant increase in computation time for the 3D case. In its current serial state, a single run takes on the order of months to complete. Since  $G(u)$  is extremely arithmetically intensive, we hope to see huge speedups from the GPU. However, even the most optimistic predictions would bring runtime into the order of days. Because of this, we would like to explore multi-GPU solutions via Nvidia's Tesla systems.

FLAGON does not currently spread computation across multiple GPUs. However, support for this functionality is being added.

### 4.6 Visualization

At its core, our project models the propagation of a wave along a field. We create animated graphics of this evolution after a complete run of the code. A more interactive and timely display would aid in visualizing this multidimensional data. Support for this is being added to FLAGON via the FreeGLUT library.

## Acknowledgements

Thanks to Manuel Tiglio for the initial code and physics advice, Ramani Duraiswami for the project idea, and Kate Despain for the Fortran/FLAGON Makefile help.

## References

- BARACK, L., AND ORI, A. 1999. Late-time decay of scalar perturbations outside rotating black holes. *Phys. Rev. Lett.* 82, 22 (May), 4388–4391.
- BEN-YU, G. 1998. *Spectral methods and their applications*. World Scientific.
- BURKO, L. M., AND KHANNA, G. 2003. Radiative falloff in the background of rotating black holes. *Phys. Rev. D* 67, 8 (Apr), 081502.
- COOLEY, J. W., AND TUKEY, J. W. 1965. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation* 19, 90, 297–301.
- ECKART, A., GENZEL, R., AND SCHÖDEL, R. 2004. The massive accreting black hole at the center of the milky way \*. *Progress of Theoretical Physics Supplement* 155, 159–165.
- EDELMAN, A., AND MCCORQUODALE, P. 1999. The future fast fourier transform. *SIAM J. Sci. Computing* 20, 1094–1114.
- GUMMARAJU, J., AND ROSENBLUM, M. 2005. Stream processing in general-purpose processors.
- HALFHILL, T. 2008. Parallel Processing with CUDA. *Microprocessor Journal*.
- HARRIS, M., OWENS, J., SENGUPTA, S., ZHANG, Y., AND DAVIDSON, A., 2007. CUDPP: CUDA data parallel primitives library.
- HOD, S. 2000. Mode coupling in rotating gravitational collapse: Gravitational and electromagnetic perturbations. *Phys. Rev. D* 61, 6 (Feb), 064018.
- KERR, R. P. 1963. Gravitational field of a spinning mass as an example of algebraically special metrics. *Phys. Rev. Lett.* 11, 5 (Sep), 237–238.
- LARS NYLAND, MARK HARRIS, J. P. 2007. *Fast N-Body Simulation with CUDA*. Addison-Wesley Professional.
- LUEBKE, D. 2008. GPU Architecture: Implications and Trends. In *Supercomputing 08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*.
- MAKINO, J., FUKUSHIGE, T., AND KOGA, M. 2000. A 1.349 tflops simulation of black holes in a galactic center on grape-6. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, IEEE Computer Society, Washington, DC, USA, 43.
- MANAVSKI, S. 2007. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In *Proceedings of IEEE International Conference on Signal Processing and Communication*, 65–68.
- N. GUMEROV, R. DURAISWAMI, W. D. 2007. Middleware for programming nvidia gpus from fortran 9x. *Supercomputing 2007*.
- NVIDIA. 2008. Cuda: Cublas library.
- NVIDIA. 2008. Cuda: Cufft library.
- PRICE, R. H. 1972. Nonspherical perturbations of relativistic gravitational collapse. ii. integer-spin, zero-rest-mass fields. *Phys. Rev. D* 5, 10 (May), 2439–2454.
- ROMERO, S., TRENAS, M. A., GUTIERREZ, E., AND ZAPATA, E. L. 2007. Locality-improved fft implementation on a graphics processor. In *ISCGAV'07: Proceedings of the 7th WSEAS International Conference on Signal Processing, Computational Geometry & Artificial Vision*, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 58–63.
- SCHWARZSCHILD, K. 1916. On the gravitational field of a sphere of incompressible fluid according to einstein's theory.
- SZERWINSKI, R., AND GUNEYSU, T. 2008. Exploiting the Power of GPUs for Asymmetric Cryptography. *Lecture Notes in Computer Science* 5154, 79–99.
- TIGLIO, M., KIDDER, L., AND TEUKOLSKY, S. 2007. High accuracy simulations of kerr tails: coordinate dependence and higher multipoles.