# Numerical Relativity in a Distributed Environment

Werner Benger [1,2] Ian Foster [3] Jason Novotny [4] Edward Seidel [1,4,5]
John Shalf [4] Warren Smith [3] Paul Walker [1]

[1] Max-Planck-Institut für Gravitationsphysik, Albert-Einstein-Institut
[2] Konrad-Zuse-Zentrum für Informationstechnik Berlin
[3] Mathematics and Computer Science Division, Argonne National Laboratory
[4] National Center for Supercomputing Applications
[5] Departments of Astronomy and Physics, University of Illinois at Urbana-Champaign

**Abstract**

The Cactus parallel simulation framework provides a modular and extensible set of components for solving relativity problems on parallel computers. In recent work, we have investigated techniques that would enable the execution of Cactus applications in wide area "computational grid" environments. In a first study, we investigated the feasibility of distributing a single simulation across multiple supercomputers, while in a second we studied techniques for reducing communication costs associated with remote visualization and steering. Distributed simulation was achieved by using MPICH-G, an implementation of the Message Passing Interface standard that uses mechanisms provided by the Globus grid toolkit to enable wide area execution. Experiments were performed across SGI Origins and Cray T3Es with geographical separations ranging from hundreds to thousands of kilometers. Total execution time when distributed increased by between 48% and 133%, depending on configuration. We view these results as encouraging as they were obtained with essentially no specialized algorithmic structures in the Cactus application. Work on remote visualization focused on the development of a Cactus module that computes isosurfaces inline with numerical relativity calculations. Experiments demonstrated that this technique can reduce network bandwidth requirements by a factor ranging from 2.5 to 114, depending on the nature of the problem.

## 1 Introduction

Many scientific research communities involve collaborations that span the globe and desire to share their widely distributed resources. Recent advances in networking technology have supported such efforts by enabling the construction of scientific applications that use geographically distributed resources. The numerical simulations community is responding by developing tools that allow better access to widely distributed supercomputing, networking, and storage resources. We discuss our experiences running simulations based on the Cactus portable parallel simulation framework for solving problems in general relativity on geographically distributed resources.

We examine two different issues for running the Cactus code in a distributed environment. The first problem is running a Cactus simulation on multiple parallel computer systems. We are examining this problem because we hope to perform larger simulations than are currently possible on a single parallel computer. We distribute Cactus simulations across multiple supercomputers using the mechanisms provided by the Globus toolkit. In particular, we use Globus mechanisms for authentication, access to remote computer systems, file transfer, and communication. The Cactus code uses MPI for communication and makes use of MPICH-G [9], an MPI implementation layered atop Globus communication mechanisms. These communication mechanisms allow a MPI application to be executed on distributed resources.

We find that without performing any code optimizations, our simulations ran 48% to 100% slower when using an Origin at the National Center for Supercomputing Applications (NCSA) and an Onyx2 at Argonne National Laboratory (ANL). We also ran simulations between Cray T3Es in Germany and a T3E at the San Diego Supercomputing Center (SDSC). Running between the T3Es in Germany resulted in an increase in execution time of 79% to 133%, and running between a German T3E and a T3E at the San Diego Supercomputing Center resulted in an execution time increase of 114% to 186%. We find these results encouraging as they indicate that even in extreme wide area configurations and in the absence of optimizations, communication costs are only around 50 percent of total runtime. We expect optimizations such as message combining, optimized protocols, and computation/communication overlap to reduce communication costs substantially. Hence, it appears likely that distributed execution of Cactus applications will prove to be practical.

The second issue we examine here is remote visualization and steering of the Cactus code. It is common when performing remote visualization to transfer 3-D datasets from a simulation code to a separate visualization code that (for example) constructs isosurfaces. While simple and modular, this approach can result in excessive communication. The modular structure of the Cactus framework makes it relatively easy to construct an isosurface module that performs isosurfacing operations on the same processors as the simulation. Experiments with realistic astrophysics applications demonstrate that this technique can reduces bandwidth requirements associated with visualization by a factor that ranges from 2.5 to 114, depending on the complexity of the data being visualized. This performance improvement and the availability of high-performance wide area networks has allowed scientists in the United States to perform real-time visualization and steering of large Cactus calculations running in Germany.

The next section describes the Cactus framework. Section 3 describes the Globus toolkit, Section 4 describes our experiences and results, and Section 5 contains our conclusions.

## 2 Cactus

Cactus [1] is a modular framework for creating portable parallel finite-difference simulation codes. The Cactus code's primary application has been solving Einstein's equations of gravity [4], including studies involving black holes [5], self-gravitating fields [16], and relativistic hydrodynamics such as the coalescence of two neutron stars (NASA Neutron Star Grand Challenge Project) [2]. The Cactus code was originally developed at the Max-Planck-Institut for Gravitational Physics (Albert-Einstein-Institut) in Potsdam, Germany, by Paul Walker and Joan Massó. Since then, development of Cactus has been taken over by a community of Cactus users, particularly at Washington University in St. Louis in collaboration with the Computational Relativity Group at AEI-Potsdam. The most recent versions of Cactus are completely generalized so it can be reused to support wider range of simulation problems.

Cactus has been designed from the start to support the development efforts of many programmers working on independent projects by making heavy use of the CVS code revision system and a very modular system for adding simulation capabilities. The modules that plug into Cactus are commonly referred to as "thorns." Parallelism and portability are achieved by hiding MPI, the I/O subsystem, and the calling interface under a simple abstraction API. Cactus allows modules to be written completely natively in either C or f77/f90. This makes it considerably easier for physicists to turn existing codes (the majority of which are implemented in Fortran) into modules (thorns) that plug into the Cactus framework. All of the benefits of a modern simulation code are available without requiring major technique changes for the programmers.

The Cactus user community spans the globe, with users in the United States, Germany, Spain, and Hong Kong. Similarly, the computational resources that the Cactus users would like access to are also widely distributed. Managing remote data poses one of the most difficult challenges to using distant resources. It can take longer to transfer the data that results from a simulation than to actually run the simulation code. For the largest problems, the computational resources needed to visualize the resulting data may be nearly as large as supercomputer that was originally used to create the data. In order to meet these challenges, researchers at NCSA and the Rechenzentrum der MPG at Garching Germany [3] created a remote visualization thorn that does the visualization computations in-line with the code and can be steered by using a remote client application. This allows the visualization computations to run with the same degree of parallelism as the simulation codes, resulting in exceptionally fast performance. The geometric information is sent via a network socket to a visualization client application, which runs on a display device such as an ImmersaDesk. The geometry data generally requires considerably less network bandwidth than does the raw data. The user can control various visualization parameters such as the isosurface level from the visualization client. In addition, selected simulation parameters can be controlled by the client so that the simulation can be steered as it proceeds.

## 3 Globus

The supercomputing resources available to Cactus users have different resource reservation procedures, data storage systems, security requirements, and programming environments. The Globus system provides a single set of tools to bring control of these worldwide distributed resources to each user's desktop. Globus consists of a core set of low-level services upon which higher-level services are constructed. The low-level services used in this work are services for authentication, information distribution, resource management, and access to remote data.

The Globus Security Infrastructure (GSI) provides authentication based on public key cryptography. Once a user has been authenticated to Globus, he is then automatically authenticated to all local administrative domains that have installed Globus and that the user has access to. This is accomplished by mapping Globus credentials to local credentials through a security gateway.

The Metacomputing Directory Service [7] (MDS) provides a logically centralized place to store information about entities. The MDS is accessed using the Lightweight Directory Access Protocol [12] (LDAP) and stores information in a hierarchical directory information tree. The directory service contains information about computer systems, networks, users, and so forth. Further, the MDS contains dynamic information such as the number of free nodes on computer systems and the latency and bandwidth between systems. This allows real-time decisions to be made about what resources to use.

The Globus Resource Allocation Manager [6] (GRAM) provides resource management functions to start, monitor, and terminate serial and parallel applications. The purpose of the GRAM is not to replace local scheduling systems but to provide a common interface to the variety of local scheduling systems already in use. GRAMs have been layered atop EASY [13], LSF [15], Condor [14], and other local scheduling systems. In addition, a GRAM that simply does a fork() and exec() is available for unscheduled local resources.

Globus provides access to remote data using the Global Access to Secondary Storage (GASS) component. This component allows applications to access files from various types of servers (currently GASS, ftp, or http) from remote systems. The remote files are accessed by using GASS versions of the standard C I/O functions. Files from these servers are moved to local caches and accessed locally. If remote files are written to, they are moved back to their remote server after they have been closed. The GASS library has been integrated with the GRAMs so that the executable given to the GRAMs can be located remotely. The GRAM will use GASS to transfer the remote executable to the local system and then execute it. This functionality along with the GASS file I/O commands enables location-independent applications.

One high-level Globus service used by Cactus is an implementation of the MPI [11] message-passing standard called MPICH. MPICH is a portable implementation of MPI developed at Argonne National Laboratory and Missouri State University. MPICH is implemented over an abstract communication device, which in turn can be implemented using many different communication mechanisms such as shared memory or a proprietary message-passing protocol. A Globus implementation [9] of this abstract communication device that uses the Nexus [10] communication library, and Globus mechanisms for resource allocation is available.

The Nexus library is a low-level Globus component that is a communication library that provides asynchronous remote procedure calls and threading mechanisms. Nexus supports multimethod communication mechanisms where a process can use multiple communication mechanisms to communicate with other processes. Nexus will automatically select the optimal communication mechanism to use. For example, Nexus will use shared memory to communicate with processes on the same SMP and will use TCP to communicate with processes on a different computer system.

## 4 Experiences and Results

We use Cactus and Globus day-to-day and also in demonstrations such as 1998 meeting of the National Computational Science Alliance (Alliance98) and the SC'98 conference. We have several goals with this work. First, we want to run very large Cactus simulations on distributed computer systems. These are simulations that we cannot perform on a single supercomputer. Second, we want to be able to interactively visualize and steer Cactus simulations while they execute.

### 4.1 Single-System Performance

The first issue we examine is the overhead of using MPICH with the Globus device (MPICH-G) instead of native MPI. Figure 1 compares the execution times of a single iteration of a 128x128x128 Cactus simulation on the ANL Onyx2 when Cactus is compiled with native MPI or MPICH-G. The execution times are nearly the same. Figure 2 shows the execution time of the same problem on a T3E at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). We see that Cactus performance decreases when MPICH-

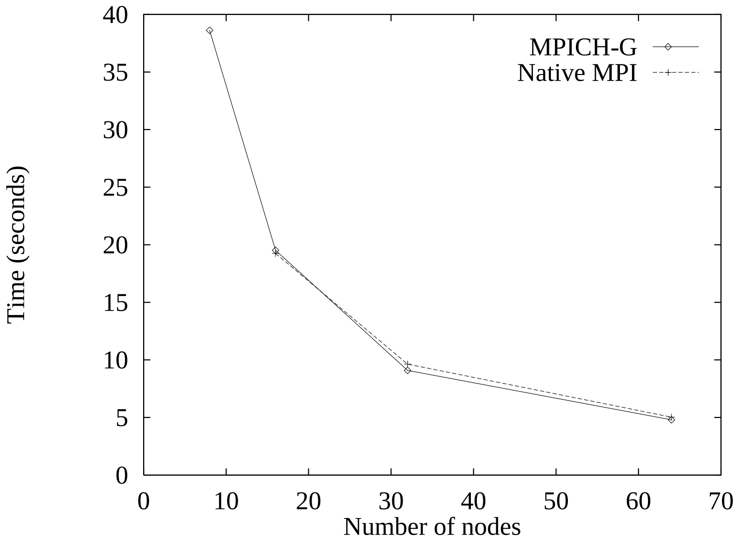


FIG. 1. *Execution time of a single iteration of a 128x128x128 Cactus simulation on the ANL Onyx2 when compiled with native MPI and MPICH-G.*

G is used instead of native MPI on the T3E. This decrease occurs because MPICH-G performance is proportionally worse than native MPI on the T3E than the Onyx2, as we will see next.

Figure 3 shows the performance of the native MPI and MPICH-G implementations on the ANL Onyx2. The data shows that the MPICH-G implementation has lower bandwidth than the native MPI implementation. The native MPI has a bandwidth that ranges from 4.3 Mbits/sec for small messages to 1.6 Gbits/sec for large messages while the MPICH-G implementation has a bandwidth of 1.7 MBits/sec for small messages to 673.7 Mbits/sec for large messages. In addition, the native MPI implementation has a latency of 14.5 microseconds while the MPICH-G implementation has a latency of 37.5 microseconds. Figure 3 also shows the same bandwidths on the ZIB T3E. The native MPI implementation has a bandwidth that ranges from 3.9 Mbits/sec for small messages to 1.3 Gbits/sec for large messages while the MPICH-G implementation has a bandwidth of 280 Kbits/sec for small messages to 539 Mbits/sec for large messages. The latency for native MPI on that T3E is 16.5 microseconds, and the latency is 225 microseconds when using MPICH-G.

The performance of the Cactus code does not suffer when using MPICH-G on the ANL Onyx2 because the Cactus code overlaps computation and communication and can hide the lower communication performance of MPICH-G. The difference in performance between native MPI and MPICH-G on the ZIB T3E, however, is too large to be hidden by the Cactus computations. The lower bandwidth of the MPICH-G implementation is due to an extra copy on the sending and receiving sides. These copies are being removed, and we expect the MPICH-G bandwidth to be close to the native MPI bandwidth.

## 4.2 Distributed Performance

We next examine the performance of the Cactus code when executed on multiple parallel computers. We use several pairs of computers in these experiments to investigate the

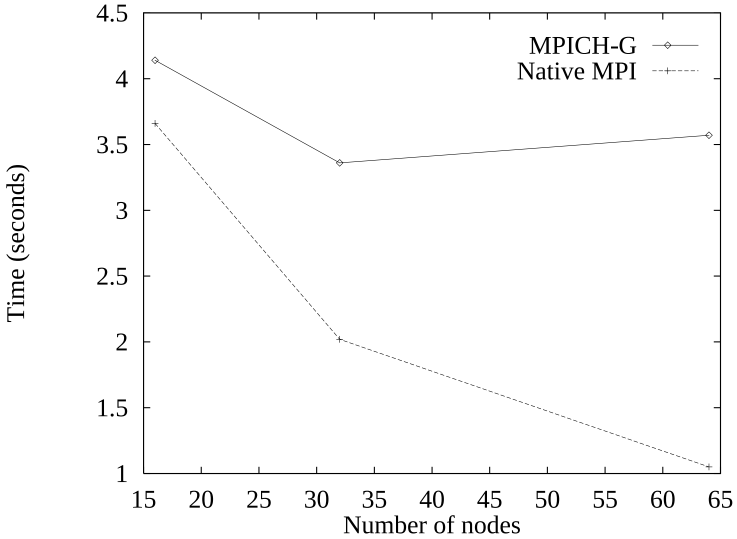


FIG. 2. *Execution time of a single iteration of a 64x64x128 Cactus simulation on the ZIB T3E when compiled with native MPI and MPICH-G.*

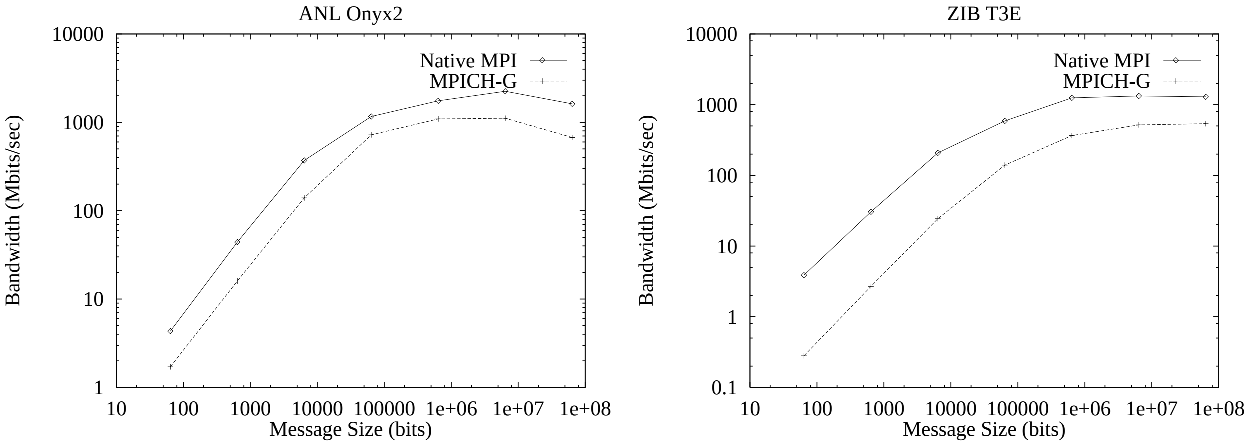


FIG. 3. *Bandwidth of native MPI and MPICH-G.*

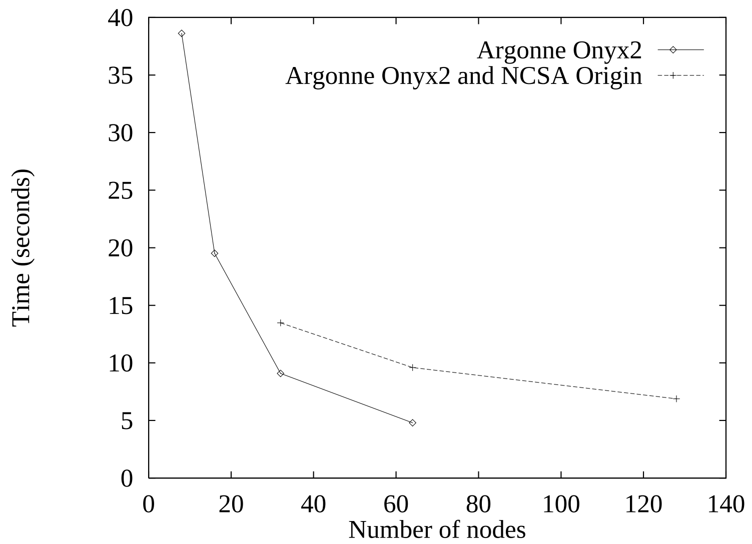


FIG. 4. *Execution time of a single iteration of a 128×128×128 Cactus simulation.*

effect of network performance. Figure 4 shows the execution time of an iteration of a 128×128×128 Cactus simulation when the simulation is run on only the ANL Onyx2 or an equal number of nodes on both the ANL Onyx2 and a NCSA Origin. The data shows that using two machines instead of one increases the execution time of each iteration by 48% to 100%. This decrease in performance is not acceptable in most circumstances, although we can imagine that the larger amount of memory available will allow us to run larger simulations than we can run on a single system.

Figure 5 shows the bandwidth obtained when running an MPI ping-pong test between the ANL Onyx2 and a NCSA Origin. Our MPI measurements show that the latency is 3.75 milliseconds and the bandwidth varies from 17 Kbits/sec for small messages to 24.2 Mbits/sec for large messages. This latency is 250 times worse than the latency between processes on the Onyx2, and the bandwidth is a factor of 30 worse.

Figure 6 shows the average execution time of an iteration of a 64x64x128 Cactus simulation when executed on one or two Cray T3Es. Using the ZIB T3E and the T3E at Rechenzentrum Garching (RZG) increases execution time from 79% to 133%. This increase is larger than the increase seen when using two SGI systems. This larger increase can be attributed to the network between the two T3Es being slower than the network between the two SGI systems. Also note that the execution time for the simulation between ZIB and RZG increased when 64 nodes were used. We believe this was due to competition for the network connection between the two machines. Figure 5 shows that the MPICH-G bandwidth between a T3E at ZIB and a T3E at RZG is 1.5 Kbits/sec for small messages and 4.2 Mbits/sec for large messages. The latency is 42.5 milliseconds. This latency is 10 times higher than the latency between the SGI systems and the bandwidth is 6 times lower.

We also ran simulations between the ZIB T3E and the SDSC T3E over a temporary higher-speed trans-Atlantic connection into STAR-TAP. Simulating on these two systems resulted in an execution time increase of 114% to 186% over using the ZIB T3E. We observed that the network performance between these systems was worse than the network performance between the two German T3Es. Unfortunately, we did not make systematic

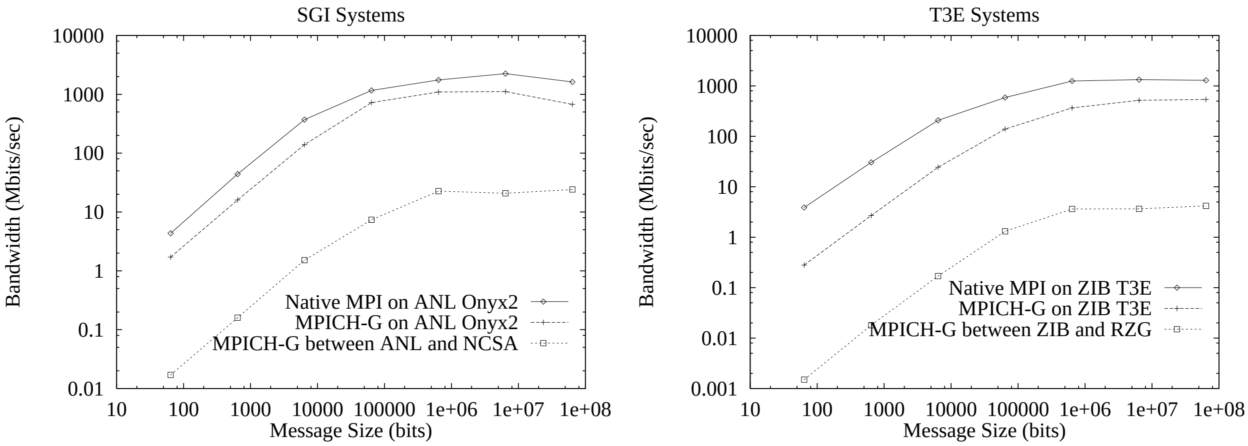


FIG. 5. *Bandwidth between multiple machines when using MPICH-G.*

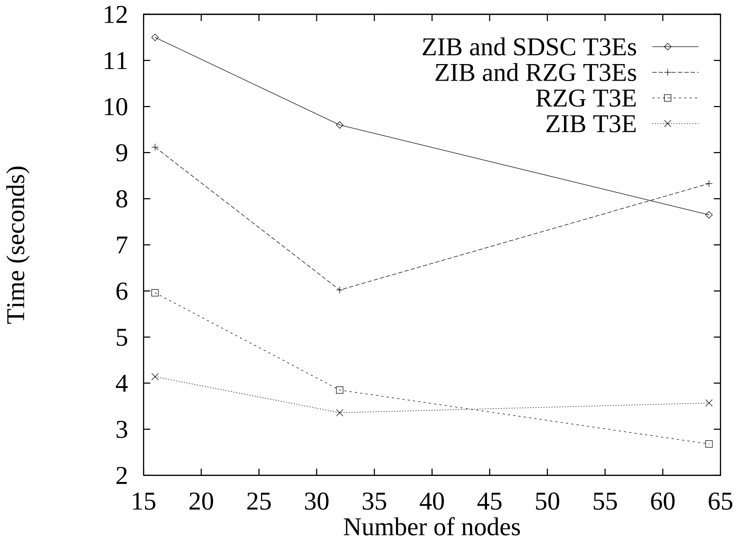


FIG. 6. *Execution time of a single iteration of a 64×64×128 Cactus simulation.*

MPICH-G bandwidth measurements during this time.

The performance results for the Cactus simulations above were obtained without performing any optimizations for a distributed environment. One possible optimization is to combine messages. We instrumented the Cactus code to examine the messages being sent between the processes. We found that most of the messages being sent are small (9300 bytes during a 128×128×128 simulation on 128 nodes). The data in Figure 5 shows that larger messages should be used to achieve higher bandwidth: the large latencies are dominating communication time for smaller messages. We can accomplish this task in two ways. First, we can send the arrays for all of the variables as one message instead of one message per array. Second, if this does not provide sufficiently large messages, we can combine messages from processes on the same machine that are going to the same process

Table 1

*Simulation performance of a 128×128×128 computational domain on a 128-processor Origin2000*

| Number of Processors | Average Simulation Time per Step (sec) | Average Isosurfacing Time per Step (sec) | Percentage of Time Isosurfacing |
|---|---|---|---|
| 1 | 129.00 | 71.3 | 35.6 |
| 2 | 66.70 | 24.1 | 26.6 |
| 4 | 30.20 | 10.6 | 26.3 |
| 8 | 15.60 | 5.28 | 25.3 |
| 16 | 7.10 | 2.63 | 27.0 |
| 32 | 3.57 | 1.37 | 27.7 |
| 64 | 2.03 | 0.77 | 27.5 |

on a different machine.

### 4.3 Remote Visualization and Steering

The last issue we address is performing isosurfacing calculations inline with the parallel simulations. This technique allows the parallel computers to quickly calculate isosurfaces and then send only the geometry data to the visualization device, reducing network traffic. Using this technique, we have found that the current generation of wide-area networks provide sufficient performance for real-time visualization and steering of the Cactus code, even when the MPP and visualization device are separated by the Atlantic ocean.

Our experiments show that computing isosurfaces on the parallel computer for a 128×128×128 problem and then sending them to the visualization device results in 0.14 to 6.4 Mbytes of data being transferred, depending on the complexity of the isosurface. If we send the data at all of the grid points, 16 Mbytes would be transferred. This is a reduction in data transmission by 2.5 to 114.

Table 1 shows a series of benchmark runs on a NCSA Origin. Each of the benchmark runs of the simulation code used 128×128×128 grid where each grid point contains 56 double-precision values, one double value for each of 56 properties being monitored. The data shows that the isosurfacer is as parallel as the simulation code (the percentage of time spent isosurfacing stays relatively constant).

Another approach is to perform the isosurfacing on a separate parallel computer. Members of our team used that approach for a distributed computing demo at SC95 on the I-WAY [8]. The problem with this approach is that the cost of moving the data from the simulation host can far exceed the benefits of parallelizing the visualization computations. We can consider the case of a 128×128×128 simulation in Table 1 over a connection with 15 Mbytes/sec (the availible bandwidth we had at Alliance98). It would have taken 80 seconds to send each step of the simulation as raw data. Sending the geometry would take 1.4 to 30 seconds, and more than makes up for the cost of computing the isosurfaces in-line with the simulation.

## 5 Conclusion

We have found that the hardware and software infrastructure exists to simulate general relativity problems in a distributed computational environment, at some cost in performance. We examine two different issues for running the Cactus code in such a distributed environment. The first issue is running a Cactus simulation on multiple parallel computer systems. Our objective is to perform larger simulations than are currently possible on a single parallel computer. We distribute Cactus simulations across multiple supercomputers using the mechanisms provided by the Globus toolkit. In particular, we use Globus mechanisms for authentication, access to remote computer systems, file transfer, and communication. The Cactus code uses MPI for communication and makes use of an MPI implementation layered atop Globus communication mechanisms. These communication mechanisms allow a MPI application to be executed on distributed resources.

We find that without performing any code optimizations, our simulations ran 48% to 100% slower when using an Origin at the National Center for Supercomputing Applications (NCSA) and an Onyx2 at Argonne National Laboratory (ANL). We also ran simulations between Cray T3Es in Germany and a T3E at the San Diego Supercomputing Center (SDSC). Running between the T3Es in Germany resulted in an increase in execution time of 79% to 133%, and running between a German T3E and a T3E at the San Diego Supercomputing Center resulted in an execution time increase of 114% to 186%. We are very encouraged that we are able to run simulations on parallel computers that are geographically distributed, and we have identified several areas to investigate to improve the performance of Cactus simulations in this environment.

The second issue we examine here is remote visualization and steering of the Cactus code. Cactus is a modular framework and we have implemented a module for this task. This module performs isosurfacing operations on the same parallel computers that are running the simulation and reduces bandwidth requirements between the simulation and visualization components by a factor of 2.5 to 114, depending on the complexity of the data being visualized. This performance improvement and the available high-performance wide area networks allow us to distribute the simulation and visualization components in different parts of the United States and Europe and interactively visualize and steer cactus simulations.

In future work we will address the performance problems when running the simulation code on distributed resources. We are improving the performance of the Globus MPICH device to increase the bandwidth that can be used to transfer data between processes. We are also looking at techniques to improve the performance of Cactus in a distributed environment. One example is combining the many small messages that the Cactus code currently sends into fewer, larger messages between the computer systems. This will help overcome the large latencies that exist between geographically distributed computers.

## Acknowledgments

We gratefully acknowledge the contributions of Manuel Panea, Ulli Schwenn, Hermann Lederer at RZG-Garching, Hans-Christian Hege and Hubert Busch at Konrad-Zuse-Institut, the groups at AEI and Washington University, especially Joan Massó and Gerd Lanfermann at AEI and Mark Miller and Malcolm Tobias at Washington U, Tom Clune at SGI/Cray, Peter Feil and staff at Deutsche Telekom/Berkom, Bill St. Arnaud and staff at Teleglobe/Canarie, Jason Novotny and Meghan Thornton at NLANR, and Brian Toonen at Argonne National Laboratory. The Cactus code is supported by the Albert Einstein Institute and NASA grant NASA-NCCS5-153. The Globus project is supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by NSF, through its PACI program; and by DARPA, as part of its

Quorum program.

## References

[1] The cactus project. http://cactus.aei-potsdam.mpg.de.

[2] GR3D. http://wugrav.wustl.edu/Codes/GR3D.

[3] Sc'97 conference demonstration. http://cactus.aei-potsdam.mpg.de/News/SC97.html.

[4] Les Houches School on Relativistic Astrophysical Source of Gravitational Radiation. 1996.

[5] P. Anninos, K. Camarda, J. Mass E. Seidel, W.M. Suen, and J. Towns. Three Dimensional Numerical Relativity: The Evolution of Black Holes, 1995.

[6] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metasystems. *Lecture Notes on Computer Science*, 1998.

[7] S. Fitzgerald, I. Foster, C. Kesselman, G. vol Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Sixth IEEE International Symposium on High Performance Distributed Computing*, 1997.

[8] I. Foster, R. Gjersten, J. Mass, M. Nardulli, M. Parashar, T. Roy, E. Seidel, J. Shalf, and D. Weber. Computing and Visualizing Einstein's Gravitational Waves across the Metacenter, 1995. http://www.ncsa.uiuc.edu/General/Training/SC95/GII.Apps.html.

[9] I. Foster and N. Karonis. A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In *Proceedings of the ACM/IEEE SC98 Conference*. ACM Press, 1998.

[10] I. Foster, C. Kesselman, and S. Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, (37):70–82, 1996.

[11] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.

[12] Timothy A. Howes and Mark C. Smith. *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.

[13] David A. Lifka. The ANL/IBM SP Scheduling System. *Lecture Notes on Computer Science*, 949:295–303, 1995.

[14] M. Litzkow and M. Livny. Experience with the condor distributed batch system. In *IEEE Workshop on Experimental Distributed Systems*, 1990.

[15] Platform Computing Corporation. *LSF 2.2 User's Guide*, Febuary 1996.

[16] Edward Seidel and Wai-Mo Suen. Formation of Solitonic Stars Through Gravitational Cooling, 1994. Proceedings of the Seventh Marcel Grossman Meeting on General Relativity.