Performance Predictions for a Numerical Relativity Package in Grid Environments

Matei Ripeanu[1] Adriana Iamnitchi[1] Ian Foster[1,2]

[1]Department of Computer Science

The University of Chicago

[2] Mathematics and Computer Science Division
Argonne National Laboratory

Abstract

The Cactus software package is suitable for a class of scientific applications that are tightly coupled, have regular space decompositions, and involve huge memory and processor time requirements. Cactus has proved to be a valuable tool for astrophysicists, who first initiated its development. However, today's fastest supercomputers are not powerful enough to perform realistic large-scale astrophysics simulations with Cactus. Instead, we must turn to innovative resource environments—in particular, computational Grids—to satisfy this need for computational power. Our paper addresses issues related to the execution of applications such as Cactus in Grid environments. We focus on two types of Grids: a set of geographically distributed supercomputers and a collection of one million Internet-connected workstations. We study the application performance on traditional systems, validate the theoretical results against experimental data, and predict performance in the two new environments.

1 Introduction

Historically, large scientific simulations have been performed exclusively on dedicated supercomputer systems. In many cases, however, a single supercomputer is not capable of simulating a real problem in reasonable time. A solution to this huge need for resources is provided by Grid computing [10], a new field that is distinguished from conventional distributed computing by its focus on the large-scale sharing of Internet-connected resources. Computational Grids are collections of shared resources customized to the needs of their users: they may be collections of resources for data-intensive applications, collections of powerful supercomputers, or simply opportunistic collections of idle workstations. To facilitate access to a potentially very large number of resources, computational Grids provide the necessary tools for resource discovery, resource allocation, security, and system monitoring.

Experiments on large pools of Internet-connected resources have been successful. For example, the recent solution of a decades-old quadratic assignment problem on 1,000 computers distributed across the United States and Europe [4] demonstrated the benefits of using pooled environments. Because of the characteristics of this new environment, however, not all applications seem at first sight to be capable of exploiting it fully. One such example is the class of tightly coupled, synchronous applications, which is sensitive to communication characteristics.

We evaluate the performance of a tightly coupled scientific application, a classic 5-point stencil computation, on two computational Grids: a pool of geographically distributed supercomputers (like those presented in [5]) and a pool of one million workstations [8]. Our goals are to determine what factors limit performance, to analyze the benefits of different algorithm tunings, and to design a performance prediction model.

To better understand our test application (presented in Section 2), we first study its behavior in sequential execution (Section 3.1). We then use the sequential execution measurements (Section 3.2) for building and validating the parallel performance model for two different supercomputer architectures, one based on shared memory and the other on message passing. The performance model, validated on supercomputers, is later adapted to a pool of supercomputers (Section 4.1) and on Internet computing (Section 4.2). We predict the application efficiency in both environments and study the factors that limit performance. We also show that existing application tunings are ineffective with currently available wide-area networks.

2 Application Description: Cactus Code

Cactus [1] was originally developed as a framework for finding numerical solutions to the Einstein equations of general relativity and has since 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 the application design: a central core (*flesh*) connects to application modules (*thorns*) through an extensible interface. Thorns can implement specialized scientific or engineering applications and more standard computational tasks, such as parallel I/O, data distribution, and checkpointing. Parallelism and portability are achieved by hiding system- or library-dependent features under thorn abstraction APIs. 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 libraries. A properly prepared scientific application thorn will work without changes with any of these parallel domain decomposition thorns, which were developed to support new software or hardware technologies.

We analyze an application that simulates the collision of two black holes. As in most astrophysics applications, the computational core of this simulation is a system of dozens of coupled, nonlinear Einstein equations. They are characterized by a high computation/communication ratio: the hyperbolic equations contain thousands of terms to be evaluated, while the only communications required are in computing finite differences for numerical derivatives.

A basic module implements unigrid domain decomposition. It decomposes a global domain over processors and places an overlap region (referred to as a *ghost-zone*) on each processor. For each time step, each processor first updates its local interior grid points, then synchronizes the boundary values. This communication pattern allows us to trade communication costs for replicated computation. Specifically, with increased ghost-zone size, the communication granularity increases significantly at the cost of replicated computation and increased memory usage. Thus, the number of messages (and hence the communication latency costs) is reduced, while the total amount of data exchanged remains constant. We shall see later the costs and benefits of this approach on different architectures.

3 Analysis of the Application Execution on Traditional Architectures

To better understand the characteristics of our application, we analyze its sequential and parallel execution on two supercomputers: a shared-memory machine (Silicon Graphics Origin2000) and a message-passing supercomputer (8-way SMP IBM SP, Ω-switch). The sequential execution analysis is relevant not only for obtaining performance numbers (e.g., execution time per grid point) used later in the parallel model, but also for understanding and avoiding unexpected behavior on multiple processors.

We then build a model for the parallel execution and validate it against real data. We also investigate the influence of ghost-zone size and other parameters on performance. We adapt the parallel model in Section 4 for predicting performance in the two computational Grids considered: a pool of Internet-connected supercomputers and a pool of a million Internet-connected workstations.

3.1 Sequential Execution

From its execution on the two machines, we obtain data about the simulation's memory usage, effective cache usage, and execution time per grid point. This data is then used to validate our theoretical model. We present here only those numeric results relevant for our study:

1. **Memory usage**: Memory requirements are the biggest constraint for running high resolution simulations. We determined the maximum number of grid points that can be handled by one processor with

its associated memory without severe performance penalties. For this, we first found that memory requirements are a linear function of the number of grid points of the locally allocated problem. For a 3D space of $x \times y \times z$ grid points, memory requirements are $Mem_{seq} = 16 + 512 \times 10^{-6}xyz$ (MB).

2. **Execution time**: In our test application the amount of computation per grid point is constant and independent of the location of the grid point in the problem space. Initialization costs are negligible compared with the total computation over a relatively large number of iterations. Hence, the execution time is proportional to the number of grid points: $T_{seq} = t_c xyz$. We validated this model for various values of x, y and z and determined that $t_c = 17\mu s$/grid point on a RISC10000 processor (250 MHz, 32 KB data and 32 KB instruction primary cache, 4MB secondary cache) and $t_c = 24\mu$s/grid point on a Power3 processor (222 MHz, 64 KB shared data and instruction primary cache, 4 MB secondary cache).

3. **Cache usage**: Figure 1 reveals the sensitivity of the application performance to the problem space shape.

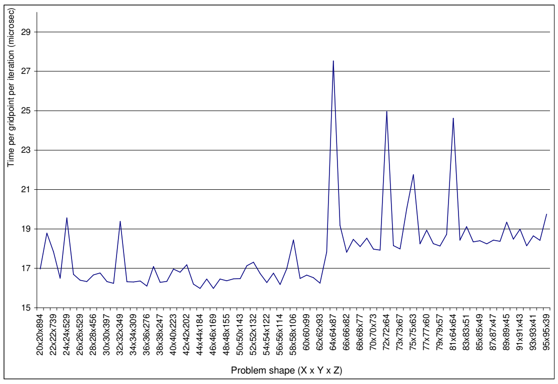

Figure 1: Execution time (in μs) per grid point on RISC10000. The problem size (number of grid points) is (approximately) constant, but the shape varies.

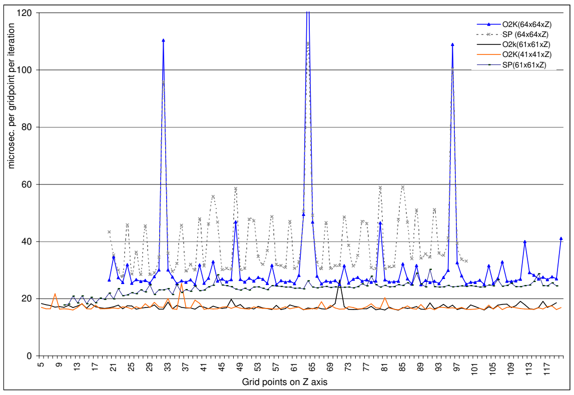

Figure 2: Execution time (in μs) per grid point on RISC10000 and Power3 processors. The x and y dimensions of the problem space are constant, while z varies.

To explore this behavior further, we traced execution time per grid point, varying z and keeping x and y fixed (Figure 2). For most values of x and y the execution time is in the expected range. For some specific x and y values, however, the execution time grows strongly on both architecture. Using hardware counters, we determined that this behavior is generated by cache conflicts: the pattern of larger execution time when z is a multiple of 32 is due to secondary cache misses, while primary cache conflicts generate the smaller peaks when z is a multiple of 8. In our subsequent tests, we tried to avoid configurations that lead to inefficient cache usage.

3.2 Parallel Execution

Communication among processors differentiates the parallel from the sequential algorithm. The use of ghost-zones decreases the number of messages exchanged, but only at the cost of replicated work: the grid points within a ghost-zone are computed multiple times, on different processors.

In the rest of this section we analyze communication costs and execution time. We present the efficiency of the parallel algorithm as demonstrated by the experiments, and we explain the differences from our theoretical model. We observe that on the architectures considered, increasing the size of the ghost-zones does not improve performance.

3.2.1 Communication Costs

The values corresponding to each grid point in the problem space are updated with each iteration based on values of the neighboring grid points. Therefore, neighboring processors communicate their border values at each iteration. In order to reduce the number of messages exchanged, larger chunks of data can be sent at once. A ghost-zone of depth $g \geq 1$ decreases the frequency of messages from 1 message per iteration to 1 message every g iterations. Let us consider $g_x = g_y = g_z = g$ for simplicity, although Cactus does support different ghost-zone sizes for each direction.

For brevity, we analyze only the 3D problem decomposition on a 3D processor topology: each processor is allocated a volume of grid points and has neighbors on all three axes. However, depending on its location, a processor has 6, 5, 4, or 3 neighbors. To model the location of a processor with respect to its neighboring processors, we use the notation $N_i^p \in \{1, 2\}$, which denotes the number of neighbors of the processor p on the i axis ($i \in \{x, y, z\}$). For example, for a corner processor in a 3D topology, $N_x = N_y = N_z = 1$.

Over I iterations, for a ghost-zone size of g and δ bytes sent per grid point, the amount of data (in bytes) sent by a processor is:

$$V = \frac{I}{g} \times L = I\delta(N_x yz + N_y xz + N_z xy). \tag{1}$$

We consider a simple latency/bandwidth communication cost model: the cost of sending a message of length L between any two processors is $t_{msg} = t_s + L \times t_w$, where t_s is the message start-up time and t_w is the cost for sending one byte of data. This model does not account for the complex interconnection network that modern supercomputers use, but we chose it for its simplicity. We shall discuss later the implications of this choice.

During the execution of I iterations, each processor sends and receives $\frac{I}{g}(N_x + N_y + N_z)$ messages. If the link connecting any two neighbors is not shared, the total time spent communicating is

$$T_{comm} = \frac{I}{g}(N_x + N_y + N_z)t_s + 2It_w(N_x yz + N_y xz + N_z xy)\delta \tag{2}$$

3.2.2 Execution Time

Each processor spends its time on *useful* work, communication (2), and *redundant* work. We ignore idle time due to synchronization, assuming perfect load balance (identical processors and identical work load per processor).

Redundant work is the work done on the grid points of the ghost-zones. In every iteration $i \leq I$ replicated work is done on (i modulo g) lines of the ghost-zone. Therefore, in each of the $\frac{I}{g}$ phases replicated work is done for $\sum_{j=1}^{g-1} j(xy+yz+xz) = \frac{g(g-1)}{2}(xy+yz+xz)$ grid points. For 3D decomposition, because each processor has two ghost-zones on each direction, the time spent on replicated work over I iterations is

$$T_r = \frac{I}{g} \times 2t_c \frac{g(g-1)}{2}(xy+yz+xz) = It_c(g-1)(xy+yz+xz). \tag{3}$$

For $x = y = z$ and a regular 3D decomposition, the total overhead time over I iterations is

$$T_{overhead} = T_{comm} + T_r = 6\frac{I}{g}t_s + 12It_w x^2\delta + 3It_c(g-1)x^2. \tag{4}$$

3.2.3 Optimal Ghost-Zone Size

We determined the optimal ghost-zone size for which the overhead introduced by parallelism is minimum. $T_{overhead}$ (4) is minimum when $g_{min} = \frac{1}{x}\sqrt{\frac{2t_s}{t_c}}$. However, if $g \geq 1$, then $x \leq \frac{2t_s}{t_c}$. For a realistic problem size (x in the range 50 to 100 grid points, limited by the available memory) and for t_s, t_c of the two supercomputers considered, this condition is not met. Therefore the execution time increases with increasing ghost-zone size.

We have validated this conclusion by measuring the execution time on eight processors on an Origin2000 and IBM SP with different ghost-zone sizes. The experimental data (Figure 3) confirms our result: execution time grows with ghost-zone size on both supercomputers.

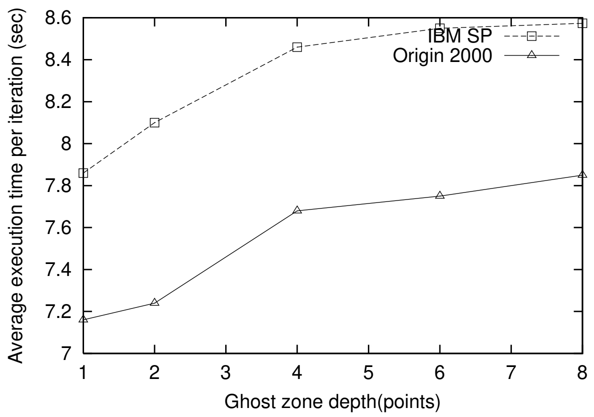

Figure 3: Average time per iteration as a function of ghost-zone size.

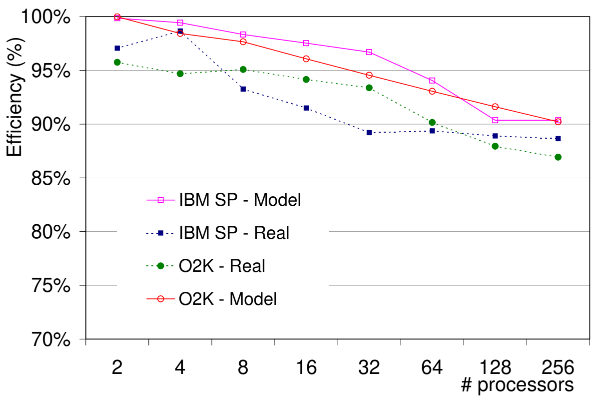

Figure 4: Efficiency: constant problem size per processor, variable number of processors.

The explanation for this result is that latency-related costs are smaller than redundant computation costs. The use of larger ghost-zones to increase performance is justifiable on architectures where $\frac{t_s}{t_c} > 5000$. Since t_c is always on the order of tens of microseconds, ghost-zones with $g > 1$ make sense only in environments with very large (more than 100 ms) latency.

3.2.4 Efficiency

We used efficiency values to validate our performance models. For $g = 1$, P processors, and a problem space of x^3 grid points per processor, maximum efficiency is

$$E_{max} = \frac{T_{seq}}{P \times T_{par}} = \frac{1}{1 + \frac{6t_s}{x^3 t_c} + \frac{12 t_w \delta}{x t_c}}. \tag{5}$$

Equation (5), in which predicted efficiency is independent of the number of processors and therefore of the number of data flows, shows the limitations of the simplified communication model used: the model ignores the fact that links within a supercomputer's interconnection network are shared and assumes that interconnections switches scale linearly. For a more accurate prediction, we used a competition for bandwidth model [7] adapted to the interconnection characteristics of the two supercomputers: we identified shared hardware components, computed the number of competing flows, and used manufacturer's performance specifications. For these experiments we used a memory-constrained model: the problem size per processor remains constant while the number of processors increases up to 256. Figure 4 compares experimental results with our predictions. Although our communication models are simplistic, the test results match the predictions within a 10% range. Other models, such as the hyperbolic model in [15], could lead to more accurate predictions.

4 Predicted Performance in a Grid Environment

We considered two different computational Grids. The first was a collection of supercomputers connected by a Grid middleware infrastructure such as the Globus toolkit [9]. We used this existing computational Grid to validate our approach on performance predictions. The second Grid environment we analyzed was a very large collection of workstations likely to be used in the near future.

4.1 Performance on a Pool of Supercomputers

Our objective was to predict Cactus performance on a pool of supercomputers. [5] presents an experiment where a small number of processors from supercomputers located thousands of miles apart are coupled together to perform a Cactus simulation. We assumed a similar architecture with potentially tens of supercomputers each providing hundreds of processors.

We predicted the application efficiency on this architecture and studied the application and environment characteristics that limit performance. We also investigated ways to increase performance by tuning application parameters and improving the code. For example, we evaluated the benefits of using larger ghost-zone size for intersupercomputer communication to offset latency costs, while maintaining minimal ghost-zone size for intrasupercomputer communication.

We made the following assumptions and notations:

- Greek letters describe functions/values at supercomputer level while the corresponding Latin alphabet letters describe values at processor level. For example, Θ_{comm} is the communication time between supercomputers, while T_{comm} is the communication time between processors.

- Supercomputers are identical, with the same number of processors and computational power. This assumption is realistic because a set of heterogeneous machines can behave in a 1D decomposition as a set of identical processors if loaded proportionally to their computational powers. For this reason we added into Cactus an irregular data distribution mechanism and implemented the additional load balancing algorithms.

- The problem space is decomposed using a 1D decomposition among supercomputers and a 3D decomposition among the processors of each supercomputer. The choice of 1D decomposition was motivated by the limited number of supercomputers that are realistically likely to be simultaneously available to a group in the near future. However, it is easy to extend the model to a 2D or 3D decomposition.

- Supercomputers are connected through identical network links to the Internet. We assumed the traffic between any two supercomputers to be limited by these links and not by the Internet backbone.

- We used the same linear model for communication costs: the cost of sending a message of L bytes from a supercomputer to another is $\theta_s + \theta_w L$ seconds. We assumed that the communication cost of transferring data over a link is independent of the number of concurrent TCP connections.

- Each supercomputer is assigned a grid space of size $X \times Y \times Z$. Since we assumed a 3D regular partition at supercomputer level, each processor was assigned a grid space of size $\frac{X}{\sqrt[3]{P}} \times \frac{Y}{\sqrt[3]{P}} \times \frac{Z}{\sqrt[3]{P}}$ (which is $x \times y \times z$). We assumed S supercomputers each having P processors. We assumed ghost-zone depth G for intersupercomputer communication and ghost-zone depth g for intrasupercomputer communication. Inside supercomputers, the total number of grid points per processor is constant, regardless of the size of the allocated ghost-zone. This ensures load balance at the processor level.

To build the performance model for the architecture described above, we assumed that each supercomputer is a computational entity that obeys the performance model described in Section 3.2. Hence, we modeled a supercomputer as a "faster" processor with a "big" associated memory. This *super*processor is characterized by the time needed to update one grid point θ_c (the equivalent of t_c presented in Section 3.2).

Execution Time. Using the model described in Section 3.2.2, we calculated the time spent for useful work on a problem of size $X \times Y \times Z$ on a supercomputer with P processors as

$$\Theta_{seq} = \theta_c XYZ. \tag{6}$$

The same amount of time is spent by each processor solving its part of the problem $\frac{X}{\sqrt[3]{P}} \times \frac{Y}{\sqrt[3]{P}} \times \frac{Z}{\sqrt[3]{P}}$ but working in parallel with efficiency E:

$$\Theta_{seq} = \frac{t_c}{E} \times \frac{X}{\sqrt[3]{P}} \frac{Y}{\sqrt[3]{P}} \frac{Z}{\sqrt[3]{P}}. \tag{7}$$

From (6) and (7), we have

$$\theta_c = \frac{t_c}{EP}. \tag{8}$$

Communication Costs. For each message sent from a supercomputer to another, communication time is $\Theta_{msg} = \theta_s + \theta_w L$. Over I iterations there are $\frac{I}{G}$ communication phases, in which each supercomputer sends two messages of $L = GXY\delta$ bytes each. Incoming and outgoing messages share the communication link. Therefore, the time spent communicating is

$$\Theta_{comm} = \frac{I}{G} \times 2(\theta_s + 2\theta_w GXY\delta) = \frac{2I\theta_s}{G} + 4I\theta_w XY\delta. \tag{9}$$

Replicated Work. Consider that each processor has ghost-zones of depth g and each supercomputer has ghost-zones of depth G. This is meant to accommodate the variation in communication costs (intersupercomputer vs. intrasupercomputer). Since replicated time on the processor ghost-zones (of size g) is already included in the model through the efficiency value E, the time spent by each supercomputer on doing replicated work is a function of $(G - g)$. In each iteration replicated work is done on $(G - g)XY$ grid points. Each supercomputer has at most two ghost-zones. The total time spent doing replicated work over I iterations is therefore

$$\Theta_r = 2I\theta_c(G - g)XY \tag{10}$$

For S identical supercomputers with P processors each and a problem space of $SX \times Y \times Z$ grid points, each supercomputer has to solve a $X \times Y \times Z$ grid point problem. Total execution time for I iterations is

$$\Theta_{par} = \Theta_{seq} + \Theta_{comm} + \Theta_r = I\theta_c XYZ + \frac{2\theta_s I}{G} + 4I\theta_w XY\delta + 2\theta_c I(G - g)XY \tag{11}$$

4.1.1 Optimal Ghost-Zone Size

From (11) maximum efficiency E' is obtained for $G_{opt} = \frac{1}{x}\sqrt{\frac{\theta_s E \sqrt[3]{P}}{t_c}} = \frac{1}{x\sqrt[3]{P}}\sqrt{\frac{\theta_s}{\theta_c}}$. This validates our intuition that larger intersupercomputer communication latency requires larger ghost-zones while slower processors or larger problems would require smaller ghost-zones. For $\theta_s = 35ms$ (usual value for coast-to-coast links), supercomputer efficiency $E = 90\%$, $t_c \simeq 20\mu$s, $P = 1024$ and $x = y = z = 60$ grid points per processor, the maximum overall efficiency is obtained for $G_{opt} = 2$.

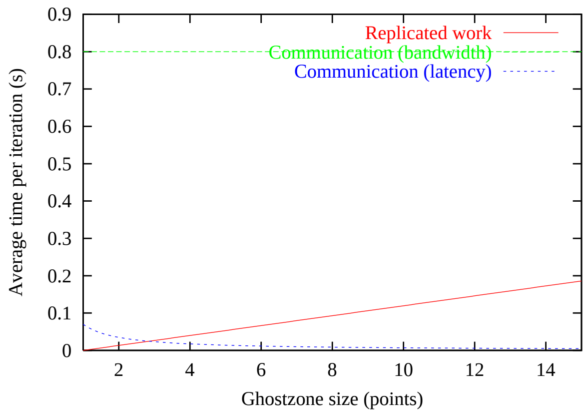

Figure 5: Components of parallel overhead time. Gigabit links connect supercomputers to the Internet.

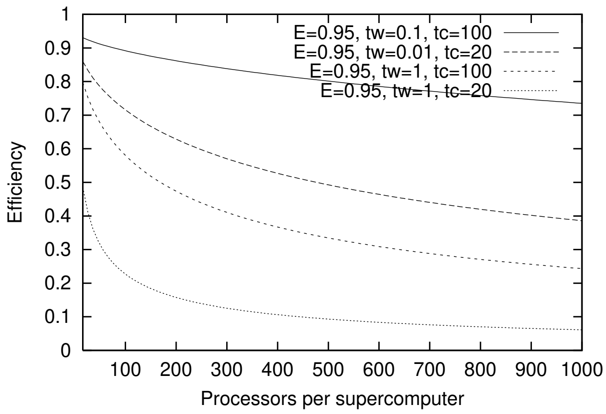

Figure 6: Cactus achievable efficiency on a pool of supercomputers with in-place networks.

As Figure 5 shows, the overhead due to replicated work (Θ_r) or to latency is small when compared to the overhead introduced by network bandwidth. In fact, the current Cactus implementation does not allow $G \neq g$. We introduced a different ghost-zone size for inter-computer communication to evaluate its potential benefits for the overall performance. The results presented suggest that using different ghost-zones sizes for supercomputer communication does increase overall efficiency. However, Figure 5 shows that more than 95% overhead is due to limited network bandwidth even in the optimistic scenario when each supercomputer has 1 gigabit connection to the outside world. We would need links one order of magnitude faster to obtain significant savings by allowing deeper ghost-zones.

4.1.2 Predicted Efficiency

In our model the overall efficiency is the product of the efficiency of a single supercomputer and the efficiency of the collection of supercomputers. For $G = g = 1$, we have

$$E_{overall} = E\varepsilon = E\frac{\Theta_{seq}}{S \times \Theta_{par}} = 1/(\frac{1}{E} + \frac{2\theta_s}{xyzt_c} + \frac{4\theta_w \delta \sqrt[3]{P^2}}{zt_c}). \quad (12)$$

We validated (12) with two experiments executed on supercomputers across the United States. The experimental setup, including details on the middleware infrastructure used (Globus and MPICH-G2), is presented in [3]. The first experiment used two supercomputers with up to 64 processors each. The second, large-scale experiment involved four supercomputers of a total of 1,500 processors (described in [2]). In both cases the results measured were at most 15% less than our predictions. We believe this difference is due mostly to the highly variable behavior of the wide area network.

In Figure 6 we consider the existing network infrastructure in our test bed ($\theta_s = 35$ ms and $\theta_w = 1\mu$s) and show the variation of the predicted efficiency $E_{overall}$ with the number of processors. We also plot efficiency for an application configuration that is five times more computationally intensive ($t_c = 100\mu$s) and for an application-observed bandwidth of 10 MB/s ($\theta_w = 0.1\mu$s). Although in our test we have not benefited from the 10 MB/s application-observed bandwidth, this is easily achievable with currently deployed networks. It is interesting to note that efficiency as high as 83% could be obtained if all supercomputers were connected to the Internet by using gigabit links.

From Equation (12) above and from Figure 6, we observe that overall efficiency $E_{overall}$ increases with the sequential execution time per grid point and with the decrease in communication costs. Even with new, more computationally demanding numerical algorithms, since the processors are increasingly powerful, we believe that a real improvement in efficiency is possible only through efficient use of "fatter" network pipes.

4.2 Internet Computing

There are over 200 million PCs around the world, many as powerful as early 1990s supercomputers. Every large institution has hundreds or thousands of such systems. Internet computing [8] is motivated by the observation that at any moment millions of these computers are idle. With this assumption, using computer cycles and data storage on these computers becomes virtually free provided satisfactory middleware infrastructure and network connectivity exist.

Computational Grids provide the middleware infrastructure: dependable, consistent, and pervasive access to underlying resources. The continuous growth of the Internet is going to provide the necessary connectivity. Internet demand has been doubling each year for more than a decade now. This has caused backbone network capacity to grow at an even faster rate [13]. Moreover, it is estimated that in the near future we will witness an explosion in network capacity [6]. Optical technologies that are driving this transition will also transform the physical structure of the Internet from one based on backbone networks carrying thousands of (virtual) network flows through dozens of large pipes to one based on backbone "clouds" consisting of thousands of possible optical paths, each with the capacity to carry traffic of multigigabits per second.

The available middleware infrastructure and the soon-to-be-available network connectivity bring the vision of a general-purpose 1 million processor Internet computing system (*megacomputer*) closer to reality. Megacomputers might be the world's first petaops (10^{15} FLOPS) computing systems. For application such as Cactus, however, increased computational power might prove to be less significant than aggregating the memory of millions of computers that will allow solving problems of unprecedented scale and resolution.

The design of the megacomputer is indeed challenging because it requires the synthesis of Internet protocols, high-performance computing, and commodity software technologies in a scalable, reliable computing system. We do not investigate these issues here, but we do estimate the performance Cactus could obtain in such an environment.

We considered the processor space divided in "clusters": groups of computers on the same gigabit local-area or campus network. They might also be PCs using DSL (ADSL, HDSL) or cable modem technologies within the same geographical area (and thus probably using the same provider's POP). We assumed that communication within a cluster is low delay, high bandwidth. A shared hub allows communication with other clusters. We imagined a cluster to have hundreds to thousands of machines. To minimize communication, we used a 3D decomposition among clusters. Even with this problem decomposition aimed at minimizing intercluster communication (and thus the number of flows that traverse cluster boundaries), the networking infrastructure must deal with an enormous number of flows. For a cluster of size P there are $6\sqrt[3]{P^2}$ communication flows going out. Given TCP's limited ability to fairly and efficiently deal with large number of simultaneous flows ([12, 14]) some mending is needed at the network transport level. Possible solutions are to use TCP concentrator nodes for intercluster communication, use an improved TCP protocol (along the lines of RFC2140), or simply replace TCP with a future transport protocol.

We analyzed the performance of a megacomputer using the same model as in preceding sections. Instead of describing the whole process in detail, we summarize our conclusions:

- Efficiency of 30–35% can be obtained in these environments even without modifying Cactus's tightly coupled computational pattern. This might seem low, but considering the huge aggregated computational power of this environment, the result is more than one order of magnitude larger than the fastest supercomputer available today. We assumed 1,000 clusters with 1,000 machines each. We considered a two-level hierarchy of gigabit LANs within a cluster and nonshared OC48 links among clusters. We picked conservative values for the application's uniprocessor execution rates (100 MFLOPS) and grid-point processing time (20 μs).

- The application is extremely sensitive to communication costs. Hence, simple improvements such as overlapping computation and communication will bring up to 100% improvements in efficiency. Communication sensitivity also means that we should consider network performance issues when making load-balancing decisions.

To conclude, we estimate that Cactus could run at an execution rate of 35 TFLOPS on a megacomputer. This is 25 times faster than the best execution rate achievable now [11] on a supercomputer. Based on

Moore's law (which still holds, if we use the annual Gordon Bell awards to judge the power of the fastest supercomputer worldwide) it will take seven years to have a supercomputer as powerful as the megacomputer we imagined. Certainly, the assumptions we made about network connectivity and the omnipresence of Grid environments will become reality well before then and offer a more powerful and low-cost alternative to supercomputers.

5 Summary

We provided a detailed performance model of a scientific application, a typical finite differences algorithm, and carefully validated it on two architectures: an IBM SP machine and an SGI Origin 2000.

We adapted our performance model for two computational Grids: an Internet-connected collection of supercomputers and a megacomputer. We investigated the benefits of increasing ghost-zone depth for increasing performance, and we determined that these are insignificant because of the high bandwidth-related communication costs. We also determined that the limiting factor for efficiency is network bandwidth, which is going to improve dramatically over the next few years.

Finally, we predicted Cactus performance in an Internet computing environment—one million Internet-connected workstations. With better network connectivity than in place today and using computational Grids, scientists will shortly have a powerful computational platform at a very low cost.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 9975020.

References

[1] G. Allen, W. Benger, C. Hege, J. Massó, A. Merzky, T. Radke, E. Seidel, J. Shalf, *Solving Einstein's Equations on Supercomputers*, IEEE Computer 32(12), 1999.

[2] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, B. Toonen. *Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus.* Supercomputing, 2001.

[3] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, B. Toonen. *Cactus-G toolkit: Supporting efficient execution in heterogeneous distributed computing environments*, In *Proceedings of 4th Globus Retreat*, July 2000.

[4] K. Anstreicher, N. Brixius, J.P. Goux, J. Linderoth, *Solving Large Quadratic Assignment Problems on Computational Grids*, 17th International Symposium on Mathematical Programming, Atlanta, GA, August 2000.

[5] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, P. Walker, *Numerical Relativity in a Distributed Environment*, Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, March 1999.

[6] C. Catlett, I. Foster, *The network meets the computer: Architectural implications of unlimited bandwidth*, Workshop on New Visions for Large-Scale Networks: Research and Applications, 2001.

[7] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995.

[8] I. Foster, *Internet Computing and the Emerging Grid*, Nature 408(6815), 2000.

[9] I. Foster, C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputing Applications, 11(2), 1997.

[10] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, CA, 1999.

[11] J. Makino, T. Fukushige, M. Koga, *A 1.349 TFLOPS simulation of black holes in a galactic center in GRAPE-6*, In Supercomputing 2000.

[12] R. Morris, *TCP behavior with many flows*, IEEE International Conference on Network Protocols, Atlanta, Georgia, October, 1997.

[13] A. Odlyzko, *Internet growth: Myth and reality, use and abuse*, Information Impacts Magazine, November, 2000.

[14] L. Qiu, Y. Zhang, S. Keshav, *On Individual and Aggregate TCP Performance*, Proceedings of 7th International Conference on Network Protocols, Toronto, Canada, 1999.

[15] I. Stoica, F. Sultan, D. Keyes, *A hyperbolic model for communications in layered parallel processing environments*, Journal of Parallel and Distributed Computing, 39(1):29–45, 1996.