Cactus-G: Enabling High-Performance Simulation in Heterogeneous Distributed Computing Environments

Gabrielle Allen[*] Thomas Dramlitsch[*] Ian Foster[†‡] Tom Goodale[*]
Nick Karonis[§] Matei Ripeanu[‡] Ed Seidel[*] Brian Toonen[†]

May 7, 2000

Abstract

Improvements in the performance of processors and networks means that it can be both feasible and interesting to treat collections of workstations, servers, clusters, and supercomputers as integrated computational resources or Grids. However, the highly heterogeneous and dynamic nature of such Grids makes application development extremely difficult. In this paper, we describe an architecture and prototype implementation for a Grid-enabled computational framework called Cactus-G. This framework integrates the Cactus simulation system with the MPICH-G2 Grid-enabled message passing library and in addition integrates a variety of specialized features to support efficient execution in Grid environments. In order to evaluate and demonstrate the effectiveness of this system, we are attempting a challenge computation involving a large astrophysics simulation distributed across multiple supercomputers at U.S. centers. In this extended abstract, we present preliminary results that suggest that this challenge computation is feasible; the final paper will present complete results.

KEYWORDS: Heterogeneous and Distributed Computing, Metacomputing, Problem Solving Environments, Grid, Numerical Relativity, Cactus, MPI, MPICH-G2

1 Introduction

A continued rapid evolution in both the sophistication of numerical simulation techniques and the acceptance of these techniques by scientists and engineers means that demand for computing cycles is increasing rapidly. This observation is particularly true at the high end of the scale: the supercomputer centers capable of supporting the most realistic simulation studies are all

[*]Max Planck Institute for Gravitational Physics
[†]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
[‡]Department of Computer Science, The University of Chicago, Chicago, IL 60637
[§]Department of Computer Science, Northern Illinois University

grossly oversubcribed. And while commodity clusters are emerging as a promising lower-cost alternative to tightly integrated supercomputers, they seem only to be spurring further demand.

At the same time, the capabilities of both "low-end" computers and commodity networks are increasing rapidly, to the point where a typical research or engineering institution will soon include large numbers of Gigaflop/s workstations connected by Gigabit/s networks, in addition to the usual collection of high-end servers and clusters. It thus becomes feasible and indeed interesting to think of the high-end computing environment as an integrated "Computational Grid" [15] rather than a set of disjoint point sources. For this Grid to be widely useful, it is essential to be able to design applications that are flexible enough to exploit various ensembles of workstations, servers, commodity clusters, and true supercomputers, matching application requirements and characteristics with Grid resources to which we have access.

It is clear that the effective exploitation of such Grid computing environments could increase dramatically the accessibility and scale of large-scale simulation. However, the development of such Grid-enabled applications (outside the domain of the "happily," i.e., trivially, parallel) presents a very significant challenge, due to the high degree of heterogeneity and dynamic behavior (in architecture, mechanisms, and performance) encountered in Grid environments. While methods for dealing with at least some of these difficulties are known and have been applied successfully in some situations [26, 27, 6, 28, 34], applying these methods in real applications tends to be extremely difficult.

A promising approach to application development in such environments is to develop *Grid-enabled computational frameworks* that implement the advanced techniques referred to above, hide irrelevant complexities, and present application developers with familiar abstractions.*In this paper, we present such a framework and describe a challenge computation that we are conducting with the goal of demonstrating the effectiveness of our approach.*

The framework, which we term Cactus-G, builds on two major software systems, namely the Cactus simulation framework [3, 2] and the Globus-based MPICH-G2 implementation of MPI integrating the two systems and extending them to incorporate various advanced techniques for Grid execution.

The challenge computation involves a computationally demanding problem from astrophysics, namely the detailed simulation of two colliding black holes. We seek, by harnessing multiple supercomputers, to execute this simulation at an unprecedented resolution. However, the framework provided by Cactus-G is sufficiently general that many different applications can become Grid-enabled; it is not in any way limited to the pariuclar astrophysics application. This extended abstract presents preliminary results that suggest that this challenge computation is feasible; the full paper will include detailed performance analysis and results for the full problem.

2 The Computational Grid Environment

The computational Grid environments that we seek to harness in this work are different in many respects from the "standard" parallel computing environment. In particular:

- A parallel computer is usually fairly homogeneous. In contrast, a Grid may incorporate nodes with completely different processor types, memory sizes, etc.

- A parallel computer typically has a dedicated, optimized, high bisection bandwidth communications network with a generally fixed topology. In contrast, a Grid can have a highly heterogeneous and unbalanced communication network, comprising a mix of different intramachine networks and a variety of Internet connections whose bandwidth and latency characteristics may vary greatly in time and space.

- A parallel computer typically had a stable configuration. In contrast, resource availability in a Grid can vary dramatically over time and space.

- A parallel computer typically runs a single operating system and provides basic utilities such as file system and resource manager. In contrast, a Grid environment, being a collection of single vendor machines, integrates potentially radically different OS and utilities.

A conventional parallel program will not run efficiently (or perhaps at all) in such an environment. High communication latencies, low bandwidths, and unequal processor powers will together ensure that overall performance is poor. The following is a partial list of the specialized techniques that can be used to overcome these problems.

1. *Smart resource selection.* While many resources may be available to us, not all may be appropriate for our application. A resource selection strategy that takes into account both application characteristics and resource properties (e.g., performance, cost) can allow us to meet cost and performance requirements [35].

2. *Resource reservations.* A variant of #1 is to use reservation mechanisms to guarantee availability of critical resources (e.g., networks) [16].

3. *Irregular data distributions.* We can avoid load imbalances by using irregular data distributions that optimize overall performance. In computing these data distributions, we need information about the application itself, the target computers, and the networks that connect these computers [31, 29].

4. *Grid-aware communication schedules.* We can schedule communication (and computation) so as to maximize overlap between computation and communication, for example by computing on the interior of a region while exchanging boundary data. We can group communications so as to increase message sizes. We can also organize data distributions and/or use dedicated communication processors so as to manage the number of processors that engage in inter-machine communication.

5. *Redundant computation.* We can perform redundant computation to reduce communication costs. For example, increasing the size of the "shadow" region in a finite difference code allows us to increase communication granularity significantly, at the cost of unnecessary computation.

6. *Coroutining of other computation.* It is common in scientific computing to analyze output data after computation completes, in a separate postprocessing phase. The amount of data and computing involved in postprocessing can be significant, so in an Grid environment, we may want to trade off computation for communication by coroutining "postprocessing" with simulation.

7. *Protocol tuning.* We can tune a particular protocol based on known characteristics of the application and environment: e.g., by selecting TCP window sizes [33].

8. *Selection of alternative protocols.* We can use specialized protocols for wide area communication that take into account application-specific knowledge. For example, we may be able to exploit multicast or use protocols other than TCP, so as to avoid TCP overheads such as slow start.

9. *Adaptive strategies.* We can compensate for changes in the behavior of the application and/or resources by reapplying any of the strategies listed above during program execution [30, 19].

10. *Network-aware communication algorithms.* We can improve the performance of communication operations by using specialized network-aware algorithms [24, 10, 20, 21, 5].

11. *Grid infrastructure.* We can reduce dramatically the difficulties associated with operating in heterogeneous multi-domain environments by using appropriate Grid services (e.g., resource discovery, single-sign on, resource management) to access remote resources [14, 17, 23, 37].

3 The Cactus-G Toolkit

Each of the techniques listed in the preceding section is difficult to apply in an application program; applying a collection of these techniques in a coordinated fashion can be extremely challenging. The Cactus-G Toolkit represents an attempt to overcome this challenge. In the following, we first describe the Cactus and MPICH-G2 systems, then describe the architecture of Cactus-G, and finally review the status of our Cactus-G prototype.

3.1 Cactus and MPICH-G2

Originally developed as a framework for the numerical solution of Einstein's Equations [32], Cactus [7, 3, 2] has evolved into a general-purpose, open source, problem solving environment that provides a unified modular and parallel computational framework for scientists and engineers.

The name Cactus comes from its design, which features a central core (or *flesh*) which connects to application modules (or *thorns*) through an extensible interface. Thorns can implement custom-developed scientific or engineering applications, such as computational fluid dynamics, as well as a range of computational capabilities, such as data distribution and checkpointing. An expanding set of Cactus toolkit thorns provides access to many software technologies being developed in the academic research community, such as the Globus Toolkit, as described below; HDF5 parallel file I/O; the PETSc scientific computing library; adaptive mesh refinement; web interfaces; and advanced visualization tools.

Cactus runs on many architectures, including uniprocessors, clusters, and supercomputers. Parallelism and portability are achieved by hiding features such as the MPI parallel driver layer, I/O system, and calling interface under a simple abstraction API. These layers are themselves implemented as thorns that can be interchanged and called as desired. The PETSc scientific library has similar concepts of data distribution neutral libraries [4], but Cactus goes further by providing modularity at virtually every level. For example, the abstraction of parallelism allows one to plug in different thorns that implement an MPI-based unigrid domain decomposition, with very general ghost zone capabilities, or an adaptive mesh domain decomposer, or a PVM version of the same kinds of libraries. A properly prepared scientific application thorn will work, without changes, with any of these parallel domain decompositon thorns, or others developed to take advantage of new software or hardware technologies.

The second system that contributes to Cactus-G is MPICH-G2, an MPI implementation designed to exploit heterogeneous collections of computers. MPICH-G2 is a second-generation version of the earlier MPICH-G [13]. Like MPICH-G, MPICH-G2 exploits Globus services [14] for resource discovery, authentication, resource allocation, executable staging, startup, management, and control; it extends MPICH-G by incorporating faster communications and quality of service, among other new features.

Other efforts concerned with message passing in heterogeneous environments include PACX [18], MetaMPI, STAMPI [22], IMPI [8], and MPIconnect [12]. MPICH-G2 is distinguished by its tight integration with the popular MPICH implementation of MPI and its use of Globus mechanisms to overcome security and other problems.

3.2 Cactus-G Architecture

The Cactus-G system defines a set of appropriately layered abstractions and associated libraries, such that irrelevant (from a performance viewpoint) complexities are hidden from higher layers, while performance-critical features are revealed. The design of such a system must inevitably evolve over time as a result of empirical study. However, our experiences to date persuade us that the architecture illustrated in Figure 1 has some attractive properties, as we describe in the following.

At the highest level, we have a *Grid-aware application*. For the user, the abstraction presented is a high-performance numerical simulation. The user controls the behavior of the simulation by specifying initial conditions, resolution, etc.; the simulation may in turn reveal performance data. All details of how this simulation is performed on a heterogeneous collection of computers are encapsulated within the application.

This *Grid-aware application* is built from several layers. At the top of these layers lie the various Cactus *application* thorns, which are used to perform the actual scientific calculation (e.g., solve differential equations from various branches of physics). These thorns can be written in Fortran or C, but they do not necessarily have to be *Grid-aware*. The application programmer only needs to take care to use correct algorithms, differenting-schemes, equations etc. Some of these schemes will be better than others in a Grid environment, and hence a Grid-aware set of application thorns will be useful, but in principle not required. All details of how data is distributed across processors and how communication is done is still hidden from the programmer at this level.

Next, we have the *Grid-aware infrastructure* thorns, which provide all features, layers, and drivers which the application thorns need, namely parallelism, I/O, web-interfaces, visualization and many more. These thorns contain all details about communication, data mapping, parallel-I/O etc. An application thorn writer simply includes those thorns into his code (without having to modify his application thorn) depending on his needs (e.g. wants to run single,

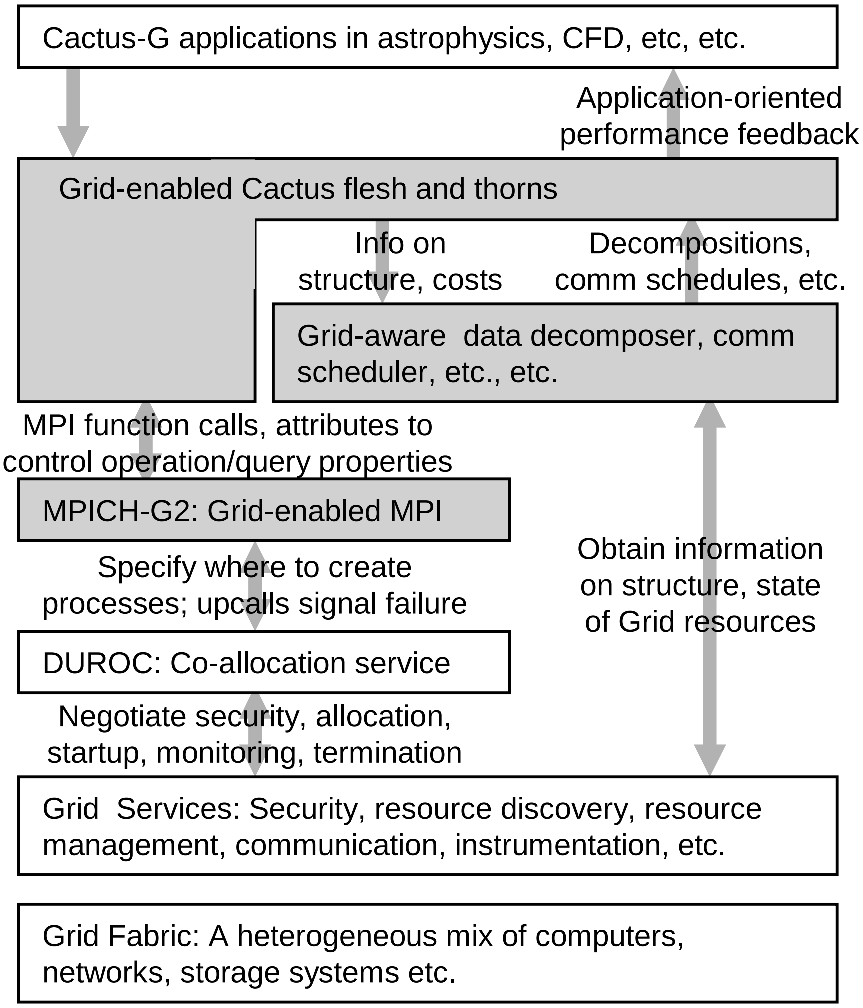

Figure 1: Cactus-G architecture, showing the information and control flows between the different layers

parallel or multi-host jobs). An important thorn in our case is the so-called PUGH-thorn, wich provides parallelism based on a MPI-implementation such as MPICH-G2. This thorn has been specifally improved and optimized in order to work even more efficiently in a heterogenous Grid environment. Other important thorns include those used to compute Grid-oriented data distributions, communication schedules, and so forth.

Below this we have a *Grid-enabled communication library*: specifically, MPICH-G2, an implementation of the MPI standard capable of running MPI programs across heterogeneous computing resources. The abstraction presented is the MPI programming model; the programmer queries structure and state and controls behavior of the underlying implementation via getting and setting, respectively, attribute values associated with MPI communicators. All details of how the MPI programming model is implemented across different resources are encapsulated, including startup, Grid-aware collective operations [20], monitoring, and control.

The Grid-enabled MPI implementation makes use of functions provided by a co-allocation library: in our case, the Dynamically Updated Resource Online Co-Allocator (DUROC) [9].

This library abstracts away details relating to how a set of processes are created, monitored, and managed in a coordinated fashion across different computers. The programmer can specify a set of machines on which processes are to be started; the DUROC library manages the acquisition of those resources and the creation of those processes. Upcalls are used to notify higher-level code of errors, timeouts, etc., hence allowing higher-level code to adapt if desired.

Finally, a set of *Grid services* abstract away the myriad complexities of heterogeneous environments. These services provide uniform protocols and APIs for discovering, authenticating with, reserving, starting computation on, and in general managing computational, network, storage, and other resources.

3.3 Cactus-G Implementation

While we are far from having a complete realization of the architecture just described, we have implemented substantial components. In particular, we have developed Grid-aware thorns for Cactus that support flexible data distributions, hence enabling (for example) grid points to be mapped to processors in a heterogeneous system according to their power and the amount of off-machine communication they have to do. We have also incorporated support for variable-sized shadow regions, hence allowing message size to be increased at the cost of some redundant computation. Furthermore, these techniques have been shown to work with MPICH-G2, which supports external management of TCP protocol parameters, the simultaneous use of multiple communication methods, Grid-aware collective operations, and efficient and secure startup across multiple computers. In principle a large family of scientific and engineering application thorns may be plugged into this environment, becoming Grid-enabled.

To date, the selection of data distribution, communication strategy, and so forth are largely manual processes, with a few exceptions. Data gathered during the present study will help with development of more automated methods, for example for computing shadow region sizes.

4 The Challenge Problem: Colliding Black Holes

In order to evaluate the effectiveness of the Cactus-G system, we apply it to a substantial application, one that has been driving an entire community of astrophysicists, relativists, and computational scientists for many years: namely, the collision of two black holes. One reason for current scientific interest in this problem is that new observatories may be able to detect the gravitational waves generated by such events [1].

The colliding black holes problem requires a solution to Einstein's equations of general relativity, which are some of the most complicated in mathematical physics, forming a complex set of nonlinear elliptic-hyperbolic coupled equations with dozens of variables and thousands of terms. This problem has generated numerous large scale collaborations among researchers across many disciplines, has led to several "Grand Challenge"-style projects in both the U.S. and in Europe [25, 32, 36, 11].

The equations to be solved are similar in some ways to those of CFD, only much more complex. They are solved here by finite difference techniques on a regular mesh, with an explicit time stepping scheme. Local communications are required between all neighboring grid points due to the necessary computation of spatial derivatives appearing in the equations. As second order finite difference approximations to derivatives are used, in principle a stencil width of two is needed for this example; however, higher stencil widths, requiring more communications, are supported and needed for some applications.

Our challenge problem involves multiple time steps of an extremely large Cactus computation, involving a mesh of size at least 256×256×1024—and hopefully 512×512×2048. The latter would be an order of magnitude larger than the largest production relativity simulations ever performed on a single dedicated machine (a 128 GB Origin 2000). The methods used are finite difference based, explicit time stepping procedures on an initial data set respresenting two black holes. Such calculations are quite demanding computationally, typically requiring several thousand floating point operations per grid point per time step, although with the most basic algorithms used for testing Grid scenarios we have managed to reduce this to about 750 flops/time step/grid zone. Scaling within a single parallel computer is excellent [32]; scaling between multiple computers depends critically on communication performance, as discussed below. In our initial work, we do not coroutine postprocessing, but an obvious candidate is the calculation of certain mathematical contraints on the solution which are known to be true analytically. The violation of these contraints is expected due to numerical error, and in production simulations these are always computed to assess the quality of the solution. The calculation of one of these four contraints adds several hundred flops per grid point per timestep, without adding to communication costs.

The Grid system on which we plan to run this computation comprises supercomputers at supercomputer centers within the U.S.: specifically, the 1,152-processor IBM RS/6000 SP at SDSC, in San Diego, California; a 512-processor T3E at NERSC, in Berkeley, California; a 1500-processor SGI Origin array at NCSA, in Urbana-Champaign, Illinois; and a 128-processor SGI Origin at ANL, in Argonne, Illinois. These systems are connected via Abilene and other high-speed networks, which in principal should provide satisfactory performance.

4.1 Performance Model

We have built a performance model in order to evaluate the performance that we can expect to achieve in a Grid environment. This model is parameterized with the number of processors, processor performance, and network performance, allowing us to quantify the sensitivity of overall performance to these parameters.

While space constraints do not permit a detailed presentation of this model, in brief a typical situation is as follows:

- A typical processor can manage a subgrid of size $64 \times 64 \times 64$, which is of order 1 Gflops for typical black hole simulations per time step.
- The off-machine communication volume associated with this subgrid in the case of "external" processors, if we assume a 1-D decomposition across computers, will be 64×64×(2 ghost zones)×(7 fields)×(8 bytes/word)=458 KB per time step.
- Hence, if we assume a 100 Mflop/s execution rate, a 1 MB/s network, and an intermachine latency of 50 msec, then total computation and communication time per time step will be 10 seconds and 0.5 seconds, respectively, assuming no computation/communication overlap. This corresponds to a computational efficiency of about 95%.

- If we scale things to assume an $4 \times 4 \times 4$ cube of computers on each computer, then total communication time increases to about $0.05 + 16 \times 0.45 \approx 7.2$ seconds, or around 60% efficiency; with computation/communication overlap and the use of various other techniques listed above, much of this loss of efficiency could be recovered. 80% overall efficiency is not unrealistic.

These numbers suggest that scaling across multiple geographically distributed supercomputers is not impossible, although total performance achieved will be quite sensitive to details of intermachine communication performance, load balancing, and our ability to overlap computation and communication.

If we assume performance similar to that listed here, then in principle we can hope to achieve performance in the range:

$$(3000 \text{ processors}) \times (100 \text{ Mflop/s}) \times (80\% \text{ efficiency}) = 300 \text{ Gflop/s}$$

While certainly not a record in terms of total Flop/s rate achieved by an application, we believe that such a result would serve as a powerful validation of the power of Grid computing.

4.2 Results to Date

At the time of writing, we have not performed the complete computation just described. However, we have performed a wide variety of preparatory experiments that lead us to believe that we can indeed complete this computation prior to submission of the complete paper. Specifically, we have performed:

- MPICH-G2 performance studies on each of the target platforms—at the time of writing, the NERSC T3E, SDSC SP2, NCSA Origin, and ANL Origin—and verified that performance is close that achieved by native MPI on those platforms;
- single-processor Cactus performance studies on each of the target platforms and verified that Cactus achieves good performance on these platforms;
- single-machine performance studies on each of the target platforms and shown that we get good single machine scalability on each of these platforms, when using MPICH-G2;
- performance tests over the network links that we expect to use, and shown that these links can deliver reasonable performance.

We have also encountered a wide variety of mostly mundane problems (which is why we have not yet completed the computation), relating in part to the fact that we are attempting both to run across multiple computers to run across large numbers of processors on each computer. These problems have included hardware failures, hardware upgrades, OS upgrades, scheduler bugs, scheduling features that prevented true "dedicated" time, interactions between vendor MPI and TCP implementations, and OS file descriptor limits. At the time of writing, we believe that we are able to deal with most of these problems.

We provide more details on selected performance results in the following.

Cactus Single Computer Performance. The Cactus code has been shown to be highly efficient, scalable, and portable across many architectures. Certain application thorns have been developed especially for performance testing on new machines, and used as a standard benchmark application in some cases. It has achieved over 95% scaling efficiency on a 1024 processor Cray T3E, where a large set of application thorns that solve Einstein's equations with general relativistic hydrodynamics achieved 142 GFlops/s. Another application thorn set achieved over 150 MFlops/processor on the 250Mhz R12000 based Origin 2000, with over 90% scaling on 256 processors, and also achieved similar scaling on the 128 processor NT supercluster at NCSA and a 128 processor linux cluster [32]. It has also been run already in Grid environments for going back the 1995 I-WAY experiments. Most recently, at SC98 and SC99 it was run on multiple T3E's in Europe and the US, achieving reasonable scaling (of order 50%) under very uncontrolled conditions. It is on top of this background that we are developing and implementing the new techniques described in this paper to create Cactus-G.

MPICH-G2 Performance. MPICH-G2 maps MPI communication operations into native MPI operations within a parallel computer or TCP/IP operations between computers. This is generally an effective strategy, as by now native MPI implementations are typically highly efficient. In the MPICH-G2 prototype, overhead is around 5% for the message sizes that arise in our challenge problem.

MPICH-G vs. Native MPI in Cactus The neglible impact of MPICH-G2 overheads on Cactus performance are demonstrated in Figure 2. We see that when using only MPI communication, overheads are tiny; even when using both MPI and TCP/IP communication, overheads are just a few percent.

Wide Area Performance. As might be expected, wide area performance varies greatly according to which sites are involved. The two most distant and least well connected sites are NCSA and SDSC; here, a single TCP ping-pong application can sustain 300 KB/s with basic tuning of TCP parameters and no network engineering work. We expect that this number can be improved to something close (or hopefully better) than the 1 MB/sec assumed in our performance model above.

4.3 Additional Results for Full paper

We are well aware that this paper is extremely light on results: the difficulties noted above mean that we only recently started on the intermachine work that are of course at the heart of this work. Our full paper will include the following additional material:

- The results of detailed performance studies across a range of multi-machine configurations, quantifying the impact of different Grid programming techniques on performance.
- Performance results for a large, optimized multi-machine run on at least 3000 processors and involving a real astrophysics problem. (There is a potential for many more than 3000 processors: if we included all of the sites that have expressed an interest in participating, we would have at least 6,000 processors. However, it is not clear that connectivity to all these sites is adequate.)

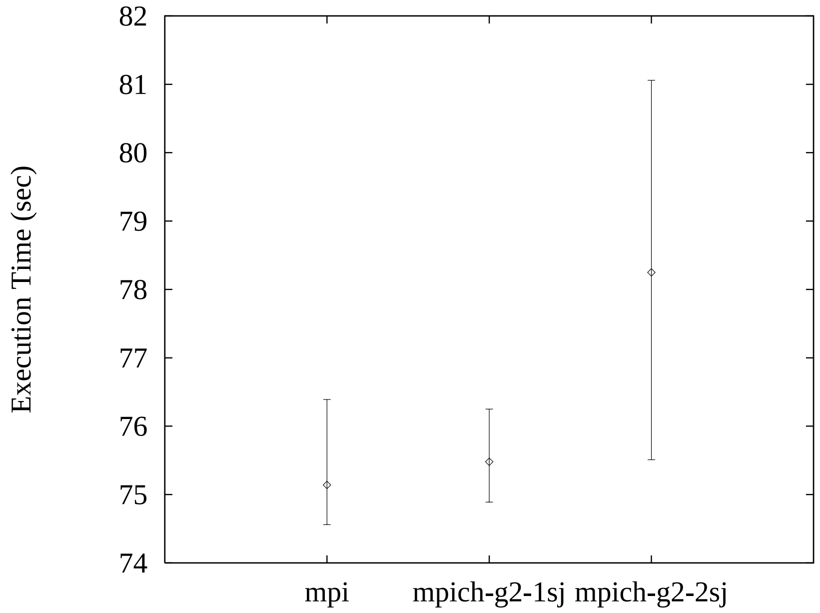

Figure 2: Cactus performance, in seconds per time step, when executing a $256 \times 256 \times 320$ problem on 64 processors of the SDSC Blue Horizon machine. Results are presented for native MPI, MPICH-G2 within a single partition, and MPICH-G2 when running across two 32-processor partitions, and hence using both native MPI (within a partition) and TCP/IP (between partitions) communications. The error bars show the minimum, mean, and maximum times measured in 10 runs.

5 Acknowledgments

We are deeply grateful to staff at NCSA, SDSC, NERSC, and ANL who helped us perform the experiments described here: without their help, this work would not have been possible. We are particularly grateful to Sandra Bittner, Randy Butler, Tina Butler, Brent Draney, Victor Hazlewood, Doru Marcusiu, Philip Papadapoulos, John Shalf, Keith Thompson, and Kenneth Yoshimoto.

We are also pleased to acknowledge many helpful discussions with colleagues within the Grid Application Development Software (GrADS) project, in particular Francine Berman, Andrew Chien, Keith Cooper, Jack Dongarra, Dennis Gannon, Lennart Johnsson, Ken Kennedy, Carl Kesselman, Daniel Reed, Linda Torczon, and Richard Wolski.

References

[1] A. A. Abramovici, W. Althouse, R. P. Drever, Y. Gursel, S. Kawamura, F. Raab, D. Shoemaker, L. Sievers, R. Spero, K. S. Thorne, R. Vogt, R. Weiss, S. Whitcomb, and M. Zuker. Ligo: The laser interferometer gravitational-wave observatory. *Science*, 256:325–333, 1992.

[2] G. Allen, W. Benger, C. Hege, J. Massó, A. Merzky, T. Radke, E. Seidel, and J. Shalf. Solving einstein's equations on supercomputers. *IEEE Computer*, 32(12), 1999.

[3] G. Allen, T. Goodale, and E. Seidel. The cactus computational collaboratory: Enabling technologies for relativistic astrophysics, and a toolkit for solving pdes by communities in science and engineering. In *7th Symposium on the Frontiers of Massively Parallel Computation-Frontiers 99*, New York, 1999. IEEE.

[4] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[5] P. Bhatt, V. Prasanna, and C. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1998.

[6] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman. Implementing distributed synthetic forces simulations in metacomputing environments. In *Proceedings of the Heterogeneous Computing Workshop*, pages 29–42. IEEE Computer Society Press, 1998.

[7] *http://www.cactuscode.org*.

[8] IMPI Steering Committee. IMPI - interoperable message-passing interface, 1998. `http://impi.nist.gov/IMPI/`.

[9] Karl Czajkowski, Ian Foster, and Carl Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

[10] B.R. de Supinski and N.T. Karonis. Accurately measuring MPI broadcasts in a computational grid. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

[11] for a description of the project, see http://www.aei-potsdam.mpg.de/research/astro/eu_network/index.html.

[12] Graham E. Fagg, Kevin S. London, and Jack J. Dongarra. MPI_Connect managing heterogeneous MPI applications inter operation and process control. In Vassuk Alexandrov and Jack Dongarra, editors, *Recent advances in Parallel Virtual Machine and Message Passing Interface*, volume 1497 of *Lecture Notes in Computer Science*, pages 93–96. Springer, 1998. 5th European PVM/MPI Users' Group Meeting.

[13] I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of SC'98*. ACM Press, 1998.

[14] I. Foster and C. Kesselman. Globus: A toolkit-based grid architecture. In *[15]*, pages 259–278.

[15] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[16] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[17] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91. ACM Press, 1998.

[18] Edgar Gabriel, Michael Resch, Thomas Beisel, and Rainer Keller. Distributed computing in a heterogenous computing environment. In *Proc. EuroPVMMPI'98*. 1998.

[19] A. Goel, D. Steere, C. Pu, and J. Walpole. Adaptive Resource Management Via Modular Feedback Control. Technical Report 99-03, Oregon Graduate Institute, Computer Science and Engineering, January 1999.

[20] N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *Proc. International Parallel and Distributed Processing Symposium*. 2000.

[21] T. Kielmann, R.F.H. Hofman, H.E. Bal, A. Plaat, and R.A.F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99)*, pages 131–140. ACM, May 1999.

[22] T. Kimura and H. Takemiya. Local area metacomputing for multidisciplinary problems: A case study for fluid/structure coupled simulation. In *Proc. Intl. Conf. on Supercomputing*, pages 145–156. 1998.

[23] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. 8th Intl Conf. on Distributed Computing Systems*, pages 104–111, 1988.

[24] B. Lowekamp and A. Beguelin. ECO: Efficient collective operations for communication on heterogeneous networks. In *Proceedings of the 10th International Parallel Processing Symposium*. IEEE Computer Society Press, 1997.

[25] R. Matzner, E. Seidel, S. Shapiro, L. Smarr, W.-M. Suen, S. Teukolsky, and J. Winicour. Geometry of a black hole collision. *Science*, 270:941–947, 1995.

[26] Paul Messina. Distributed supercomputing applications. In *[15]*, pages 55–73.

[27] J. Nieplocha and R. Harrison. Shared memory NUMA programming on the I-WAY. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, pages 432–441. IEEE Computer Society Press, 1996.

[28] P. M. Papadopoulos and G. A. Geist. Wide-area ATM networking for large-scale MPPS. In *SIAM conference on Parallel Processing and Scientific Computing*, 1997.

[29] R. Ponnusamy, J. Saltz, and A. Choudhary. Runtime-compilation techniques for data partitioning and communication schedule reuse. Technical Report CS-TR-3055, Department of Computer Science, University of Maryland, 1993.

[30] Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, and Daniel A. Reed. Autopilot: Adaptive control of distributed applications. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1998.

[31] J. Saltz and M. Chen. Automated problem mapping: The crystal runtime system. In *Proceedings of the Second Hypercube Microprocessors Conference*, Knoxville, TN, September 1986.

[32] Edward Seidel and Wai-Mo Suen. Numerical relativity as a tool for computational astrophysics. *J. Comp. Appl. Math.*, 1999. in press.

[33] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. *Computer Communication Review*, 28(4), 1998.

[34] T. Sheehan, W. Shelton, T. Pratt, P. Papadopoulos, P. LoCascio, and T. Dunigan. Locally self consistent multiple scattering method in a geographically distributed linked MPP environment. *Parallel Computing*, 24, 1998.

[35] Jaspal Subhlok, Peter Lieu, and Bruce Lowekamp. Automatic node selection for high performance applications on networks. In *Proceedings of the Seventh ACM SIGPLAN Symposium on the Principles and Practice of Parallel Programming (PPoPP'99)*, pages 163–172. ACM Press, 1999.

[36] W.-M. Suen. Computational general relativistic astrophysics: the neutron star grand challenge project. In *Proceedings of the Yukawa Conference*, Kyoto, Japan, 1999.

[37] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, Portland, Oregon, 1997. IEEE Press.