J Sci Comput (2013) 56:366–380
DOI 10.1007/s10915-012-9679-3

High-Precision Numerical Simulations on a CUDA GPU: Kerr Black Hole Tails

Gaurav Khanna

Received: 13 August 2012 / Revised: 14 November 2012 / Accepted: 19 December 2012 /
Published online: 8 January 2013
© Springer Science+Business Media New York 2013

Abstract Computational science has advanced significantly over the past decade and has impacted almost every area of science and engineering. Most numerical scientific computation today is performed with double-precision floating-point accuracy (64-bit or $\sim$15 decimal digits); however, there are a number of applications that benefit from a higher level of numerical precision. In this paper, we describe such an application in the research area of black hole physics: studying the late-time behavior of decaying fields in Kerr black hole space-time. More specifically, this application involves a hyperbolic partial-differential-equation solver that uses high-order finite-differencing and quadruple (128-bit or $\sim$30 decimal digits) or octal (256-bit or $\sim$60 decimal digits) floating-point precision. Given the computational demands of this high-order and high-precision solver, in addition to the rather long evolutions required for these studies, we accelerate the solver using a many-core Nvidia graphics-processing-unit and obtain an order-of-magnitude speed-up over a high-end multi-core processor. We thus demonstrate a practical solution for demanding problems that utilize high-precision numerics today.

Keywords CUDA · GPU · Precision · Tails · Quadruple · Octal

1 Introduction

Computation has joined theory and experiment as a third pillar of science and engineering research today. One aspect of scientific computing that perhaps is not discussed as much currently, is the issue of numerical floating-point precision. Most scientific applications and algorithms utilize double-precision floating-point (64-bit) accuracy, which translates to numerical

G. Khanna (✉)
Physics Department, University of Massachusetts Dartmouth, 285 Old Westport Rd.,
North Dartmouth, MA 02747, USA
e-mail: gkhanna@umassd.edu

computations accurate to $\sim$15 decimal digits. It was in the early 1960s that mainframe manufacturers like IBM began offering support for double-precision numerics, to prevent problems due to accumulation of round-off errors.[1] Now, there is a growing list of applications and areas that benefit from higher than double-precision accuracy i.e. quadruple (128-bit) and even octal (256-bit) precision. Examples include: *Chaotic aspects of Climate Modeling, Supernova Simulations, Quantum Calculations in Atomic and Nuclear Physics, Electromagnetic Scattering, Vortex studies in Fluid Flows, Experimental Mathematics, Planetary Orbit Computations* etc. More detail on why these different problems benefit from high-precision numerics may be found in Ref. [1] and the references therein. The need for high-precision numerics is expected to grow rapidly as the scientific community is in a position to perform even larger and longer scale simulations [2]. Computations utilizing higher-order methods (pseudo-spectral, radial-basis-functions, etc.) are likely to benefit from high-precision numerics [4,5]. There may also be some benefits in using high-precision numerics in the context of studying some borderline ill-conditioned problems; and not necessarily only cases where very high accuracy is desirable [6]. Now, single- and double-precision floating-point hardware support is available in all major compute hardware today, but support for higher precision is severely lacking. One can easily emulate such high levels of numerical precision through CPU software libraries, but these often perform orders-of-magnitude slower than the full hardware-supported double-precision equivalents [1], thus making such an option somewhat impractical. In addition, it is highly unlikely that the computer industry will develop a commodity processor in the near future that natively supports quadruple or higher floating-point precision. We suggest here an interim solution to this major challenge, leveraging the massive parallelism offered by the many-core GPU architecture. These compute technologies are expected to play a critical role in the future of supercomputing, especially in scaling up from the peta-scale to the exa-scale regime.

In this paper, we make use of a many-core Nvidia Tesla M2050 "Fermi" CUDA GPU [7] to accelerate an application from the Numerical Relativity (NR) community: a Teukolsky equation solver for studying late-time radiative "tails" in Kerr black hole spacetime [8–12]. This solver is essentially a hyperbolic partial-difference-equation (PDE) code that uses high-order finite-differencing. One distinguishing aspect of this application is that the numerical simulations are such that they require high numerical precision i.e. quadruple and octal floating-point precision. In this work, we focus on the outcome of implementing high-precision floating-point computation on a many-core GPU and compare the resulting performance with that from a traditional multi-core CPU.

This paper is organized as follows: In Sect. 2, we briefly introduce the Teukolsky equation, the relevant background gravitational physics and the numerical method used by the solver code. We also emphasize the need for high numerical precision and high-order finite-differencing for the study of late-time Kerr tails. In Sect. 3, we provide a very brief introduction to CUDA GPUs and emphasize those aspects that are relevant to the CUDA implementation we present. In Sect. 4, we describe the parallel CUDA code's implementation details and then in Sect. 5 we present this code's overall performance results. Finally, in Sect. 6, we summarize this work and make some conclusive remarks.

[1] Perhaps the most significant and tragic event that occurred due a limited numerical precision issue in recent history was the failure of the *Patriot Missle* during the Persian Gulf War in the early 1990s. This was attributed to the limited precision capabilities of the control computer that was unable to operate accurately over long periods of time due to the accumulation of round-off error [3].

2 Late-Time Kerr Black Hole Tails

Several operating gravitational wave observatories all over the world are currently being upgraded: LIGO in the United States, GEO/Virgo in Europe and TAMA in Japan. These upgraded observatories will open a new window onto the Universe by enabling scientists to make astronomical observations using a completely new medium— gravitational waves (GWs), as opposed to electromagnetic waves (light). These waves were predicted by Einstein's relativity theory, but have not been directly observed because the required experimental sensitivity was simply not advanced enough, until very recently.

Numerical Relativity [13,14] is an area of computational science that emphasizes the detailed modeling of strong sources of GWs—collisions of compact astrophysical objects, such as neutron stars and black holes. Thus, it plays an extremely important role in the area of GW astronomy and gravitational physics, in general. Moreover, the NR community has also contributed to the broader computational science community by developing an open-source, modular, parallel computing infrastructure called *Cactus* [15].

The specific NR application we have chosen for consideration in this work is one that very accurately evolves the perturbations of a rotating (Kerr) black hole i.e. solves the Teukolsky equation in the time-domain [8–12]. This equation is essentially a linear wave-equation in Kerr space-time geometry. The following subsections provide more detailed information on this equation and the associated numerical solver code. There are also other applications in gravitational physics where high-precision numerics could potentially be beneficial: *Critical phenomena* associated to gravitational collapse [16] and also in other contexts; *particle self-force computations* that are necessary for modeling extreme-mass-ratio inspirals (EMRIs) [17] that are important sources of GWs for future space-borne missions.

2.1 Teukolsky Equation

The Teukolsky master equation describes scalar, vector and tensor field perturbations in the space-time of Kerr black holes [18]. In Boyer–Lindquist coordinates, this equation takes the form

$$
\begin{aligned}
&-\left[\frac{(r^2+a^2)^2}{\Delta} - a^2\sin^2\theta\right]\partial_{tt}\Psi - \frac{4Mar}{\Delta}\partial_{t\phi}\Psi \\
&-2s\left[r - \frac{M(r^2-a^2)}{\Delta} + ia\cos\theta\right]\partial_t\Psi \\
&+\Delta^{-s}\partial_r\left(\Delta^{s+1}\partial_r\Psi\right) + \frac{1}{\sin\theta}\partial_\theta\left(\sin\theta\,\partial_\theta\Psi\right) \\
&+\left[\frac{1}{\sin^2\theta} - \frac{a^2}{\Delta}\right]\partial_{\phi\phi}\Psi + 2s\left[\frac{a(r-M)}{\Delta} + \frac{i\cos\theta}{\sin^2\theta}\right]\partial_\phi\Psi \\
&-\left(s^2\cot^2\theta - s\right)\Psi = 0,
\end{aligned} \tag{1}
$$

where M is the mass of the black hole, a its angular momentum per unit mass, $\Delta = r^2 - 2Mr + a^2$ and s is the "spin weight" of the field. The $s = 0$ versions of these equations describe the radiative degrees of freedom of a simple scalar field and are the equations of interest in this work. As mentioned previously, this equation is an example of linear, hyperbolic (3+1)D PDEs which are quite common in several areas of science and engineering and can be solved numerically using a variety of finite-difference schemes.

2.2 Teukolsky Code

Reference [20] demonstrated stable numerical evolution of Eq. (1) for using the well-known second-order, time-explicit, two-step Lax–Wendroff numerical evolution scheme. The Teukolsky code presented in this paper uses a similar approach, therefore the contents of this section are largely a review of the work presented in the relevant literature [20].

Our code uses the tortoise coordinate r^* in the radial direction and azimuthal coordinate $\tilde{\phi}$. These coordinates are related to the usual Boyer–Lindquist coordinates by

$$dr^* = \frac{r^2 + a^2}{\Delta} dr \tag{2}$$

and

$$d\tilde{\phi} = d\phi + \frac{a}{\Delta} dr \ . \tag{3}$$

These coordinates are better suited for performing numerical evolutions in a Kerr space-time background for a number of reasons that are detailed in Ref. [20] that we will not repeat here. Next, we factor out the azimuthal dependence and use the ansatz,

$$\Psi(t, r^*, \theta, \tilde{\phi}) = e^{im\tilde{\phi}} r^3 \Phi(t, r^*, \theta) \tag{4}$$

that allows us to reduce the dimensionality of the PDE to (2+1)D. Defining

$$\Pi \equiv \partial_t \Phi + b\, \partial_{r^*} \Phi \ , \tag{5}$$

$$b \equiv \frac{r^2 + a^2}{\Sigma} \ , \tag{6}$$

and

$$\Sigma^2 \equiv (r^2 + a^2)^2 - a^2\, \Delta\, \sin^2\theta \tag{7}$$

allows the Teukolsky equation to be rewritten in first order form as

$$\partial_t \boldsymbol{u} + \boldsymbol{M} \partial_{r*} \boldsymbol{u} + \boldsymbol{L}\boldsymbol{u} + \boldsymbol{A}\boldsymbol{u} = 0, \tag{8}$$

where

$$\boldsymbol{u} \equiv \{\Phi_R, \Phi_I, \Pi_R, \Pi_I\} \tag{9}$$

is the solution vector. The subscripts R and I refer to the real and imaginary parts respectively (note that the Teukolsky function Ψ is a complex valued quantity). Explicit forms for the matrices $\boldsymbol{M}$, $\boldsymbol{A}$ and $\boldsymbol{L}$ can be easily found in the relevant literature [20]. Rewriting Eq. (8) as

$$\partial_t \boldsymbol{u} + \boldsymbol{D} \partial_{r^*} \boldsymbol{u} = \boldsymbol{S} \ , \tag{10}$$

where

$$\boldsymbol{D} \equiv \begin{pmatrix} b & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & -b & 0 \\ 0 & 0 & 0 & -b \end{pmatrix}, \tag{11}$$

$$\boldsymbol{S} = -(\boldsymbol{M} - \boldsymbol{D}) \partial_{r^*} \boldsymbol{u} - \boldsymbol{L}\boldsymbol{u} - \boldsymbol{A}\boldsymbol{u}, \tag{12}$$

and using the two-step Lax–Wendroff iterative scheme, we obtain stable evolutions. Each iteration consists of two steps: In the first step, the solution vector between grid points is obtained from

$$\begin{aligned} \boldsymbol{u}_{i+1/2}^{n+1/2} = {} & \frac{1}{2}\left(\boldsymbol{u}_{i+1}^{n} + \boldsymbol{u}_{i}^{n}\right) \\ & - \frac{\delta t}{2}\left[\frac{1}{\delta r^*}\boldsymbol{D}_{i+1/2}^{n}\left(\boldsymbol{u}_{i+1}^{n} - \boldsymbol{u}_{i}^{n}\right) - \boldsymbol{S}_{i+1/2}^{n}\right]. \end{aligned} \tag{13}$$

This is used to compute the solution vector at the next time step,

$$\boldsymbol{u}_{i}^{n+1} = \boldsymbol{u}_{i}^{n} - \delta t\left[\frac{1}{\delta r^*}\boldsymbol{D}_{i}^{n+1/2}\left(\boldsymbol{u}_{i+1/2}^{n+1/2} - \boldsymbol{u}_{i-1/2}^{n+1/2}\right) - \boldsymbol{S}_{i}^{n+1/2}\right]. \tag{14}$$

The angular subscripts are dropped in the above equation for clarity.

Following Ref. [20], we set Φ and Π to zero on the inner and outer radial boundaries. Symmetries of the spheroidal harmonics are used to determine the angular boundary conditions: For even $|m|$ modes, we have $\partial_\theta \Phi = 0$ at $\theta = 0, \pi$ while $\Phi = 0$ at $\theta = 0, \pi$ for modes of odd $|m|$.

2.3 The Tails Problem

The main science goal for the development of such a Teukolsky equation solver in the context of this work is to study the "controversial" Kerr black hole "tails" problem. The statement of the problem is simple: place an observer in a circular orbit around a black hole, and have them measure at late times a generic perturbation field, that had compact support at some initial time. It is generally accepted that the observer measures the late-time perturbation field to drop off as an inverse power-law of time, specifically as t^{-n}. In the case of a non-rotating Schwarzschild black hole, $n = 2\ell + 3$, where ℓ is the multipole moment of the initial perturbation field [21]. Namely, if the initial (compactly supported) perturbation field has the angular dependence of Y_ℓ^m, the angular dependence remains unchanged (due the hole's spherical symmetry) and the decay rate of the field is governed by the ℓ value of the initial perturbation. However, in the context of rotating black holes, it is the value of n that has been controversial in the literature, with some conflicting results reported. See for example [10] for a recent and detailed review of the controversy.

Generating accurate numerical simulations in this context involves a number of challenges. Firstly, these simulations need to be rather long—this is because typically the observed field exhibits an exponentially decaying oscillatory behavior in the initial part of the evolution and only much later this transitions over to a clean power-law decay. Therefore, one needs to wait for the initial oscillations (so called "quasi-normal ringing") to dissipate away. Secondly, because each multipole has its own decay rate (which increases with an increase in ℓ) at late times one ends up with numerical data in which different multipoles have widely different amplitudes (often 30–40 orders of magnitude apart!). For this reason, not only does the numerical solution scheme have to be high-order (to reduce the discretization errors to the required levels) but it also requires high-precision floating-point numerical computation (due to the large range of amplitudes involved). Figure 1 clearly demonstrates all these computational challenges of the problem, using a sample Kerr tails evolution.

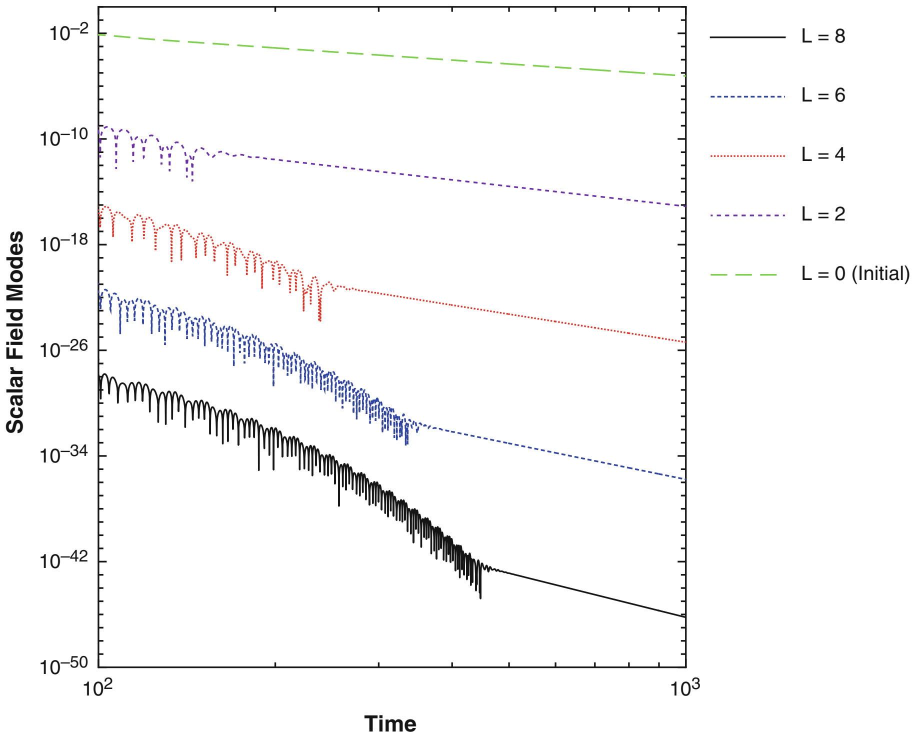

Fig. 1 Kerr tails for a range of ℓ multipoles (0–8) starting with a pure $\ell' = 0$ multipole. These tails obey the tail formula $(\ell + \ell' + 3)$ [11]. It is clear from the wide range of amplitudes involved, that such numerical simulations require octal-precision floating-point computations.

2.4 Second-Order Finite-Differencing

The standard of scientific computation in the finite-difference approach has been second-order algorithms for decades. Second-order convergence strikes a balance between accuracy, computational cost and ease of implementation. Convergence is typically guaranteed by the *Lax Equivalence Theorem* [19] which basically states if the discrete formula approximating the PDE is *consistent*, and that if numerical errors do not grow unboundedly (*stability*), then the solution is convergent in the sense that as the grid spacing goes to zero, the solution would approach the correct one.

Early codes to calculate the late-time Kerr tails were therefore designed to be second–order in all coordinates i.e. temporally, radially, and also in the angular coordinate [20]. We demonstrate below that while this approach seems compelling, it can mislead into qualitatively wrong solutions.

Consider a non-rotating Schwarzschild black hole, excited by an azimuthally symmetric ($m = 0$) field with $\ell = 4$. The decay rate of the well-known $t^{-2\ell-3}$ Price tail [21] is incontrovertible in this case: since the background is spherically symmetric, spherical harmonic modes evolve independently, and the decay rate is therefore t^{-11}. The numerical solution with a second-order (2+1)D numerical code may lead however to the erroneous conclusion that the decay rate is t^{-3}. In Fig. 2 we show the late-time field as a function of time for three choices of the angular grid spacing, and for fixed radial and temporal grid spacings. As can be

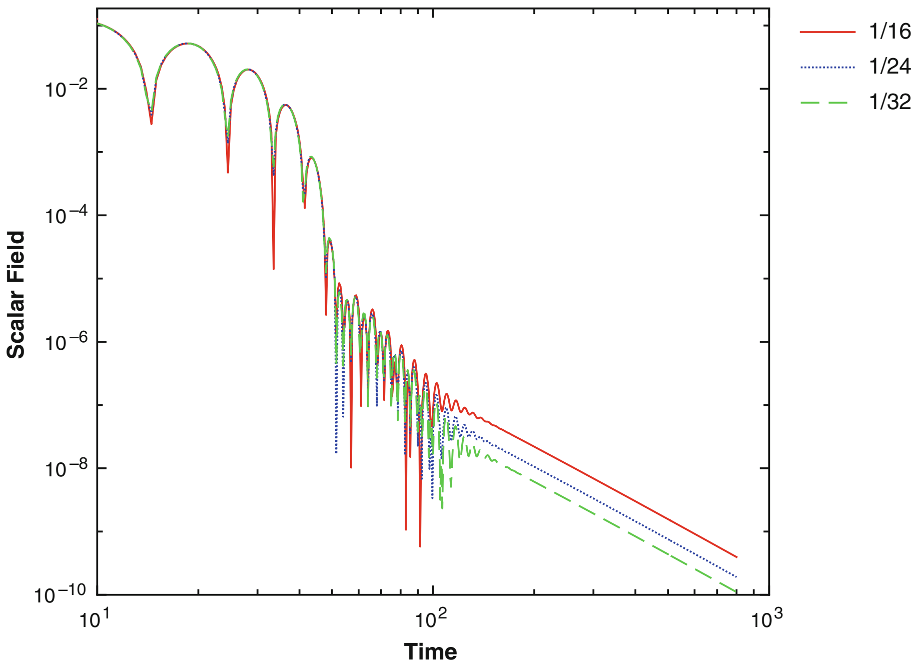

Fig. 2 Strong dependence of the tail amplitude on angular grid resolution, upon using a second-order differencing stencil. The power-law decay rate is incorrect, because the tail data is purely dominated by numerical noise. Higher-order accuracy is clearly necessary for reliable results here.

easily inferred the decay rate is t^{-3}, which is definitely wrong. The origin of this wrong result is indicated in the same figure: the field values themselves reduce considerably in amplitude under refinement of the grid! The signal seen is therefore not the physical field, but purely an evolved numerical noise ("leakage of multipoles"). While the physical signal indeed drops like t^{-11}, it is swamped by this evolved numerical noise, which has a monopole component that drops like t^{-3}. Higher angular grid resolution can postpone the numerical-noise dominated regime, but cannot completely eliminate it. If the grid is fine enough, the noise level at a particular value of the time reduces to below the level of the physical field; however, as the former drops more slowly than the latter, at very late times the field is dominated by numerical noise. *Therefore, any very long time evolution will be dominated by numerical noise*.

Thus, we have here a situation in which the Lax Equivalence Theorem appears to provide us *false* confidence in our numerical results: the numerical noise does not grow unboundedly. In fact, it even drops quadratically with the grid spacing, and drops with time. However, it does so, slower than the physical signal, so that the signal may be dominated by numerical noise even though the regular theorem criteria are satisfied. The key point here is to note that convergence guarantees the correct solution only in an asymptotic sense i.e. in the limit of zero grid spacing. It makes no statement whatsoever about the accuracy of the numerical result in the context of a practical, non-zero grid spacing. And that is the reason for the unexpected outcome we mention above in the context of the non-rotating Schwarzschild case.

2.5 High-Order and High-Precision

As noted above, it is clear that we require higher-order finite-differencing to solve the Teukolsky equation in the context of these Kerr tails simulations[2]. It turns out that it is sufficient that only the angular differentiation (i.e. θ-derivatives of the field) be implemented using a higher-order numerical stencil. The temporal and the radial direction related operations can simply stay second-order and such a mixed approach yields sufficiently good results [10]. This is likely because the dependence of the field on radius and time is significantly more gradual when compared to the polar-angle dependence. In addition, the angular differentiation involves computing first- and second-order derivatives (see Eq. 1), whereas the radial-time part is cast into first-order form (see Eq. 10). For this reason, in this work we choose the finite-difference angular differentiation operator to be 10th-order accurate and leave the rest of the numerical scheme as a standard second-order Lax–Wendroff algorithm. In addition, as pointed out before, we also require high-numerical precision: in particular, quadruple and octal precision may be required depending upon the details of Kerr tails simulation being attempted.

Both these computationally demanding requirements make performing scientifically meaningful Kerr tails numerical simulations rather difficult, especially using traditional desktop processors. For this reason, in this work we turn to hardware accelerators such as GPUs. Now, double-precision (64-bit) floating-point operations are supported on nearly all compute hardware including CUDA GPUs and multi-core processors. Therefore, no special considerations are necessary for double-precision computations. On the other hand, only a few options support quadruple-precision (128-bit) datatype and operations. And to the best of our knowledge, no options directly support octal-precision (256-bit) arithmetic.

Therefore, finding a software solution for these high-precision requirements is necessary. After examining a number of open-source high-precision floating point arithmetic packages, we find that the LBNL QD library is one that is well suited for both CPU [25] and GPU [26] architectures. In this library, the high-precision datatypes (quadruple and octal precision types) are implementing using a representation based on the appropriate number of double-precision floats (*double-double* (DD) for quadruple, and *quad-double* (QD) for octal precision) and similarly the high-precision floating-point operations are performed ultimately using standard double-precision operations. Fig. 1 depicts some sample results from a Kerr tails simulation that makes use of all these enhancements in floating-point precision and also a high-order accurate numerical evolution scheme.

3 Nvidia CUDA GPU

All processor manufacturers have moved towards multi-core designs today in the quest for higher performance. At the time of the writing of this article, high-end processors by Intel have a maximum of eight (8) cores. On the other hand, there are other computing technologies that have been in existence for several years that have traditionally had many more compute cores than standard desktop processors. These are sometimes referred to as

[2] A pseudo-spectral approach is better suited to address the challenges mentioned in this context; however, our higher-order finite-difference implementation requires relatively modest changes to our original second-order code, therefore we simply proceed with that approach. It is worth pointing out that other work in the same context using a pseudo-spectral approach posed similar precision issues [22], and thus made use of quadruple precision numerics on a CPU.

hardware accelerators and have a *many*-core design. The best example of such an accelerator is a graphics-processing-unit (GPU).

In the context of Nvidia's CUDA, a many-core GPU (called *device*) is accessible to the CPU (called *host*) as a co-processor with its own memory. The device executes a function (usually referred to as a *kernel*) in a data-parallel model i.e. a number of threads run the same program on different data. The many-core architecture of the GPU makes it possible to apply a kernel to a large quantity of data in one single call. If the hardware has a large number of cores, it can process them all in parallel (for example: Nvidia's Tesla M2050 GPU used in this work has as many as 448 compute cores clocked at 1.3 GHz). In the context of high-performance computing, this approach of *massive parallelism* plays a critical role. GPUs also provide significant flexibility in terms of memory management: Six (6) main types of memory exist in the form of *registers, local memory, shared memory, global memory, constant memory* and *texture memory*. We will not attempt to go into detail with these different memory arrangements in this document; instead we will simply refer the reader to online resources on this somewhat involved topic [7].

4 Implementation Details

In this section we briefly document our approach towards parallelism, designed to take advantage of the many cores of a CUDA GPU. We describe here the main idea we adopted, while the resulting final performance details appear in the next section. Most of the ideas presented here have been borrowed from a double-precison OpenCL implementation of the Teukolsky EMRI code, which is a second-order, Teukolsky equation solver with a complex particle-source-term [23,24].

A data-parallel model is relatively straightforward to implement in a code like the one we consider here. We simply perform a domain-decomposition of the finite-difference numerical grid and allocate the different parts of the grid to different cores. More specifically, on the CUDA GPU, each thread performs all the computations for a single pair of r^* and θ grid values. In addition, it is necessary to establish the appropriate data communication between the GPU cores—we make use of global memory on the GPU to simplify communication between the GPU cores. We estimate that this simplification (only making use of global memory) will not impact overall performance significantly because of the relatively intense computation involved in the high-precision floating-point calculations i.e. the *arithmetic intensity* of the computation is very high.

Its worth pointing out that, we port *all* the Lax–Wendroff related compute routines (such as the computation of the evolved fields half-way between grid points, the boundary condition imposition, updating of the fields using the right-hand-side data) as separate kernels onto the GPU. In this manner, no communication would be necessary with the rest of the computer system and we would thus overcome the challenge of working with the relatively poor bandwidth of the system's PCIe bus where the GPU is located. It is worth noting that some of these routines are perhaps not ideal for execution on the GPU (for example, some don't quite have the same level of parallelism that would be essential to obtain high performance from the GPU architecture) but we still port these over for execution on the GPU regardless, simply because the goal is to minimize data transfer back and forth from main memory.

Below, we depict a sample kernel from the CUDA code. This kernel updates the fields after the right-hand-side computation and gets them ready for the next time-step. The array variables are defined as follows: **qre** and **qim** are the real and imaginary parts of the solution Φ, while **pre** and **pim** are the real and imaginary parts of the "momentum" Π. The integers

a and **b** label the array indices that are relevant to the r and θ grid values involved in the kernel that have sizes **N** and **M** respectively.

```
/* ---------------------------------------------------- */

#include <gqd.cu>       /* load GQD library functions */
#define idx(b,a) (N*(b)+(a))    /* 2D field index map */

__global__ void update (gdd_real* qre, gdd_real* qim,
                          gdd_real* pre, gdd_real* pim)
{
int b;
int a;

/* CUDA thread index labels */
b = blockIdx.y * blockDim.y + threadIdx.y;
a = blockIdx.x * blockDim.x + threadIdx.x;

/* perform time-stepping update */
if (b >= 5 && b < M-5)
if (a >= 1 && a < N-1)
{
 qre[idx(b,a)] = qre[idx(b,a)] + dt * rhs_qre[idx(b,a)];
 qim[idx(b,a)] = qim[idx(b,a)] + dt * rhs_qim[idx(b,a)];
 pre[idx(b,a)] = pre[idx(b,a)] + dt * rhs_pre[idx(b,a)];
 pim[idx(b,a)] = pim[idx(b,a)] + dt * rhs_pim[idx(b,a)];
}
}

/* ---------------------------------------------------- */
```

Finally, as mentioned already, we implement high-precision floating-point operations by using a port of the LBNL QD library for the GPU cores [26]. This library makes a number of GPU-specific optimizations to deliver high performance: it represents data in an arrangement that allows for *coalesced* memory accesses; it utilizes the limited, but fast *local memory* storage on the GPU and also uses *vector* datatypes and operations. Its worth pointing out that in earlier work [27], we had also developed a port of the QD library for the STI Cell BE and also Nvidia Tesla GPUs[3]. Its also worth noting that recently Nvidia has released a *beta* version of its own quadruple-precision floating-point software implementation for its CUDA GPUs.

5 Performance Results

In this section, we report on the final results from the CUDA parallel implementation as described in the previous section. We use the following hardware for performance tests: a high-end Linux workstation with dual Intel "Sandy-Bridge" Xeon E5-2600 8-core processors clocked at 2.2 GHz and 32 GBs of RAM memory. This system is equipped with a Nvidia Tesla

[3] The Cell BE port of the QD library performed quite well. However, the GPU implementation performed poorly, simply because we did not make any GPU-specific optimizations at the time and worked with much older, relatively low-performing hardware.

Table 1 The relative performance of basic arithmetic operations using quadruple (double-double) precision numerical accuracy on a multi-core Intel Xeon CPU and a Nvidia Fermi CUDA GPU. The baseline here is the performance of a single-core of the Xeon CPU

Operation	16-cores	GPU	Ratio
DD-add	1.7	6.5	3.8
DD-mul	3.3	13	3.9
DD-div	7.1	31	4.4
DD-sqr	12	51	4.3
DD-exp	14	100	7.1
DD-log	14	110	7.9
DD-sin	14	50	3.6
DD-tan	14	72	5.1

Table 2 The relative performance of basic arithmetic operations using octal (quad-double) precision numerical accuracy on a multi-core Intel Xeon CPU and a Nvidia Fermi CUDA GPU. The baseline here is the performance of a single-core of the Xeon CPU

Operation	16-cores	GPU	Ratio
QD-add	8.6	30	3.5
QD-mul	14	40	2.9
QD-div	14	82	5.9
QD-sqr	14	84	6.0
QD-exp	15	79	5.3
QD-log	14	74	5.3
QD-sin	14	40	2.9
QD-tan	14	38	2.7

M2050 CUDA GPU accelerator with three (3) GBs of video memory. Standard open-source GCC compiler suite for code development is available on this system.

In Tables 1 and 2 we present the relative performance of basic arithmetic operations using quadruple (128-bit) and octal (256-bit) precision numerical accuracy, respectively. The baseline here is the performance of the QD library on a single-core of the Xeon CPU[4]. For each operation presented in the tables, an array with 10 million elements was processed in parallel and careful timing was performed. It is clear from the speed-up presented in the tables that even though the 16-cores of the dual CPUs provide substantial benefit to the overall performance, they still are far outperformed by the 448-cores of the GPU by factors in the range of 3 – 8. *The performance data provides strong evidence of an order-of-magnitude level performance gain from the many-core GPU over a multi-core Xeon CPU on high-precision floating-point computation.*

We next turn to the performance of the CUDA GPU implementation of the high-order and high-precision Teukolsky equation solver code. In Fig. 3 we plot the overall run time of the code as a function of problem size (in units of a basic $1000(r^*) \times 32(\theta)$ grid) for both the quadruple and octal precision cases. The trends are exactly as expected. The run time doubles with the doubling of the problem size[5]. The GPU versions exhibit significantly lower run times (more detail appears below in Fig. 4.) when compared with the single-core CPU case depicted here. In addition, it is clear that doubling the precision (quadruple to octal) increases run time by an order-of-magnitude; no matter what processor architecture is involved.

[4] It is worth pointing out that (surprisingly) the performance of the Intel C++ compiler supported *long double* datatype is much *lower* than that of double-double (DD) from the QD library.

[5] Note a slight variation to this trend in the CPU-DD plot. This is more clearly seen in Fig. 4.

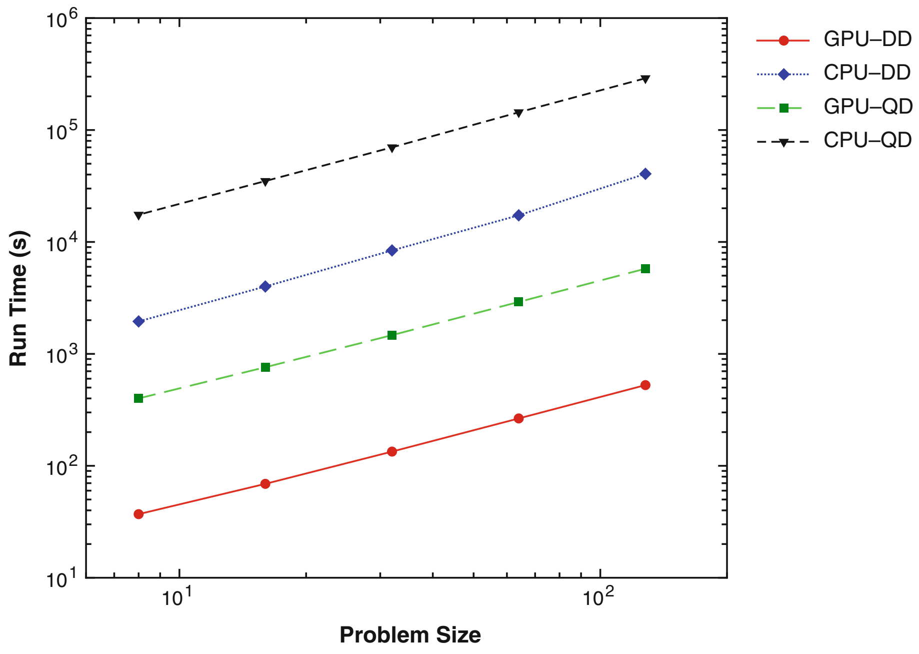

Fig. 3 The overall run time of the Teukolsky solver code as a function of problem size (in units of a basic $1000(r^*) \times 32(\theta)$ grid) for both the quadruple (DD) and octal (QD) precision cases. The GPU cases exhibit significantly lower run times.

In Fig. 4 we present the actual speed-up from the CUDA GPU over a single-core of the Xeon CPU as a function of problem size for both the quadruple and octal precision cases. As the problem size grows, the speed-up the GPU delivers gets even larger. This is simply because the larger problem size allows for a better exploitation of the parallel many-core GPU architecture, thus yielding a higher performance gain in those cases. *For the largest problem size we could fit in the GPU memory, we nearly obtain two orders-of-magnitude gain in performance using the many-core GPU over a single-core of the Xeon CPU*. This is again strongly suggestive of an order-of-magnitude level performance gain from the many-core GPU over a multi-core Xeon CPU that we noted earlier. It is worth noting that the speed-up from the GPU is lower in the octal precision cases, as compared with the quadruple precision cases. We also observe this in Tables 1 and 2. We think that this is because the algorithms for octal precision operations are much more complex than those for quadruple precision – they involve a lot more conditional branching and that typically hurts GPU performance. Finally, it is also worth pointing out that for the quadruple precision case, the GPU performance appears to jump disproportionately to a much higher level at the very last data point i.e. for a grid size of 128000×32. This does not occur in the octal precision context. We currently do not have a clear explanation for this unexpected behavior.

6 Summary & Conclusions

The standard for numerical scientific computation today is double-precision floating-point accuracy (64-bit or ∼15 decimal digits); however, there are a growing number of

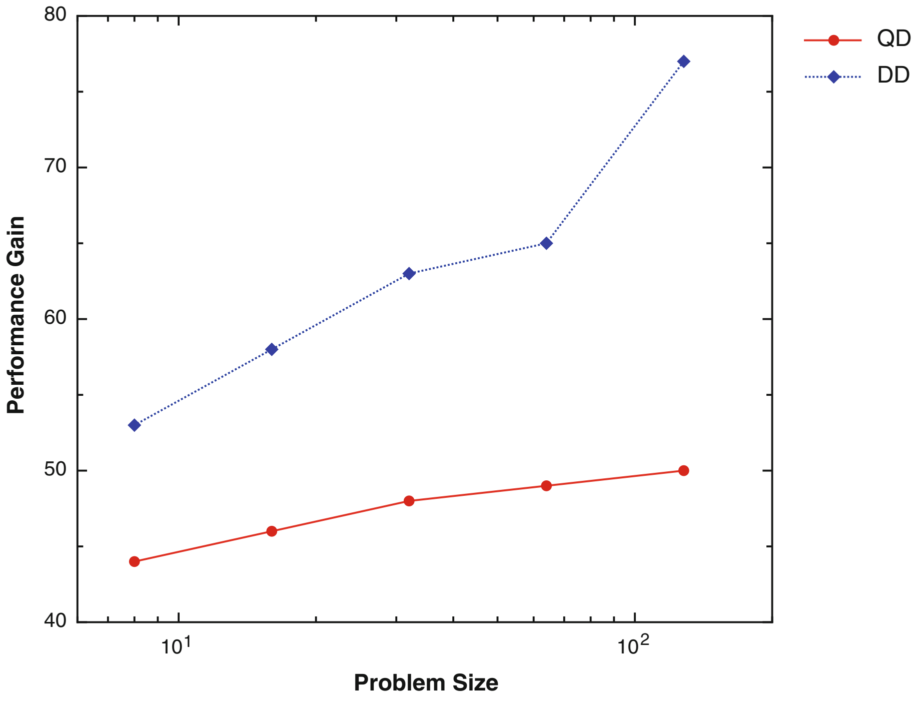

Fig. 4 The speed-up from the Fermi CUDA GPU over a single-core of the Xeon CPU as a function of problem size for both the quadruple (DD) and octal (QD) precision cases.

applications that benefit from a higher level of numerical precision, especially as larger and long-duration simulations become commonplace, and higher-order methods are more widely used. Although the current generation of processors do not support higher than double-precision computation, one can easily emulate such high levels of numerical precision through CPU software libraries, but these often perform orders-of-magnitude slower than the full hardware-supported double-precision equivalents, thus making such an option impractical. Since it is highly unlikely that the computer industry will develop a commodity processor in the near future that natively supports quadruple or higher precision, leveraging the massive parallelism offered by the many-core GPU architecture is a promising interim solution to this major challenge.

In this paper, we describe such an application in the research area of black hole physics: studying the late-time behavior of decaying fields in Kerr black hole space-time. The Teukolsky equation solver code requires high-precision numerical computation because of the rather wide range of amplitudes of the different multipole modes involved. In addition, the high-order of the numerical method and the long evolutions needed, make the application highly demanding computationally. These requirements make performing scientifically meaningful Kerr tails numerical simulations rather challenging, especially using traditional desktop processors. For this reason, in this work we utilize hardware accelerators such as many-core GPUs. We describe the parallelization approach and its detailed implementation in detail in this paper, and then present the final performance outcome. *The final performance data provides strong evidence of an order-of-magnitude level performance gain from the many-core Fermi GPU over a multi-core Xeon CPU for high-precision floating-point computations.*

Such performance gains will allow us to further advance research on the topic of late-time radiative tails in Kerr black hole space-time. We thus demonstrate a practical solution for demanding problems that benefit from high-precision numerics today.

Our performance results suggest that high-precision computations can be accelerated quite well using many-core GPUs, much like double-precision computations. This is an outcome that is quite likely to be generally applicable to other areas of computational science and engineering.

We would like to thank Alfa Heryudono for helpful suggestions and discussion. In addition, we are also grateful to the journal reviewers and handling editor for their feedback that substantially improved the manuscript. We acknowledge research support from NSF Grant Nos. PHY-0902026, CNS-0959382, PHY- 1016906 and PHY-1135664, and AFOSR DURIP Grant No. FA9550-10-1-0354 and also the Massachusetts Space Grant Consortium.

References

1. Bailey, D.H.: High-precision arithmetic in scientific computation. Comput. Sci. Eng. **7**, 54 (2005)
2. Bailey, D.H., Barrio, R., Borwein, J.M.: High-precision computation: Mathematical physics and dynamics. Appl. Math. Comput. **218**, 10106 (2012)
3. Higham, N..: Accuracy and Stability of Numerical Analysis, pp 506–507. SIAM, (1996).
4. Valdettaro, L., Rieutord, M., Braconnier, T., Fraysse, V.: Convergence and round-off errors in a two-dimensional eigenvalue problem using spectral methods and ArnoldiChebyshev algorithm. J. Comput. Appl. Math. **205**, 382 (2007)
5. Sarra, S.: Radial basis function approximation methods with extended precision floating point arithmetic. Eng. Anal. Boundary Elements **35**, 68 (2011)
6. Gottieb, S., Fischer, Paul F.: Modified Conjugate Gradient Method for the Solution of $Ax = b$, J. Sci. Comp. **13**, 173 (1998)
7. Nvidia's CUDA, http://www.nvidia.com/cuda/. Accessed 13 Aug 2012
8. Burko, L., Khanna, G.: Radiative falloff in the background of rotating black hole. Phys. Rev. D **67**, 081502 (2003)
9. Burko, L., Khanna, G.: Universality of massive scalar field late-time tails in black-hole spacetimes. Phys. Rev. D **70**, 044018 (2004)
10. Burko, L., Khanna, G.: Late-time Kerr tails revisited. Class. Quant. Grav. **26**, 015014 (2009)
11. Burko, L., Khanna, G.: Late-time Kerr tails: generic and non-generic initial data sets, "up" modes, and superposition. Class. Quant. Grav. **28**, 025012 (2011)
12. Zenginoglu, A., Khanna, G., and Burko, L., Mode coupling and intermediate behavior of Kerr tails, Class. Quant. Grav., 2012.
13. Baumgarte, T., Shapiro, S.: Numerical Relativity. Cambridge University Press, (2010)
14. Alcubierre, M.: Introduction to 3+1 Numerical Relativity. Oxford University Press, (2008)
15. Cactus Code, http://www.cactuscode.org/. Accessed 13 Aug 2012
16. Gundlach, C.: Critical Phenomena in Gravitational Collapse. Living Rev. Relat. **2**, 4 (1999)
17. Amaro-Seoane, P., et al.: Intermediate and Extreme Mass-Ratio Inspirals - Astrophysics, Science Applications and Detection using LISA. Class. Quant. Grav. **24**, R113 (2007)
18. Teukolsky, S.: Perturbations of a rotating black hole. Astrophys. J. **185**, 635 (1973)
19. Lax, P., Richtmyer, R.: Survey of the stability of linear finite difference equations. Comm. Pure Appl. Math. **9**, 267 (1956)
20. Krivan, W., Laguna, P., Papadopoulos, P., Andersson, N.: Dynamics of perturbations of rotating black holes. Phys. Rev. D **56**, 3395 (1997)
21. Price, R.: Nonspherical Perturbations of Relativistic Gravitational Collapse. Phys. Rev. D **5**, 2419 (1972)
22. Tiglio, M., Kidder, L., Teukolsky, S.: High accuracy simulations of Kerr tails: coordinate dependence and higher multipoles. Class. Quant. Grav. **25**, 105022 (2008)
23. Khanna, G., McKennon, J.: Numerical modeling of gravitational wave sources accelerated by OpenCL. Comput. Phys. Commun. **181**, 1605 (2010)
24. McKennon, J., Forrester, G. and Khanna, G.,: High accuracy gravitational waveforms from black hole binary inspirals using OpenCL. Proceedings of the NSF XSEDE12 Conference, Chicago IL, 2012.
25. LBNL QD Library, http://crd.lbl.gov/~dhbailey/mpdist/. Accessed 13 Aug 2012
26. GQD Library, http://code.google.com/p/gpuprec/. Accessed 13 Aug 2012

27. Ginjupalli, R., Khanna, G.: High-precision numerical simulations of rotating black holes accelerated by CUDA, Proceedings of the International Conference on High Performance Computing Systems (HPCS). Orlando, FL 2010.