Performance of a Block Structured, Hierarchical Adaptive Mesh Refinement Code on the 64k Node IBM BlueGene/L Computer*

Jeffrey A. Greenough, Bronis R. de Supinski, Robert K. Yates

Lawrence Livermore National Laboratory
Livermore, CA 94550

Charles A. Rendleman, David Skinner, Vince Beckner, Mike Lijewski, and John Bell

Lawrence Berkeley National Laboratory
Berkeley, CA 94720

James C. Sexton

IBM, T.J. Watson Research Center
Yorktown Heights, NY 10598

April 25, 2005

Abstract

We describe the performance of the block-structured Adaptive Mesh Refinement (AMR) code Raptor on the 32k node IBM BlueGene/L computer. This machine represents a significant step forward towards petascale computing. As such, it presents Raptor with many challenges for utilizing the hardware efficiently. In terms of performance, Raptor shows excellent weak and strong scaling when running in single level mode (no adaptivity). Hardware performance monitors show Raptor achieves an aggregate performance of 3.0 Tflops in the main integration kernel on the 32k system. Results from preliminary AMR runs on a prototype astrophysical problem demonstrate the efficiency of the current software when running at large scale. The BG/L system is enabling a physics problem to be considered that represents a factor of 64 increase in overall size compared to the largest ones of this type computed to date. Finally, we provide a description of the development work currently underway to address our inefficiencies.

1 Application and Software Overview

Raptor is a multi-physics Adaptive Mesh Refinement (AMR) code being developed at Lawrence Livermore National Laboratory. It can simulate physical systems in such diverse fields as astrophysics and Inertial Confinement Fusion (ICF). Raptor has a fairly complete set of physics capabilities as required by these applications. These include gray diffusion radiation, electron

*This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48 and the Applied Mathematics Program of the DOE Office of Mathematics, Information, and Computational Sciences under the U.S. Department of Energy under contract No. DE-AC03-76SF00098.

conduction and multifluid hydrodynamics. Raptor can also simulate a wide variety of materials by either using analytic or tabular equation-of-state.

The current work on BlueGene/L (BG/L) at LLNL is focused on multifluid hydrodynamics. The governing equations, the Euler equations, are solved using a higher-order Godunov method that has evolved from and is based on the original method of Colella [5] and later extended to multiple materials [6]. The radiation and conduction algorithms require the use of an ancillary software package, HYPRE [7], for solving implicit equations using an iterative multigrid method. At present this package's port to BG/L is not yet completed. Raptor has been proven to provide high-fidelity simulations of shock accelerated interfaces by comparison with experimental data [9, 13, 18].

Raptor is a hybrid C++/`Fortran` code that uses software infrastructure developed and maintained by the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory [8, 17]. The base library, known as BoxLib, provides C++ classes and data containers for representing block-structured data and software for distributing and exchanging data on parallel computers using MPI. An Additional library, AmrLib, implemented in BoxLib, supports AMR methods on block-structured data. The algorithms contained in this library are much the same as those described in the original AMR papers [4, 3]. Rendleman, et al. describe how AMR methods are extended to parallel computations within the BoxLib framework [17]. These libraries, in addition to the tasks of controlling the details of a calculation (number of levels of refinement, how many steps to take, parallel I/O, etc.), implement the main operations found in an AMR method. These are communicating among the grids at a particular level of refinement (intra-level communications) and communications between different levels of refinement (inter-level communications). The former is handled by copying data from one block to another using an efficient point-to-point protocol while the later is handled by interpolation and averaging operations between data at different spatial resolutions and temporal locations.

The remainder of the Raptor code consists of an applications layer containing the physics classes defined in terms of virtual functions within the AmrLib class hierarchy. Data blocks are managed in C++, in which ghost cells are filled and temporary storage is dynamically allocated, so that when the calls to physics algorithms (usually finite difference methods implemented in `Fortran`) are made, the same stencil can be used for all points and no additional special treatments are required. By structuring the software in this way, the high-level objects that encapsulate the functionality for parallelization and AMR, are independent of the details of the physics algorithms. This makes it easy to replace or add physics modules as long as they adhere to the AmrLib interface requirements.

The remainder of this paper, following an overview of the system architecture of the BG/L system in Section 2, contains a description of the basic AMR methodology in Section 3, and its parallel implementation in Section 4. A set of results in Section 5 describe the weak and strong scaling behavior of the software on BG/L for single level and adaptive simulations, as well as the floating point performance of the numerical kernels. Section 6 describes the physics problem to be addressed on the machine and, finally, Section 7 discusses changes underway to support AMR simulations on very large scale problems.

2 The BlueGene/L Computational Science Platform

BlueGene/L (BG/L) is a massively-parallel computing system for research and development in computational sciences that was designed and implemented by a partnership between Lawrence Livermore National Laboratory (LLNL) and IBM. Its extremely high compute-density design results in a very high cost-performance system with comparatively modest power and cooling requirements. We briefly describe the system architecture [21], focusing on features that impact Raptor.

At this writing the fastest computer in the world, based on 135 Tflops achieved on the

Linpack benchmark, is a 32,768-node BG/L system installed at LLNL. A compute node of BG/L is composed of only 10 chips: its 700 MHz compute ASIC plus 9 DRAM main memory chips. The compute node ASICs include all networking and processor functionality. Each compute ASIC includes two 32-bit superscalar PowerPC 440 embedded cores (note that L1 cache coherence is not maintained between these cores). Two copies of the PPC floating point unit (FPU) are associated with each core; this double FPU does not support independent operations [2]. Instead, the second FPU is used through an extensive set of parallel instructions for which the double precision operands can come from the register file of either unit. Raptor is well suited to this design; it has previously demonstrated outstanding performance on SSE units that provide similar capabilities. Once the compiler support for using the parallel instructions matures, we expect to make extensive use of them, including in the use of the `Fortran MERGE` intrinsic.

The system software supports two modes for applications to use the cores. In communication coprocessor mode (CM), one core is dedicated to servicing communication requests, offloading substantial message passing overhead from the first core that executes the actual computation. In virtual node mode (VNM), two MPI tasks run on each node, one on each processor. Although the coprocessor mode sacrifices half of the compute power of the machine and effectively splits the available main node memory, it realizes higher network bandwidth and was the original intent of the hardware design. Testing to date has focused on coprocessor mode, although some small node count VNM tests have been done.

The scale of LLNL's system substantially exceeds that of previous high-end computing platforms. The final machine, expected in the fall of 2005, will have 65,536 compute nodes for a total peak performance of over 360 Tflops; half of that machine is already installed and in use at LLNL. The bulk of porting applications such as Raptor to this system focuses on coping with this unprecedented processor count.

As shown in Figure 1, the final system will include 1024 I/O nodes that are nearly identical to the compute nodes. The difference is that they include gigabit ethernet (Gig-E) connections that connect the system to a large parallel file system. They also connect the system to the front-end nodes, on which users compile their codes and launch their jobs.

BG/L includes five networks: three dimensional torus, tree, global interrupt, Gig-E I/O and control. Three, the torus, the tree and interrupt, are used for MPI communications. Point-to-point communications are handled by the 16x32x64-node torus, with each node connected to its nearest neighbors via six independent bidirectional links with a raw hardware performance of 175 MB/s per link.The other MPI networks are used to perform global operations like broadcasts, reductions, and barriers with very low latency and high bandwidth. For example, the current 32,768-node machine can complete an `MPI_Barrier` across the entire machine in under 2 μsec.

Although measured MPI pingpong bandwidth between neighboring nodes is 150 MB/s, on-node message processing limits total realized MPI bandwidth over the torus links to less than 500 MB/s. This limitation is particularly relevant to Raptor: its domain decomposition requires a 26-way stencil computation in which an MPI task communicates with all of its neighbor in the logically surrounding cube of tasks. As will be discussed later, we are exploring the use of Hilbert curves to provide efficient MPI task mappings for Raptor.

3 The Adaptive Mesh Refinement Algorithm

Adaptive Mesh Refinement is fundamentally a technology for solving partial differential equations using a hierarchy of grids of differing resolution. The grid hierarchy is composed of different levels of refinement ranging from coarsest to finest. Each level is represented as the union of rectangular grid patches of a given resolution contained within the computational domain.

Both the initial creation of the grid hierarchy and the subsequent regriding operations, the operation of dynamically changing the grid hierarchy to reflect different flow conditions, use the same procedures to create new grids. Cells requiring additional refinement are identified

and tagged using a user supplied error estimation criteria and the tagged cells are grouped into rectangular patches. In general, the new patches contain cells that were not tagged for refinement. These rectangular patches are refined to form the grids at the next level. The process is repeated until either the error tolerance criteria are satisfied or a specified maximum level of refinement is reached.

The method we use for solving partial differential equations on a grid hierarchy is to solve on a given level using Dirichlet data obtained from coarser levels. This results in flux errors at the boundary with the coarse grid, which are then fixed in a synchronization step. The time-step algorithm recursively advances grids at different levels using time steps appropriate to that level based on Courant-Friedrichs-Levy (CFL) considerations and the flux corrections are typically imposed in a time-averaged sense.

Before turning to parallelization issues, we first discuss some of the details of the above algorithm related to the communication of data. Essentially all inter-level data communication occurs in two phases of the algorithm. The coarser grids supply boundary data in order to integrate finer grids, and the coarse and fine grids are synchronized at the end of fine grid time steps when the coarse and fine grid solution have reached the same time. For the case considered here, boundary data is provided by filling *ghost cells* in a band around the fine grid data whose width is determined by the stencil of the finite difference scheme. In the present case, we use five ghost cells in the normal direction. Four ghost cells are required by the fourth-order slope approximation [5] and the fifth one is required for the artificial viscosity calculation. A simple copy provides the data if it is available from grids at the same level of refinement. If the data is not available from grids at the same level, it is obtained by interpolation of the coarse grid data in time and space.

Two corrections are required when we synchronize the coarse and fine grids when they reach the same time. First, for all coarse cells covered by fine grid cells, we replace the coarse data by the volume weighted average of the fine grid data. Second, we must correct the coarse cell values by adding the difference between the coarse and fine grid fluxes because coarse cells adjacent to the fine cells were advanced using different fluxes than were used for the fine cells on the other side of the interface.

The AMR software is organized into five relatively separate components: 1) the error estimation and grid generation routines identify regions needing refinement and generate fine grids to cover the region, 2) grid management routines manage the grid hierarchy allowing access to the individual grids as needed, 3) interpolation routines initialize a solution on a newly created fine grid and also provide the boundary conditions for integrating the fine grids, 4) synchronization routines correct mismatches at coarse/fine boundaries arising because we integrate the levels independently, apart from boundary conditions and 5) the integration routines that discretize the physical processes on the grids.

4 Parallelization of AMR

We have adopted a coarse-grained, message-passing model, using MPI [20, 10], as our approach to parallelization. In this model, the grids or data blocks on a level are distributed across processors (or nodes) in such a way as to achieve load-balance of computational work across processors and to minimize the cost of data-communications between processors. For single level calculations, i.e. ones where we only advance level 0 data it is easy to construct a problem manually that is ideally load balanced: simply create a set of grids so that there are exactly N blocks of size M per processor.

For AMR calculations, it is in general difficult to achieve perfect load balance. The current AMR implementation is quite general in terms of the range of block sizes that can be generated for a given problem that meet the constraints of the grid generation algorithm. This means that for a typical problem, on any given level there is a range of sizes of blocks. By size, we mean

total number of cells in a block. We can use this range of sizes to our advantage to achieve load balance [17].

Generality, however leads to additional overhead. On each processor, we duplicate the mapping between processors and the array of grids that it contains. This provides a good optimization for pre-computing the communications required for filling boundary conditions on each processor. The storage requirements for storing the mapping per 1000 grids is 28 Kb. For full machine runs on BG/L, we envision millions of grids. This balloons the storage requirements for the mapping into the tens of Megabytes. With only 512 Mb total available, this model must be changed.

Regriding, and also some portions of the intra- and inter-level data communication, as currently implemented, contain some N^2 components where N is the number of data blocks. The algorithm to determine which processors must communicate boundary information when the blocks are of non-constant size is one such component. At nominal data block counts (e.g., 1000), regriding time is manageable. But at block counts approaching $1,000,000$, which is envisioned for BG/L at 32K nodes, N^2 is prohibitive.

5 Results

5.1 Single Level Scaling: Weak and Strong Results

In Figure 2, we show a summary of a number of scans (calculations performed on a fixed size partition and varying the number of blocks per node) for different partition sizes. We have performed these scans on both the LLNL machine and the IBM Rochester BG/L systems. In this plot, we use the time (in seconds) to advance a grid or block one step in time as the metric for measuring performance; smaller times are better. In computing this number, we take 5 steps and then average the time. Note that the time per step at a given number of blocks on BG/L is exceptionally reproducible due to very low system noise so that timings for every step are virtually identical. The curves show different partition sizes on the indicated system, using different versions of the system software (driver level). The driver levels correspond to the week of the year in which the source code was frozen. Three driver levels are shown in Figure 2. Driver 480 using system software source from late November 2004, while drivers 100 and 120 are from the middle and end of February 2005.

We compare runs based on the number of blocks per node. That is, we normalize the total work in each run by the partition size. This normalization compares the time per step across a range of partition sizes but ignores the increase in the aggregate amount of work as the partition size is increased. The normalization provides a compact representation of both weak and strong scaling in one plot. For blocks per node less than one, some processors are idle and contain no data. Thus, this part of the plot for a given partition size demonstrates Raptor's weak scaling behavior. For partition sizes less than or equal to 8k, the times are very flat which is the desired result, i.e., the time per step is constant as we scale up the problem size but keep the work per node constant. For blocks per node greater than 1, we are in the strong scaling regime. Here the work per node is increasing equally.

Comparing across the partition sizes, we observe a bi-modal distribution. That is, there is a lower cluster of curves and an upper one. The lower curve contains two results from the current LLNL BG/L 32k, running driver level 100, machine on 4k and 8k partition sizes. An 8k partition run from the 32k IBM Rochester BG/L system running driver level 120 also lies in the lower cluster. Although not shown, all smaller partition sizes give curves in this cluster. The upper curves were also obtained on the current BG/L 32k system on a 16k partition. A 16k IBM Rochester system result also lies in the upper cluster but has a flatter shape.

In the lower cluster of curves, Raptor's weak scaling is very good as these curves are essentially flat. That is, the time per step is constant as the problem size is scaled up while the work

per node is kept constant.

The code is exhibiting excellent strong scaling as shown when the blocks per node is greater than one. Both clusters are scaling at no worse than a linear rate (e.g., the time per step with two blocks per node is less than or equal to the twice the time per step with one block per node).

The time per step in the upper cluster is approximately twice that of the lower cluster for any given number of blocks per node. We are currently investigating the cause of this performance problem. Initial indications are that it arises from memory usage in the MPI library although this is as yet uncertain. We will report on these investigations more fully in the final paper.

5.2 Communications Efficiency

We have profiled a number of Raptor runs of various sizes on the BG/L systems using the IPM software package developed at NERSC [19]. IPM is a portable profiling infrastructure providing a low-overhead performance summary of the computation and communication in a parallel program. The amount of detail reported is selectable at runtime via environment variables or through a `MPI_Pcontrol` interface. IPM is scalable and easy to use, requiring no source code modification. Figure 3 shows, on the left, a breakdown of the total time spent in the MPI routines used by Raptor for a 1k run. On the right of the figure is shown the results for a 32k run. Note that the color coding is based on the relative amounts of time spent in each routine. `MPI_Barrier`, the largest time consumer, is artificially high since we have added `MPI_Barrier`'s to the code to force MPI time to be charged to this routine instead of reductions or send/receive. The timings for the `MPI_Barrier` are also used as a benchmark for examining communications induced load-imbalance. In the 1k case, `MPI_Send` is the next highest followed by the reductions (`MPI_Allreduce` and `MPI_Reduce`), `MPI_WaitAll` and then `MPI_Alltoall`. In the 32k run, after `MPI_Send`, we have a large `MPI_Waitall` that was very small in the 1k case. The reductions and `MPI_Alltoall` are also very small in a relative sense. The relative communications cost for the 32k run, as compared to the total wall clock time, is nearly 65% compared to only 10% for the 1k run. The differences between MPI routine usage in these two cases may help to understand the offset discussed above and will be reported more fully in the final paper.

5.3 Hardware Performance Counters

We used BG/L's hardware performance monitors to measure the floating point performance of Raptor. With only the main integration kernel instrumented, excluding all MPI message passing, on the full BG/L 32k system Raptor achieves 91.2 Mflops per node or 3.0 Tflops aggregate performance. This represents about 3.3% of the peak performance of the full BG/L system in CM despite only using half of the available floating point pipes. Analysis of smaller partition sizes (1k, 2k, 4k, 8k and 16k) show the same performance.

If we enlarge the portion of the code instrumented to include the MPI message passing required to fill ghost cells as well as the integration kernel, we obtain for another 32k run only 0.51 Mflops per node. Including the MPI overhead has severely degraded the performance. This is apparent in the smaller partition runs performed. The flop rates for 1k and 8k runs are 68.8 Mflops and 22.7 Mflops, respectively. We will continue to investigate this issue as we currently feel that understanding why MPI is slowing down so drastically will explain the time per step anomaly seen in the scaling studies.

5.4 AMR Results

Some preliminary AMR calculations have been performed on the LLNL BG/L system. We constructed a model problem that could be run on a 4k partition at initial time, but that would evolve to utilize the entire 32k system by the final time. The problem was an Argon bubble, formed using a soap film, that is accelerated by a planar strong shock wave with Mach number

of 2.88 in air. It contained 3 levels of refinement (over the base grid level), each level refined by a factor of 4. This gave a physical resolution on the finest level of $\Delta x = \Delta y = \Delta z = 50\ \mu m$. Using AMR, this is a relatively modest problem containing $250M$ cells on the finest level containing $34,746$ finest level grids initially. An equivalent uni-grid calculation at this same resolution would be over 250 *trillion* cells, clearly a untractable calculation.

The time spent in regrid, according to our native runtime profiling, is 31% of the total computation time while the main integration kernel used 61% of the time. The remainder of the time was spent in problem setup (which would be amortized when running for longer times) and communication overhead. We are pleased with these preliminary results and can make improvements over the short term.

6 Physics Problem Description

The problem we propose for Gordon Bell Prize consideration is a strong shock wave, Mach number of 2.88, impinging on an Argon bubble, using soap film to contain the gas, in air. This problem is based on the model problem reported in Section 5.4. For low Mach numbers, this problem has been extensively studied experimentally [11] and numerically [15] in two-dimensions. In the high Mach number regime, there are high energy density experiments conducted on laser platforms [18] and the recent experimental study at the University of Wisconsin [16], as well as the three-dimensional numerical work in support of that later experiment [13]. The shock bubble interaction is an important fluid dynamics problem as it contains all the fundamental fluid instabilities in a tightly coupled flow as well as having implications that extend into the fluid dynamics of astrophysics. Specifically, an important problem in astrophysics is the interaction of shock waves with interstellar gas clouds [12].

Niederhaus, et al. used a resolution of R_{120} which means there are 120 cells per radius of the bubble, following the conventional resolution notation for these problems. It was completed on LLNL's IA32 ALC Linux Cluster using 128 $1.7GHz$ Xeon Processors for approximately 7 hours. For these so-called shock/bubble problems, R_{120} is the highest resolution simulation achieved to date in three-dimensions. In Figure 4, we show a rendering of the shocked bubble at late time from this calculation. The red ring-like structures are isosurfaces of vorticity magnitude. The blue isosurface is the Argon gas concentration and the color field where the Argon is cut-away is the concentration of soap film material. There are three clearly defined vortex rings in the flow as well as substantial turbulence. The behavior of the turbulence and the associated vortex dynamics, the mixing of the Argon with the background gas as well as the late time development of the ring-like structures are of key scientific importance.

It is well known [12] that R_{120} is required to obtain moderate convergence on integral flow measures in two-dimensions. However to obtain the details of the internal structure of shock driven flows in three-dimensions, resolution approaching the Taylor scale is required. The model problem using the resolution described in the Section 5.4 represents an R_{512} calculation which is a factor of 4 larger, per bubble radius, than any other previous simulation. Note that a factor of four in linear dimension corresponds to a three-dimensional problem that is 64 times larger overall.

Admittedly 50 μm resolution, R_{512}, is insufficient to resolve the Taylor scale, $O(1\ \mu m)$, but it does represent nearly *two* orders of magnitude increase in size (factor of 64) over the largest three-dimensional calculations of this type previously attempted. BG/L is making calculations possible that currently available machines cannot. We are confident that this problem can be completed given enough access to the BG/L system at LLNL.

7 Path Forward Development

To achieve scalability and to meet the stringent data storage requirements imposed by BG/L's limited memory, significant refactoring is required of the general AMR algorithm. The main feature of AMR that has to be surrendered is the use of arbitrary size data blocks in the AMR hierarchy. With arbitrary sized data blocks it is not possible to conduct the regrid and data communications in less than $O(N^2)$ (N is the number of blocks) operations or to limit the amount of meta-data to less than an $O(N)$ amount of storage. It is possible to reduce several, but not all, of the $O(N^2)$ methods used in data-communication to $O(N \log N)$ but several portions of the method can not be so reduced.

The refactored version of Raptor decomposes the data at a given level into equal sized blocks. This specialization makes load-balance trivial: the software need only ensure that each processor has an equal number of blocks. This also solves the storage requirement for the mapping on each processor. With equal sized grids on each level, there is a simple bit-wise method for computing which processors must communicate boundary condition information that is $O(N/P)$ (P is the number of processors).

The second issue of ensuring that the data blocks are distributed in such a way as to lower communication cost is less trivial. In this implementation we have chosen to use compositions of Hilbert curve mappings. Hilbert curves are examples of space-filling curves that map integers into the 2 dimensional plane or 3 dimensional space in such a way that integer values that are close map to positions in the volume that are also close. Hilbert curves, and other space-filling curves, have been used extensively in the past in a variety of applications including parallel domain decomposition [1] and in the parallel load-balancing [14]. However, since points close in 2 or 3 dimensional space are not necessarily mapped to points close in the space filling curve, the Hilbert curve mapping can, at best, be expected to only reduce the number of messages that require large communication paths.

Much of the `Fortran` code will have to be written to gain full performance from the double FPU. Programming practices that did not affect code performance on DEC Alpha based or Intel based systems parallel systems seem to matter a great deal on the 440-based BG/L system. For example, loops containing an `if` statement must be inverted or replaced with a `MERGE` statement. Some care must also be taken to ensure proper 16-byte alignment, but this can probably be dealt with by inserting conditionally compiled `pragma`s. Also a more careful examination of the `Fortran` may yield opportunities for taking advantage of the double FPU capability for better floating point performance.

8 Conclusion

Raptor is a block-structured AMR application for simulating a diverse range of physical systems in fields such as shock physics, astrophysics and ICF. The computational core of Raptor has achieved 3.0 Tflops aggregate performance on the 32k compute node BlueGene/L system at LLNL. We have also discussed planned improvements to the application to improve its scaling behavior as well as issues with some part of the BG/L system software that thus far limits the code's overall performance. Despite these limitations, our initial results demonstrate that we can perform a shock wave computation with important scientific implications at an unprecedented resolution. The BG/L design is enabling simulations that will provide insight into a problem with implications both in the laboratory and astrophysics. The BG/L design is providing nearly *two* orders of magnitude (factor of 64) increase in the size of the three-dimensional simulations of this problem, well beyond the current capabilities of other large-scale machines.

References

[1] S. Aluru and F.E. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *4th International Conference on High-Performance Computing*, pages 230–235, 1997.

[2] L. Bachega, S. Chatterjee, K. Dockser, J. Gunnels, M. Gupta, F. Gustavson, C. Lapkowski, G. Liu, M. Mendell, C. Wait, and T.J.C. Ward. A high-performance simd floating point unit design for BlueGene/L: Architecture, compilation and algorithm design. *Parallel Architecture and Compilation Techniques (PACT 2004)*, 2004.

[3] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82(1):64–84, May 1989.

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

[5] P. Colella. A direct Eulerian MUSCL scheme for gas dynamics. *SIAM J. Sci. Statist. Comput.*, 6:104–117, 1985.

[6] P. Colella, H.M. Glaz, and R.E. Ferguson. Multifluid algorithms for Eulerian finite difference methods. *unpublished*, 2002.

[7] R.D. Falgout, J.E. Jones, and U.M. Yang. *The Design and Implementation of HYPRE, a Library of Parallel High Performance Preconditioners.* Numerical Solution of Partial Differential Equations on Parallel Computers. Springer-Verlag, to appear.

[8] Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory. `http://seesar.lbl.gov/CCSE`.

[9] J.A. Greenough and J.W. Jacobs. A numerical study of shock-acceleration of a diffuse helium cylinder. *Proceedings of the Fifth International Workshop on the Physics of Compressible Turbulent Mixing*, July 1995.

[10] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Scientific and Engineering Computation. The MIT Press, Cambridge, Mass, 1994.

[11] J.-F. Haas and B. Sturtevant. Interation of weak shock waves with cylindrical and spherical gas inhomogeneities. *Journal of Fluid Mechanics*, 181:41–76, 1987.

[12] R. I. Klein, C.F. McKee, and P. Colella. On the hydrodynamic interaction of shock waves with interstellar clouds. I. Nonradiative shocks in small clouds. *Astrophysical Journal*, 420, 1994.

[13] J. Niederhaus, J. Oakley, M. Anderson, D. Ranjan, R. Bonazza, and J. Greenough. Mach number scaling and soap film effects in 3-d computations for a shocked spherical gas bubble. *Physics of Fluids*, in review, 2004.

[14] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with space-filling curves. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):288–300, 1996.

[15] J. Quirk and S. Karni. On the dynamics of a shock-bubble interaction. *Journal of Fluid Mechanics*, 318:129–163, 1996.

[16] D. Ranjan, M. Anderson, Oakley J., and R. Bonazza. Experimental investigation of a strongly shock gas bubble. *Physical Review Letter*, in review, 2004.

[17] C.A. Rendleman, V.E. Beckner, M. Lijewski, W.Y Crutchfield, and J.B. Bell. Parallelization of structured, hierarchical adaptive mesh refinement algorithms. *Computing and Visualization in Science*, 3(3):147–157, 2000.

[18] H.F. Robey, T.S. Perry, R.I. Klein, J.O. Kane, J.A. Greenough, and T.R. Boehly. Experimental investigation of the three-dimensional interaction of a strong shock with a spherical density inhomogeneity. *Physical Review Letters*, 89(8), 2002.

[19] David Skinner. National Energy Research Scientific Computing Center, `http://www.nersc.gov/projects/ipm/`.

[20] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. Scientific and Engineering Computation. The Mit Press, Cambridge, Mass, 1996.

[21] BlueGene/L Team. An overview of the BlueGene/L supercomputer. *Proceedings of the ACM/IEEE SC2003 Conference*, November 2003.

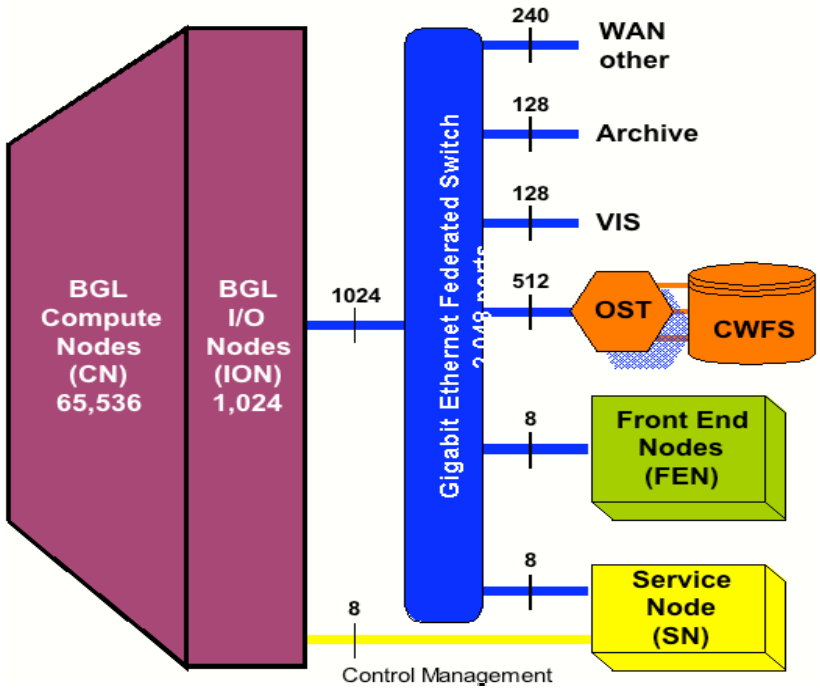

Figure 1. An overview of the integrated BG/L system architecture.

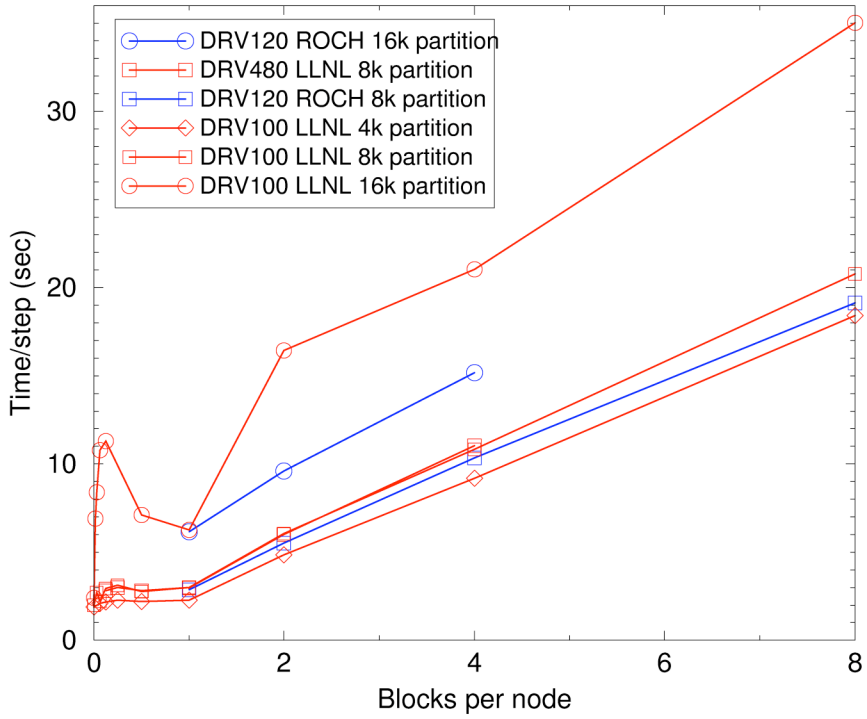

Figure 2. The summary of Raptor runs on various partition sizes of the BG/L systems at LLNL and IBM Rochester. The average time per step, in seconds, is plotted versus data blocks per node.

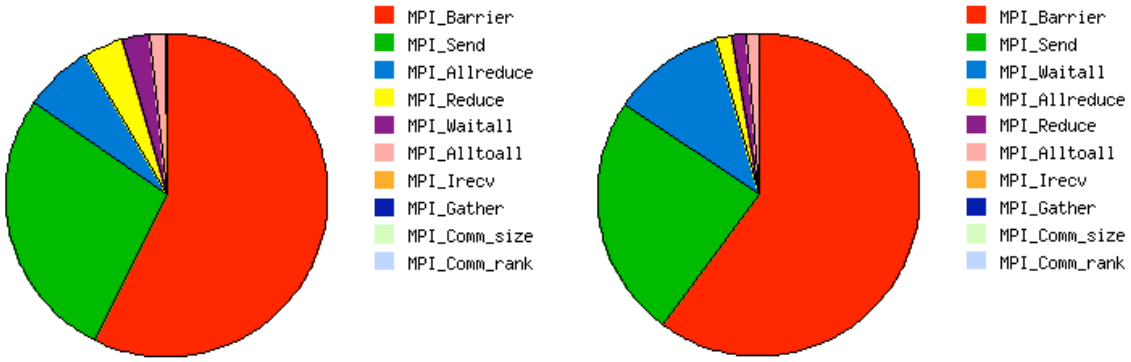

Figure 3. A summary from a 1k and a 32k run, on the left and right respectively, of the relative amounts of time spent in the various MPI routines used by Raptor. Note that the color coding is based on the relative amounts of time spent in each routine.

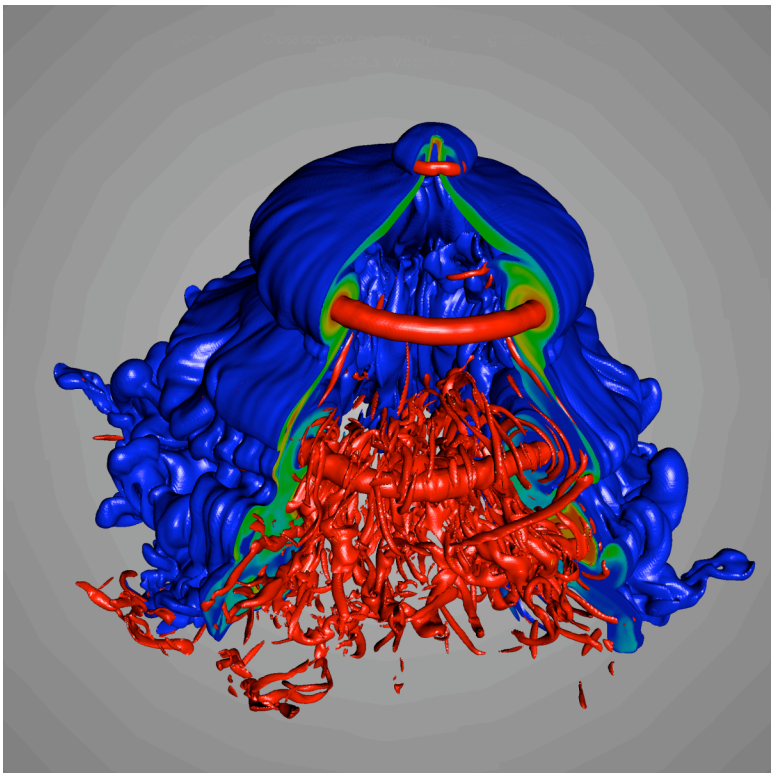

Figure 4. A visualization of the late time structure of an Argon bubble that has been accelerated by a strong, M = 2.88, shock wave at t = 509 μsec. The blue is an isosurface Argon, the red is an isosurface of the vorticity magnitude. On the cut-away is shown the soap film concentration.