A Case Study for Petascale Applications in Astrophysics: Simulating Gamma-Ray Bursts

C. D. Ott
Department of Astronomy and Steward Observatory
The University of Arizona
Tucson, AZ, USA
cott@as.arizona.edu

E. Schnetter
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA, USA
schnetter@cct.lsu.edu

G. Allen
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA, USA
gallen@cct.lsu.edu

E. Seidel
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA, USA
eseidel@cct.lsu.edu

J. Tao
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA, USA
jtao@cct.lsu.edu

B. Zink
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA, USA
bzink@cct.lsu.edu

ABSTRACT

Petascale computing will allow astrophysicists to investigate astrophysical objects, systems, and events that cannot be studied by current observational means and that were previously excluded from computational study by sheer lack of CPU power and appropriate codes. Here we present a pragmatic case study, focussing on the simulation of gamma-ray bursts as a science driver for petascale computing. We estimate the computational requirements for such simulations and delineate in what way petascale and peta-grid computing can be utilized in this context.

1. INTRODUCTION

The past decades have seen a quasi-exponential increase in theoretical peak single CPU performance and an even greater performance increase in the integral (theoretical) peak performance of massively-parallel supercomputers. Soon the first machines will reach the 1-petaflop mark, then being more than 4.2 million times faster than 30 years ago the Cray-1 (peak performance 240 Megaflop/s)[1], arguably the world's first true supercomputer.

Computer software, in particular scientific software and scientist-developed code, is in the main unable to keep up with the exploding computing power. In fact, many physics codes that ran on the Cray-1 thirty years ago can still be found in nearly unaltered form running on modern quasi-scalar desktop CPUs or as modules of large simulation codes running on massively-parallel systems. Such codes typically are not well optimized for modern-day architectures, leading to severe limitations on resource efficiency (sustained flop/s vs. peak flop/s) and parallel scaling.

To profit fully from the extreme hardware performance increase we are witnessing, science codes must be upgraded, optimized and enabled for petascale computing. Only with petascale-enabled codes utilizing the full potential of petascale platforms will it be possible to tackle the next generation of computational complex scientific problems (science drivers).

Hundreds of millions of dollars are being targeted at designing and deploying petascale hardware and developing new programming languages and models, yet there is a lack of understanding of how they will be employed for real-life applications, or rather, real science, applications. In this paper, we describe in some detail the computational needs of a challenging problem in astrophysics – the modeling of gamma-ray bursts (GRBs, introduced in section 2) – which is motivating this group of authors to be ready to exploit petascale computers.

Although solving this GRB simulation problem is likely to be a decade long challenge, requiring advances in physics and algorithms as well as computer science, we believe that continuing to develop our scientific software through the component framework Cactus [14, 11] will provide a path to petascale, and we describe some initial steps that we are taking in this direction.

Much of the discussion in this paper will be in the context of the Cactus Computational Infrastructure [11] and codes that build upon it, including the general-relativistic (GR) spacetime curvature evolution code Ccatie [2, 1, 13], the adaptive mesh-refinement driver Carpet [22, 12], and the GR hydrodynamics code Whisky [5, 26] as well as their combination put to work in the context of massive star collapse [19]. Cactus is used by more than two dozen groups worldwide and is part of the petascale tool development for

[1]See http://en.wikipedia.org/wiki/Cray-1

MG '08, January 29-February 3 2008, Baton Rouge, LA, USA

NSF's Blue Waters petascale facility[2].

In the following we discuss our specific example of the numerical modeling of gamma-ray bursts (GRBs) as a science driver for petascale computing. In the next section we provide a concise scientific motivation of the problem and then go on to discuss the problem's computational complexity in section 3. Section 4 introduces our computational tools as examples for codes that may find application in the GRB context. In section 5 we go on to present strategies to petascale GRB codes, discussing parallel-scaling issues, as well as resource efficiency, (meta-)data handling requirements and technologies, and the possibility for spawning off sub-simulations or parts of simulations via grid technology, combing the computational power of multiple supercomputers or delegating dynamically irrelevant parts of the simulations to distributed computing. We summarize and discuss our concepts and efforts in section 6.

2. A GRB PRIMER

Gamma-Ray Bursts (GRBs) are intense narrowly-beamed flashes of γ-rays (very high-energy photons) of cosmological origin. The riddle concerning their central engines and emission mechanisms is one of the most complex and challenging problems of astrophysics today.

GRBs last between 0.5–1000 secs and have a bimodal distribution of durations [15], indicating two distinct classes of mechanisms and central engines powering the bursts. The short-hard (duration $\lesssim$ 2 secs) group of GRBs (hard, because their γ-ray emissions peak at higher frequency) predominantly occurs in galaxies with old stellar populations at moderate astronomical distances [27, 15]. The energy released in a short-hard GRB and its duration suggest [27, 15] a black hole with a $\sim$0.1 solar-mass ($1\ \mathrm{M}_\odot = 1.99\times10^{33}$ grams) accretion disk as the central engine. Such a BH–accretion-disk system is likely to be formed by the coalescence of NS–NS or NS–BH systems (e.g. [24]).

Long-soft (duration $\sim$2–1000 secs) GRBs on the other hand seem to occur exclusively in galaxies with young stellar populations and low metallicity Observations that have recently become available (see [15] for reviews) indicate features in the x-ray and optical afterglow spectra and luminosity evolutions of long-soft GRBs that show similarities with spectra and light curves obtained from Type-Ib/c core-collapse supernovae whose progenitors are evolved massive stars ($M \gtrsim 25\,\mathrm{M}_\odot$) that have lost their extended hydrogen envelopes and probably also a fair fraction of their helium shell. These observations support the collapsar model ([27], see fig. 1) of long-soft GRBs that envisions a stellar-mass black hole formed in the aftermath of a stellar core-collapse event with a massive ($\sim$1 $\mathrm{M}_\odot$) rotationally-supported accretion disk as the central engine, powering the GRB jet that punches through the stellar envelope reaching ultra-relativistic velocities [15].

Although observations are aiding our theoretical understanding, much that is said about the GRB central engine will remain speculation until it is possible to self-consistently model (i) the processes that lead to the formation of the GRB central engine and (ii) the way the central engine utilizes gravitational (accretion) and rotational energy to launch the GRB jet via magnetic stresses and/or polar neutrino pair-annihilation processes. The physics necessary in such a model includes general relativity, relativistic magneto-hydrodynamics, nuclear physics (describing nuclear reactions and the equation of state of dense matter), neutrino physics (weak interactions), and neutrino and photon radiation transport. In addition, it is necessary to adequately resolve physical processes with characteristic scales from $\sim$25–100 meters near the central engine to $\sim$5–10 million kilometers, the approximate radius of the collapsar progenitor star.

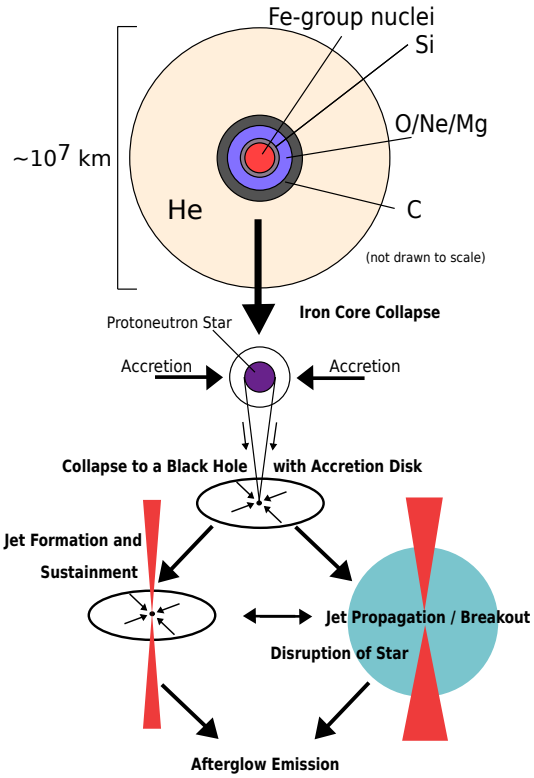

Figure 1: Schematic view of a collapsar-type GRB and its multiple phases. See text for details.

3. GRBS AND PETASCALE: A MULTI-COMPONENT APPROACH

In this case study we aim at a comprehensive approach, considering the computational modeling of a GRB from the onset of its progenitor star's collapse to the break-out of GRB jet from the stellar envelope to the modeling of the GRB afterglow emissions in x-ray, optical and radio bands. We envision a computational scenario in which the entire GRB event is treated by multiple physics codes but within one computational infrastructure, responsible for workflow and data management, coordinating the action of individual physics modules.

Simulating a long-soft GRB means capturing the physics and dynamics occurring over a multitude of spatial and temporal scales with each temporal and spatial scale and physical pro-

[2]http://www.ncsa.uiuc.edu/BlueWaters/

cess having different relevance at different times of the GRB evolution. For example, photon transport in the outer stellar envelope and the outer envelope dynamics are irrelevant in the collapse phase of the stellar core embedded deeply inside the star. On the other hand, neutrino transport, crucially important in the initial collapse phase and at later times near the central engine, is utterly irrelevant for the jet propagation, breakout and GRB afterglow phases, all of which need a detailed photon-transfer treatment.

Hence, quite naturally, the GRB modeling is broken down into multiple components, each component being associated with a specific epoch and/or physical process in the overall GRB evolution and requiring physics that is partly (or completely) different from what is needed in different GRB components.

We identify the following components of the GRB problem, schematically depicted in figure 1, and delineate their main physics inputs and modules. We also estimate roughly and only accurate to an order-of-magnitude their computational requirements in terms of memory and floppage.

3.1 Iron Core Collapse, Supernova Evolution, and Black Hole Formation

In this GRB component the collapse of the stellar iron core and overlying silicon and oxygen layers to a protoneutron star (PNS) are tracked. A supernova shock is launched, but quickly stalls due to energy losses to neutrinos and dissociation of heavy nuclei. Material accretes onto the PNS and eventually leads to its collapse to a black hole that is quickly accreting from a massive accretion disk.

The key physics components are nuclear equation of state (EOS), neutrino physics and transport, GRMHD, and GR curvature evolution. The physical domain of relevance encompasses (in radius) $\sim 10^9$ cm (10,000 km), and scales down to $\sim$25 m (in the black hole (BH)-formation phase) must be resolved to capture the physical dynamics. Assuming a basegrid (= largest grid) spacing of 50 km, 11 levels of mesh refinement (each level yielding a factor of 2 increase in resolution) are required to resolve 25 m scales. Experience from stellar collapse calculations [20, 19] shows that each level of refinement requires about the same number of computational zones, cutting the computational domain covered in half for each refinement level added. The total number of computational zones with all refinement levels active is then $400^3 \times 11 = 640$ million. Assuming parallel execution on 1000 cores, each core will have a load of $\sim 11 \times 40^3$ zones, with 3 inter-process ghost zones (see section 4) on each face, the total number of zones on a core will be $\sim 11 \times 46^3$, yielding a total number of $\sim$1.1 billion zones.

Memory Requirements. The GR curvature evolution part requires typically $\sim$200 3D arrays (grid functions [GFs], including 2 previous time levels for the evolved fields), GRMHD (including the nuclear EOS) also requires $\sim$200 GFs, neutrino radiation transport (with neutrino physics and in the multi-group flux-limited diffusion approximation, see e.g. [10]) requires $\sim$1700 GFs (assuming 32 neutrino energy groups and 3 neutrino flavors). Assuming double precision real numbers, the total memory requirement will be near 18 TByte.

Sustained Performance Requirements.[3] Assuming a time integration step that is half the light-crossing time of each grid cell, the basegrid timestep is 1.7 milliseconds. The physical period to be covered during this GRB stage is $\sim$2 seconds, leaving us with 2400 base grid time steps, $2^{11-1} \times 2400$ finest grid time steps and, summing over all refinement levels (2^{11} - 1) $\times$ 2400 $\sim$5 million total updates of $\sim 460^3$ points. The GR curvature evolution and GRMHD both require each $\sim$20 kflop for the update of a single point, the EOS calls from the GR hydrodynamics and neutrino transport modules require an additional $\sim$10 kflop per single-point update while the radiation transport and neutrino interaction module will consume no less than $\sim$500 kflop per single-point as its time-integration and coupling with matter must be performed time-implicitly, requiring the inversion of large semi-sparse matrices.

The total required floppage is then $\sim$270,000 Pflop. Hence, a supercomputer on which the GRB simulation could run at 1 Pflop/s sustained performance would require 3 days, and on a 1 Pflop/s peak performance machine, this part of the GRB evolution would require $\sim$30 days of wall time.

3.2 Accretion Dynamics, Jet Formation, Jet Sustainment

This GRB simulation component tracks the long-term (1 s – $\gtrsim$200 s) accretion dynamics onto the black hole and captures the initial formation and the sustainment of the MHD and/or neutrino-driven GRB jet. Scales on the order of $\sim$25 m near the BH horizon must be resolved and a physical domain of $\sim$1000 km^3 must be covered. Critical physics components are GR curvature evolution, GRMHD, and neutrino radiation transport. The entire post-BH formation epoch of the GRB, up to times on the order of $\sim$200 s in physical time after BH formation must be covered providing continuous input to the jet propagation component of the GRB simulation discussed in the next subsection.

Memory Requirements. Using a 10-level AMR hierarchy with a 200^3 points on each level and distributing the grid over 1000 CPU cores yields a local load of 10×26^3 zones (including 3 inter-process ghost zones on each face). Assuming the same number of GFs as used in the first stage of the GRB simulation, the total memory requirement will be $\sim$3 TByte.

Sustained Performance Requirements. Evolving the AMR hierarchy for 200 s in physical time corresponds to 6×10^5 base grid timesteps and a total of 6×10^8 updates of a set of 260^3 grid points. This maps to a total floppage of 6 million Pflop, requiring a total of $\sim$70 days on a supercomputer on which this GRB simulation component runs at sustained 1 Pflop/s.

3.3 Jet Propagation and Break Out

This component follows the jet propagation through the low-density polar regions of the stellar envelope. This and the previous simulation component coincide in time and bound-

[3]The performance requirements quoted here are based on actual measurements (with extrapolation) using the Ccatie spacetime evolution code and the GR hydrodynamics package Whisky; see http://www.cactuscode.org/Benchmarks/.

ary information for the energy input into the jet is required. High-resolution tracking of the propagating GRB fireball via AMR techniques is crucial in this epoch and angle-dependent photon transport will be necessary to track the jet's traversal from the optically-thick stellar envelope to the optically thin regime at jet breakout. Neutrino transport and GR curvature evolution are not required at this stage and are replaced by photon transport whose computational complexity is in the same ballpark as that of neutrino transport. In addition to tracking the jet propagation, this simulation component will receive boundary hydrodynamics data from the accretion dynamics and jet sustainment component and will capture the spatial widening of the explosion that will eventually tear the star apart.

Tracer-particle trajectories may be saved and later fed into specialized and computationally-intensive chemical element synthesis (nucleosynthesis) codes that require only a subset of the hydrodynamic data. Alternatively, small independent nucleosynthesis jobs taking data directly from the GRB simulation may be spawned-off via the grid (see also section 5.3).

Using a 15-level AMR hierarchy to cover scales from $\sim$1 km and extend out to 10 million km with a base grid spacing of 25000 km, we have (using 1000 CPU cores) a total of 15×460^3 grid points in this part of the GRB simulation that must be tracked for $\sim$200 s after BH formation in conjunction with the GRB simulation component discussed in the previous section.

Memory Requirements. Assuming a comparable number of GFs to previous phases and with the increased number of refinement levels leading to a total number of 1.5 billion grid points, we estimate a memory requirement of $\sim$25 TByte.

Sustained Performance Requirements. 200 s in physical time must be covered. This converts to $\sim$170 coarse grid timesteps and a total number of 5.5 million updates of 460^3 points. Adopting a similar computational complexity to the other phases (here dominated by photon transport) of $\sim$500 kflop per grid point update, we obtain a total floppage of $\sim$300,000 Pflop. On a machine with 1 Pflop/s sustained this corresponds to $\sim$3.5 days of continuous computation. On a machine with 1 Pflop/s peak performance and 0.1 Pflop/s sustained, the computation would require 30 days of wall time, assuming one can achieve 10% of the peak performance. Note, however, that because of physical coincidence, this and the previously discussed component must be run simultaneously, but could, possibly, be run on separate machine as the amount of communication necessary is limited to the exchange of boundary information.

3.4 Late-time, post GRB evolution: Afterglow

This phase sets in after the GRB central engined has stopped operating, the powerful GRB jet has died away, and the entire star has been disrupted, leaving behind the central black hole and many solar masses of debris material that is ejected at varying velocities. While the hydrodynamics of the ejecta is moderately simple and does not require the full GRMHD treatment, photon transport and photon absorption/re-emission by debris material as well as photon creation in nuclear decay are complicated aspects of this late-time phase in the GRB evolution. A detailed, 3D photon transport scheme is necessary and dominates memory requirements and computational complexity. Moreover, the afterglow evolution must – in principle – be followed for months of physical time, making this a formidable task not even accomplishable in all details with petascale supercomputers. A typical and well suited approximation is the adoption of the Monte-Carlo method for radiation transport in which the scattering, emission/absorption random-walks of test particles are followed (e.g.,[7]). This approach scales very well to large computers but can be memory intensive. Detailed memory and performance requirements depend strongly on the level of approximation and on how well photon emission and absorption lines are resolved. The authors cannot claim expertise in this particular aspect of the GRB, hence cannot make reliable estimates for its computational needs.

Sections 3.1–3.4 show that performing a simulation of even a single phase of the GRB phenomenon requires petascale computing power. The estimates that we have made on memory and sustained performance requirements are optimistic and they may in fact be underestimating the real computational cost of such simulations as on-line analysis routines and I/O are not included in our order-of-magnitude estimates and may contribute significantly to the overall cost.

We are also optimistic in terms of performance – the first petascale machines are unlikely to provide 1 Pflop/s sustained performance and the resource efficiency of GRB codes may be worse than our optimistic estimate of $\sim$10% of peak performance. In particular resource efficiency and parallel scaling must be optimized in present codes. We discuss possible paths to peta-enable present codes in section 5 below.

Finally, we point out that any long-term simulation tends to amplify the small numerical errors acquired in each iteration by accumulation, sometimes to significant magnitudes (say, 10% or more). A conservative estimate of the accumulated error can be obtained from a linear growth model, although the actual time dependence may be sub-linear or super-linear depending on the quantity and the physical model. In the finite-difference and finite-volume schemes, the errors obtained from each iteration directy depend on the resolution, so, given an error limit, there is a direct relation between the total physical simulation time and the resolution required in the computational domain - more directly, in the GRB problem, a simulation time of 200 seconds may well require a much higher resolution than a simulation time of 2 seconds.

A linear growth model estimates the total error after physical time t to $E(t) = E_0 \frac{t}{t_0} \left(\frac{h}{h_0}\right)^\alpha$, where $E(t)$ is the error after the physical time t, h is the spatial resolution in one direction, and E_0 is the growth rate, i.e., the error accumulated in time t_0 at resolution h_0. The number α determines how quickly the error diminishes with resolution, and is also called the *order of convergence* of the numerical scheme. From this formula, we can obtain estimates for the maximal error growth rates given a physical time by solving for E_0: $E_0 = \frac{E(t)t_0}{t} \left(\frac{h_0}{h}\right)^\alpha$. As reference time to measure the growth rate we will use $100\mu s$, which is typical timescale for neutron stars. Therefore, for a maximum error

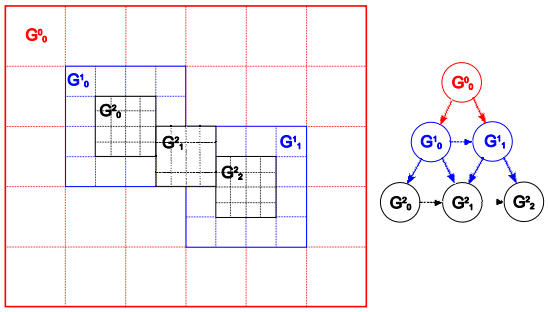

Figure 2: A sample $2D$ grid hierarchy with 3 refinement levels and a refinement factor of 2. *Proper nesting* (see main text) requires that the fine grids are wholly contained in coarser grids.

of $E(t) = 10\%$ and a physical time of 2 seconds, the scheme must not exceed a relative error accumulation of $5 \cdot 10^{-6}$ per $100\mu s$ in all relevant quantities, and for 200 seconds the limit is accordingly $5 \cdot 10^{-8}$. The error can be reduced by (i) using an adapted grid (i.e., reducing E_0) [29], (ii) using more cells (i.e., reducing h) or (iii) using high-order schemes (i.e., increasing α).

4. COMPUTATIONAL INFRASTRUCTURE

In this section we discuss the basics of our computational approach to the GRB problem. We introduce adaptive-mesh refinement (AMR), the central component of the GRB simulations discussed here. AMR allows for the computationally efficient resolution of dynamics on many length scales. Furthermore, we introduce our computational framework Cactus and a subset of the physics codes necessary in the GRB simulations.

4.1 3D Cartesian Meshes with Adaptive Refinement

The block-structured AMR method, introduced by Berger and Oliger (B&O) in 1984 [6] to solve hyperbolic PDEs, is one of the most widely used AMR algorithms. It is built upon one or more nested grid hierarchies of rectangular grids with increasing resolution. The B&O AMR algorithm is adaptive in both space and time, making it very efficient.

The B&O algorithm starts from a given base grid. Beginning with the data on the base grid, some specified criteria estimates the discretization error and flags those grid points requiring higher accuracy. The flagged points are clustered into rectangular blocks, and new grids with higher resolution are generated to cover those blocks. This regridding procedure continues recursively until the criteria are met or the maximum number of refinement levels is reached. The *proper nesting* property of the B&O algorithm requires that the grids on a refinement level must reside inside the region covered by the grids on the next coarser level, though it is possible for a grid to reside in more than one grid on the next coarser level. Figure 2 shows an example hierarchy, where G_1^2 overlaps indeed with both G_0^1 and G_1^1. This whole regridding procedure is repeated every so often to keep the grid structure adapted to the solution's resolution requirements.

A constant ratio between the time step Δt and grid spacing Δx of different levels is often used to simplify scheduling the time integration of different levels. Such a constant ratio follows also naturally from the Courant-Friedrichs-Levy criterion for hyperbolic evolution equations. In this case, all the grids are guaranteed to arrive at the same time at every base time step.

4.2 Computational Framework and Components

The Cactus Framework [14, 11] was developed to reduce the development time for creating simulation codes, to shield developers from changes in system properties, and to enable large-scale science collaborations. Cactus and other modular, component-based frameworks allow scientists and engineers to develop their own application modules and assemble them with a body of existing components, creating applications that can solve complex multi-physics computational problems. Cactus runs on a wide range of hardware ranging from desktop PC's to the largest supercomputers and grid environments.

Carpet [22, 12] is a layer which implements Berger–Oliger mesh refinement (see above). Carpet refines parts of the simulation domain in space and/or time, where each refined region is a block-structured regular grid, allowing for efficient internal representations e.g. as Fortran arrays. In addition to mesh refinement, Carpet also provides parallelism and load distribution by distributing grid functions onto processors. Our finite differencing stencils require an overlap of 3 *ghost zones* between neighboring processors' subdomains.

Our spacetime evolution component Ccatie uses a variant of the BSSN formulation of the Einstein equations [2, 1, 25, 13]. These are a set of 25 coupled hyperbolic partial differential equations which we discretize with high-order finite differences and evolve in time using explicit integrators.

The Whisky code [5, 26] is a GR hydrodynamics code. While Ccatie evolves the curvature part of spacetime, Whisky evolves the "right-hand side", the matter part of the Einstein equations. Whisky computes the update terms for the hydrodynamic variables via a flux-conservative finite-volume method, exploiting the characteristic structure of the equations of GR hydrodynamics.

Combined, these four existing implementations provide a state-of-the-art general relativistic hydrodynamics code that can handle large spatial and temporal scale differences. Together with a radiation transport module (under development) and a component of Whisky that can handle magnetic fields (under development), these form the basis of our proposed petascale application.

5. DEVELOPING FOR PETASCALE

Petascale architectures will differ from current terascale architectures in several important respects. Current code architectures will need to be updated to cope with the challenges that these new hardware architectures pose.

5.1 Hybrid Communication Schemes

Within the next few years, systems are projected to have between 16 or 64 cores (or more) per node. Such multiprocessor systems are both a blessing and a burden. Combining multiple cores into one node reduces the cost for memory and network interfaces, and it reduces fragmentation of the total memory. On the other hand, it increases the already large pressure onto the memory and network systems. It is therefore necessary to orchestrate memory and network usage within a node very carefully to avoid bottlenecks.

Virtually all modern high-performance codes are designed for distributed memory systems, based on MPI [16]. Since MPI can handle communication between nodes, it is natural to use MPI also to parallelize the code within a node. This is certainly the easiest approach from a code architecture point of view, and the resulting memory fragmentation within the node can help efficiency. However, this approach does not scale beyond a few cores per node for several reasons:

Memory per core. Because it is cheaper to just add cores to a node than to also increase the memory per node, and because TOP500 pressure asks for cores, not memory, the amount of memory per core will likely decrease in the future. Fragmenting a node's memory also means that each core can handle only a very small part of the overall problem, which increases the MPI overhead and eventually leads to loss of scaling.

MPI overhead. If there are many cores per node, then using multi-threading instead of MPI can actually remove existing intra-node MPI overhead, increasing the performance. Such overhead consists e.g. of the memory required to store ghost zones for domain-decomposed grid functions, which is significant for higher order methods and physical systems with many independent variables.

Total number of cores. The total number of cores in petascale systems will be orders of magnitude larger than on current systems.[4] It is very difficult to make MPI codes scale to such numbers; each incremental increase in scalability needs to be bought through a dedicated development effort, and most codes will have to cover a factor of 100 or 1000 in scalability. Reducing the number of MPI processes by using multi-threading seems a very viable approach.

Presently, we find that our codes work best with at least 1 GByte of memory per core. Given that each process also requires memory for the executable, static variables, I/O buffers etc., and that the MPI parallelization overhead increases when the problem size per core decreases, this makes it very difficult to use low-per-core memory machines.

We performed tests on Abe (one of our main production platforms) at the National Center for Supercomputing Applications (NCSA) with a model equation, comparing straight MPI parallelization vs. hybrid parallelization combining MPI and OpenMP [18]. We used the Cactus framework [11], the Carpet mesh refinement driver [22, 12], and solved a scalar wave equation in the unit cube as model problem. Figure 3 shows results, comparing the efficient unigrid MPI solver PUGH and the mesh refinement solver Carpet.

[4]NSF assumes $O(1,000,000)$ processors per system; see http://www.nsf.gov/pubs/2007/nsf07559/nsf07559.htm.

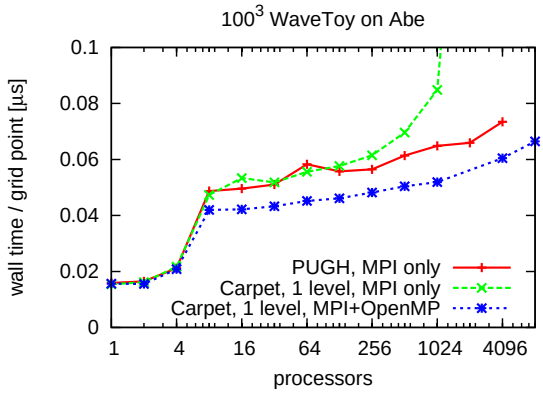

Figure 3: Benchmark results from Abe at NCSA, comparing straight MPI parallelization with a hybrid approach combining MPI and OpenMP. The graphs show the time per grid point vs. the number of cores used. The hybrid solver Carpet/MPI+OpenMP is faster and scales further than Carpet/MPI. For comparison, results from the unigrid solver PUGH are also shown. The efficiency loss between using 4 and 8 cores is likely caused by memory contention within a node; Abe has 8 cores per node. For additional details see the main text.

This figure includes three graphs showing the time per grid point vs. the overall number of cores. As expected, the unigrid solver PUGH scales well; this solver is very mature and has been optimized for a wide range of architectures. The mesh refinement solver Carpet/MPI ceases to scale at about 1024 cores. This is due to increasing communication and administrative overhead as the number of domains increases. (Understanding and improving this is the topic of ongoing research.) However, the mesh refinement solver Carpet/MPI+OpenMP scales up to at least 8192 cores, and it is also about 20% faster when 8 or more cores are used. These improvements are due to two different factors. The efficiency gain within a single node is caused by reducing memory contention via additional cache optimizations. Such optimizations can be more powerful when OpenMP is used, since then the application's memory space within a node is not fragmented as with pure MPI. The non-scaling of Carpet/MPI for large numbers of nodes is remedied in Carpet/MPI+OpenMP since there are fewer MPI processes, and the application's administrative overhead is smaller.

5.2 Future Hybrid Accelerated Architectures

Future multi-core architectures may also be supplemented by computational accelerators like the NVIDIA Tesla and the AMD FireStream solutions. These are multiprocessors evolved from the traditional graphics accelerators, now called *graphics processing units* or *GPUs*, which achieve high computational throughput by parallel computation on a large number of stream processors. Current products like the Tesla GPUs achieve a peak performance of about 350 GFlop/s, though this is limited to single-precision floating point operations. The FireStream GPUs and future NVIDIA products will support double-precision operations as well, and are therefore an interesting target for scientific computation.

Taking advantage of these GPUs poses a major challenge, however. These architectures do currently not admit to port standard OpenMP/MPI code in an efficient manner, which is partly related to the fact that several stream processors share instruction units, and partly to the requirement of implementing manual caching schemes for accessing the frame buffer memory. In fact, programming interfaces similar to standard languages like C have only become available recently (e.g. NVIDIA CUDA), replacing earlier attempts to use the graphics API (shaders) for general purpose computing.

One of the authors has recently demonstrated that a general relativistic evolution code can be accelerated substantially by using an NVIDIA Quadro FX 5600 GPU [28]. Compared to a serial code running on an AMD Opteron 2.4 GHz, the parallel GPU code using CUDA can run up to 26.5 times faster. While the actual acceleration can vary significantly with the particular problem, this shows that future CPU/GPU hybrid architectures can potentially increase the performance of scientific codes to the Petascale range *even in medium-sized clusters* (assuming a double-precision peak performance of about 2 TFlop/s in GPUs within the next 3 years, a petaflop machine can be built from only 500 GPUs).

A likely scenario is that highly parallel problems, like calculating the right hand side of a system of partial differential equations, are off-loaded to a GPU kernel, while the CPU asynchronously takes care of inter-GPU/inter-node communication, analysis and output. This, of course, requires an appropriate computational infrastructure and standard abstractions to achieve portability for the kernel codes between different GPUs (or CPUs). Therefore, we expect this aspect of software architecture and computational middleware to be an important facet of future petascale computation.

5.3 Multi-Machine Simulations

Another approach towards petascale computing is to take advantage of the combined power of multiple geographically-distributed machines. Numerical relativity has long been a driving application for Grid computing [4], and here we describe several scenarios where Grid technologies could contribute to petascale-enabling the GRB problem.

Distributed "metacomputing" simulations. In this paper we describe a science driver requiring petascale capabilities. Although planned, such machines will not be available in the short term, and even when available it may be hard to schedule the use of whole machines. An alternative approach is to combine several large more available machines into a single "virtual" petascale computer which is then used for a tightly coupled GRB simulation [17, 23, 9].[5]

Coupled models. Multi-machine simulations can also be a good way to handle multi-model simulations, e.g. in order to calculate the steps described in sections 3.2 and 3.3 at the same time. Different physics models will exhibit very different execution characteristics, and may well best be executed on different architectures – e.g., one on a large shared memory machine such as an SGI Altix, the other on a cycle powerhouse such as the BlueGene.

Task Spawning. In section 3.3 we described how part of the simulation process could be calculated off-line. This can be achieved by automated task spawning – moving a part of an application to an appropriate remote resource, while the core simulation continues. This approach can also be used to spawn general analysis tasks which are carried out frequently (potentially at every iteration) during a simulation, for example, to locate a black hole horizon or compute the emitted gravitational waves. These time consuming tasks do not feed back to the main simulation, or may not be easily parallelized, and hence can be spawned to a more appropriate resource, allowing the primary resource to fully concentrate on its task of advancing the main simulation, improving efficiency and throughput. Spawning scenarios were demonstrated at SC01, where a black hole simulation running in Germany was able to automatically spawn analysis tasks to resources in Europe, Asia, and North America [3, 21].

5.4 Data and Metadata

The GRB simulations will generate large amounts of output data. Checkpoint/Restart files of size ∼20 TByte will be periodically generated and potentially transported to alternative machines for restart. Total output of order 5 PByte per simulation (assuming output every 10 basegrid iterations) will be generated containing data which must be analyzed and visualized (often by several different members of a research group) for physical results, and archived for later use. Each simulation can generate hundreds of files with different file formats, and the discovery and manipulation of these files is complicated by the fact that users are running on different machines, with different filesystems, quotas, and archiving possibilities. For multi-machine Grid scenarios, the data management problem is exacerbated; users will not necessarily even know on which machine their simulation is running, or the simulation could even be moving between resources leaving data in multiple locations.

Scientific results also need to be validated and cross-checked, often requiring several simulations at different resolutions for every data point. A single physics paper in numerical relativity can currently take many months of dedicated use of a TOP500-class machine, and the access to and management of the data becomes crucially important. A scientific group must be able to defend its results not only to the paper referee, but also months and years after publication. One approach would be to simply store source code and rerun simulations as required, however this in turn leads to challenges in providing a complete description of the run including exact machine architecture, operating system, compilers, compiler and OS configuration options etc as well as simulation input files — with no guarantee that it will be possible to replicate the run on contemporary hardware.

These needs necessitate advances in efficient parallel file I/O, automated management of data including metadata generation, and fast networks for data transport, as well as strategies for long term data archival.

Metadata are central to the management, archival and re-

[5]Such a simulation using the Cactus framework won the 2001 Gordon Bell prize for this; see http://www.cactuscode.org/News/Archives2001/GordonBell2001/.

trieval of simulation data. In the metadata model currently under development for Cactus, we define for each simulation extendable alphanumeric and machine-parsable key–value sets of *core* and *key* metadata: *Core* metadata describe the basic and generic characteristics of the simulation, including all parameter settings, information on active Cactus thorns, size of the simulation, machine and performance information, source code tags, etc. *Key* metadata, on the other hand, consist of application-specific key–value sets, for example, for a GRB simulation, may contain a short description of the progenitor star characteristics (mass/rotational configuration etc), information on when the black hole horizon first appeared or when the GRB jet broke through the stellar envelope. Each key and core metadata key–value pair may be classified static (cannot change during the simulation) or dynamic (can change during the simulation). In this way, the simulation metadata can in addition be used easily for on-line remote monitoring of central characteristics while the simulation is running [8].

Cactus binaries that include metadata management contain their source code and build information. Once invoked, Cactus writes its complete source tree in compressed archive format, as well as static key and core metadata into the output directory. It also announces the starting simulation and transfers metadata (including a source code fingerprint) to a metadata server. Dynamic metadata are updated periodically for simulation monitoring. Unique identifiers are included in output file headers, allowing connecting it to the stored metadata. Upon simulation termination, the output data are transferred to a mass storage system and their detailed location is announced to the metadata server.

6. CONCLUSIONS

We have described above what we take to be the current trend of technology, how we intend to take advantage of it, and what pitfalls it will present. We have been somewhat conservative in our assumptions. Are more radical changes possible? Certainly. For example, a switch to many-core architectures with disjoint, small memories, such as IBM's Cell architecture, could become reality, much to the horror of application developers used to current concepts. Is it likely? We hope not, unless there is a viable programming model, which is, in our opinion, not in sight for the majority of applications.

The old way to describe performance is in terms of "flop", on which the TOP500 and all current supercomputer allocations are based. A newer terminology includes terms like "memory latency", "cache access", "communication bandwidth", etc., which are now used by programmers, but not yet in accounting. Recently the expression "power consumption" came into play, leading to multi-core architectures. What terms could a radically new architecture introduce? Will it be "MTBF", the mean time between failures? Or "data mobility", the ease with which data can be moved to a different device? Or maybe "elasticity", the ability of hardware to reconfigure itself to changing application needs or parital system failures?

We have outlined some of the effects that coming petascale computing environment will have on computational science and on science applications. We have presented a case study of a particular petascale computing problem. Using the simulation of gamma-ray bursts as a science driver, we have examined how petascale computing can help solve problems that are currently computationally impossible. Our computational scenario treats GRBs by multiple physics codes which are embedded within one computational infrastructure. We have also briefly described the computational algorithms for these individual codes, and have given order-of-magnitude estimates for CPU and memory requirements.

Our estimates show that this kind of new (astro-)physics is indeed possible on future petascale architectures. Petascale computing will allow astrophysicists, but, of course, also scientists from other disciplines, to study and solve problems that are impossible to access and solve by any other means.

7. ACKNOWLEDGMENTS

We thank the organisers of the 15th Mardi Gras Conference in Baton Rouge. This work used the resources of the machine Abe at the NCSA under the LRAC allocation MCA02N014. C.D.O. acknowledges support through a Joint Institute for Nuclear Astrophysics postdoctoral fellowship, sub-award no. 61-5292UA of NFS award no. 86-6004791.

8. REFERENCES

[1] M. Alcubierre, B. Brügmann, P. Diener, M. Koppitz, D. Pollney, E. Seidel, and R. Takahashi. Gauge conditions for long-term numerical black hole evolutions without excision. *Phys. Rev. D*, 67:084023, 2003.

[2] M. Alcubierre, B. Brügmann, T. Dramlitsch, J. A. Font, P. Papadopoulos, E. Seidel, N. Stergioulas, and R. Takahashi. Towards a stable numerical evolution of strongly gravitating systems in general relativity: The conformal treatments. *Phys. Rev. D*, 62:044034, 2000.

[3] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *Proceedings of Supercomputing 2001*, Denver, USA, 2001. http://www.cactuscode.org/Articles/Cactus_Allen01e.pre.pdf.

[4] G. Allen and E. Seidel. *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, chapter Collaborative Science: Astrophysics Requirements and Experiences, pages 201–213. Morgan Kaufmann, 2004.

[5] L. Baiotti, I. Hawke, P. J. Montero, F. Löffler, L. Rezzolla, N. Stergioulas, J. A. Font, and E. Seidel. Three-dimensional relativistic simulations of rotating neutron star collapse to a Kerr black hole. *Phys. Rev. D*, 71:024035, 2005.

[6] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.

[7] C. Bernes. A Monte Carlo approach to non-LTE radiative transfer problems. *Astron. Astrophys.*, 73:67, 1979.

[8] R. Bondarescu, G. Allen, G. Daues, I. Kelley, M. Russell, E. Seidel, J. Shalf, and M. Tobias. The astrophysics simulation collaboratory portal: a framework for effective distributed research. *Future Generation Computer Systems*, 2003. Accepted.

[9] R. Bondarescu, G. Allen, G. Daues, I. Kelley, M. Russell, E. Seidel, J. Shalf, and M. Tobias. The Astrophysics Simulation Collaboratory portal: a framework for effective distributed research. *Future Generation Computer Systems*, 21:259–270, 2005.

[10] S. W. Bruenn, C. J. Dirk, A. Mezzacappa, J. C. Hayes, J. M. Blondin, W. R. Hix, and O. E. B. Messer. Modeling core collapse supernovae in 2 and 3 dimensions with spectral neutrino transport. *arXiv:0709.0537 [astro-ph]*, 2007.

[11] Cactus Computational Toolkit home page, http://www.cactuscode.org/.

[12] Mesh Refinement with Carpet, http://www.carpetcode.org/.

[13] Spacetime evolution with CCATIE, http://numrel.aei.mpg.de/Research/codes.html.

[14] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[15] P. Mészáros. Gamma-ray bursts. *Reports of Progress in Physics*, 69:2259, 2006.

[16] MPI: Message Passing Interface Forum, http://www.mpi-forum.org/.

[17] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, and T. Goodale. Scientific Application Performance on Candidate PetaScale Platforms. In *International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, Ca., March 24-30 2007. Winner Best Paper.

[18] OpenMP: Simple, Portable, Scalable SMP Programming, http://www.openmp.org/.

[19] C. D. Ott. *Stellar Iron Core Collapse in 3+1 General Relativity and The Gravitational Wave Signature of Core-Collapse Supernovae*. PhD thesis, Universität Potsdam, Potsdam, Germany, 2006.

[20] C. D. Ott, H. Dimmelmeier, A. Marek, H. T. Janka, I. Hawke, B. Zink, and E. Schnetter. 3D Collapse of Rotating Stellar Iron Cores in General Relativity including Deleptonization and a Nuclear Equation of State. *Phys. Rev. Lett.*, 98:261101, 2007.

[21] M. Ripeanu, A. Iamnitchi, and I. Foster. Performance predictions for a numerical relativity package in grid environment. In *International Journal of High Performance Computing Applications*, volume 15, pages 375–387. Sage Publications, 2001. http://people.cs.uchicago.edu/~matei/PAPERS/.

[22] E. Schnetter, S. H. Hawley, and I. Hawke. Evolutions in 3D numerical relativity using fixed mesh refinement. *Class. Quantum Grav.*, 21:1465–1488, 2004.

[23] E. Schnetter, C. D. Ott, G. Allen, P. Diener, T. Goodale, T. Radke, E. Seidel, and J. Shalf. Cactus Framework: Black holes to gamma ray bursts. In D. A. Bader, editor, *Petascale Computing: Algorithms and Applications*, chapter 24. Chapman & Hall/CRC Computational Science Series, 2007.

[24] S. Setiawan, M. Ruffert, and H.-T. Janka. Three-dimensional simulations of non-stationary accretion by remnant black holes of compact object mergers. *Astron. Astrophys.*, 458:553–567, 2006.

[25] J. van Meter, J. G. Baker, M. Koppitz, and D.-I. Choi. How to move a black hole without excision: gauge conditions for the numerical evolution of a moving puncture. *Phys. Rev. D*, 73:124011, 2006.

[26] Whisky, EU Network GR Hydrodynamics Code, http://www.whiskycode.org/.

[27] S. E. Woosley and J. S. Bloom. The Supernova Gamma-Ray Burst Connection. *Ann. Rev. Astron. Astrophys.*, 44:507, 2006.

[28] B. Zink. A general relativistic evolution code on cuda architectures. (In preparation), 2008.

[29] B. Zink, E. Schnetter, and M. Tiglio. Multi-patch methods in general relativistic astrophysics – i. hydrodynamical flows on fixed backgrounds. arXiv:0712.0353, 2007.