

Cactus-G Toolkit: Supporting Efficient Execution in Heterogeneous Distributed Computing Environments

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

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. Here 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.

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 grossly oversubscribed. 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

*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

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 [4, 25, 26, 7, 27, 33], 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 the rest of this paper we present such a framework, which we term Cactus-G. It builds on two major software systems, namely the Cactus simulation framework [2, 1] 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 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 particular astrophysics application that was the initial reason for Cactus development.

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 has 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.

2.1 Applicable Techniques

A conventional parallel program will not run efficiently (or perhaps at all) in such an environment. The above described Grid characteristics lead to 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 [34].
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 [30, 28].
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 [32].
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 [29, 19].
10. *Network-aware communication algorithms.* We can improve the performance of communication operations by using specialized network-aware algorithms [24, 11, 20, 21, 6].
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, 35].

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 [31], Cactus [8, 2, 1] 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 [3], 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 decomposition 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.

What distinguishes MPICH-G2 from other efforts concerned with message passing in heterogeneous environments (PACX [18], MetaMPI, STAMPI [22], IMPI [9], MPIconnect [12]) is its tight integration with the popular MPICH implementation of MPI and its use of Globus mechanisms for resource allocation or security.

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, differencing-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, parallel or multi-host jobs). An important thorn in our case is the so-called PUGH-thorn, which provides parallelism based on a MPI-implementation such as MPICH-G2. This thorn has been specially 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. The abstraction presented is the MPI programming model; the programmer queries structure and state

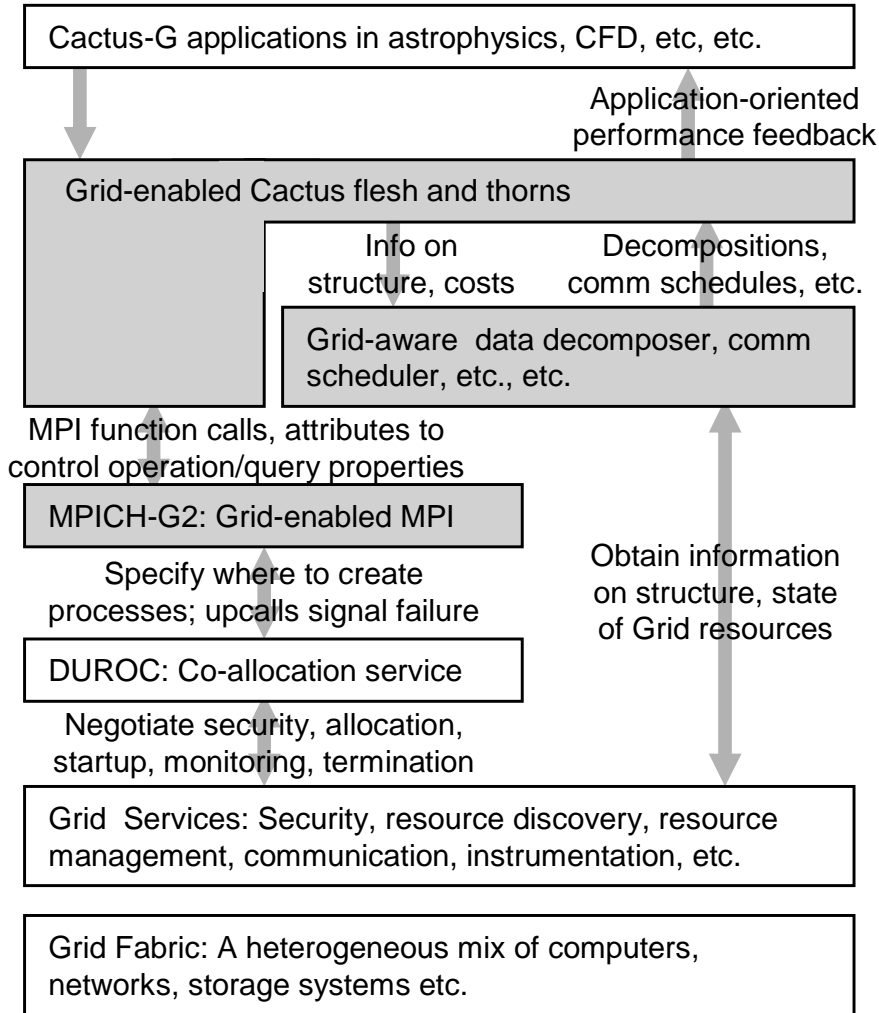


Figure 1: Cactus-G architecture, showing the information and control flows between the different layers

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) [10]. 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 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. We are in the process of implementing a grid-aware communication schedule that will allow to schedule communication and computation to maximize their overlap. 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. We have gathered data that will help develop more automated methods, for example for computing shadow region sizes.

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 execution environment data (number and performance of processors, network performance) and application data (problem size, ghostzone size, etc), allowing us to quantify the sensitivity of overall performance to these parameters.

4 Summary

The distinct characteristics of the Grid environment: highly heterogeneous, dynamic and unreliable have major implications on application development. The paper presents the architecture and current prototype status of Cactus-G a Grid-enabled computational framework that implements advanced techniques designed to hide irrelevant Grid related complexities, and to present application developers with familiar abstractions. We are currently engaged in implementing and evaluating the techniques presented here and will present experimental results in a future version of this paper.

References

- [1] 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.
- [2] 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.

- [3] 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.
- [4] F. Berman. High-performance schedulers. In [15], pages 279–309.
- [5] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96*. ACM Press, 1996.
- [6] 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.
- [7] 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.
- [8] <http://www.cactuscode.org>.
- [9] IMPI Steering Committee. IMPI - interoperable message-passing interface, 1998. <http://impi.nist.gov/IMPI/>.
- [10] 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.
- [11] 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.
- [12] Graham E. Fagg, Kevin S. London, and Jack J. Dongarra. MPLConnect 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. EuroPVMMPPI'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] Paul Messina. Distributed supercomputing applications. In [15], pages 55–73.
- [26] 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.
- [27] 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.
- [28] 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.

- [29] 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.
- [30] J. Saltz and M. Chen. Automated problem mapping: The crystal runtime system. In *Proceedings of the Second Hypercube Microprocessors Conference*, Knoxville, TN, September 1986.
- [31] Edward Seidel and Wai-Mo Suen. Numerical relativity as a tool for computational astrophysics. *J. Comp. Appl. Math.*, 1999. in press.
- [32] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. *Computer Communication Review*, 28(4), 1998.
- [33] 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.
- [34] 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.
- [35] 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.