Performance Predictions for a Numerical Relativity Package in Grid Environments

Matei Ripeanu Adriana Iamnitchi

Department of Computer Science
The University of Chicago

Abstract

The Cactus software is representative for a whole class of scientific applications: tightly coupled, regular space decomposition, huge memory and processor time requirements. It proved to be a valuable tool for astrophysicists, who first initiated its development. However, today's fastest supercomputers are not powerful enough to solve realistically large astrophysics simulations with Cactus. The emergence of innovative resource environments like computational Grids satisfies this need for computational power. Our paper addresses issues related to the execution of applications like Cactus in Grid environments. We focus on two types of Grids: a set of geographically distributed supercomputers and a collection of the scale 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, a single supercomputer is not capable of simulating a real size problem in reasonable time. A solution to this huge need for resources is provided by Grid computing [11], a new field that distinguishes from conventional distributed computing by its focus on large-scale sharing of Internet-connected resources. Computational Grids are collections of shared resources specialized to the needs of their users: they may be collections of resources for data intensive applications, or collections of powerful supercomputers, or simply opportunistic collections of idle workstations. In addition 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 mechanisms.

Experiments on large pools of Internet-connected resources have been successful. For example, [3] demonstrated the potential efficiency of these environments by solving an optimization problem (stated in 1968 and unsolved since then) on a pool of 1000 computers distributed across the US and Europe. However, because of the characteristics of this new environment, not all applications are obviously capable of fully exploiting it. One such example is the class of tightly coupled, synchronous applications, which are 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 [4]) and a pool of one million workstations [9]. Our goals are to determine what factors limit performance, to analyze the benefits of potential algorithm tunings, and to design a performance prediction model.

To better understand our test application (presented in Section 2), we study its sequential behavior (Section 3.1) and measure its performance. These results are used in Section 3.2 for building and validating the parallel performance model for two different supercomputer architectures: a shared memory and a message passing based machine. The performance model, validated on supercomputers, is later adapted to a pool of supercomputers (Section 4.1). We predict the application efficiency in this environment and study the factors that limit performance. We also show that the existent application tunings are ineffective with currently available wide-area networks. In Section 4.2 we move our application performance discussion to Internet computing.

2 Application Description: Cactus Code

Cactus [2] was originally developed as a framework for numerical solutions of Einstein's equations and 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 the application design: a central core (*flesh*) connects to application modules (*thorns*) through an extensible interface. Thorns can implement custom developed scientific or engineering applications, such as the simulation of black holes collision, or standard computational tools, such as parallel I/O, data distribution, or checkpointing. Parallelism and portability are achieved by hiding system or library dependent features under thorn abstraction APIs. For example, the abstraction of parallelism allows plugging in different thorns that implement an MPI-based unigrid domain decomposition with very general ghost zone capabilities, 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's 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 modules (thorns) implements unigrid domain decomposition. It decomposes a global domain over processors and places an overlap region (referred to as *ghost-zone*) on each processor. For each time step, each processor updates first its local interior grid points, then synchronizes the boundary values. This communication pattern allows us to trade communication costs for replicated computation. Specifically, by increasing the ghost-zone size, the communication granularity increases significantly at the cost of replicated computation and increased memory usage. This reduces the number of messages (and therefore the communication latency costs) 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

In order to better understand the characteristics of our application, we analyze its sequential and parallel execution on two supercomputers: a shared memory machine (Silicon Graphics Origin 2000) and a message passing based supercomputer (IBM SP). We learn about its memory usage, execution time and communication pattern. We study how the ghost-zone size influences performance and what parameters of the environment limit it. We build a performance model for the parallel execution, validate it against real data, and use it in the next section to predict performance in the two computational Grids considered: a pool of Internet-connected supercomputers and one million of Internet-connected workstations.

3.1 Sequential Execution

From the sequential execution we learned about memory usage, execution time per grid point and effective cache usage. In all cases we validated our theoretical model against executions on two processors: RISC10000 and Power3. We present here only the numeric results relevant for our study:

1. **Memory usage**: Memory requirements are the biggest constraint for running high resolution simulations. We determine the maximum number of grid points that can be handled by one processor with its associated memory without severe performance penalties. For this, we determined 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 to the total computation over a relatively large number of iterations. Hence, the execution time is proportional with the number of grid points: $T_{seq} = t_c xyz$. We validated this model for various values of x, y and z and we determined $t_c = 17\mu s$/grid point on a RISC 10000 processor and $t_c = 24\mu s$/grid point on a Power3 processor.

   We measured using hardware counters approximately 800 floating point operations per grid point per iteration. The resulting execution rates are 47 MFLOPS on RISC10000, and 33 MFLOPS on Power3. The difference between these values and the peak performance listed in processors' specifications (more than 400MFLOPS) suggest that there is room for performance improvements. Considering only peak FLOPS rates, we would expect Power3 processor perform better than R10000. However, our test application runs faster on R10000 due to about two times faster memory access (`llcbenchmark` measurements show over 200 MB/s as compared to about 100 MB/s for R10000). This emphasizes that efficient cache usage is essential for achieving a high FLOPS rate.

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

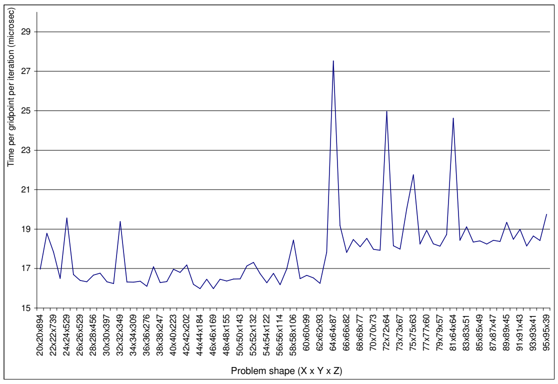

Figure 1: Execution time (in μs) per grid point on RISC10000. The problem size (number of grid points) is (approximately) constant but the shape varies.

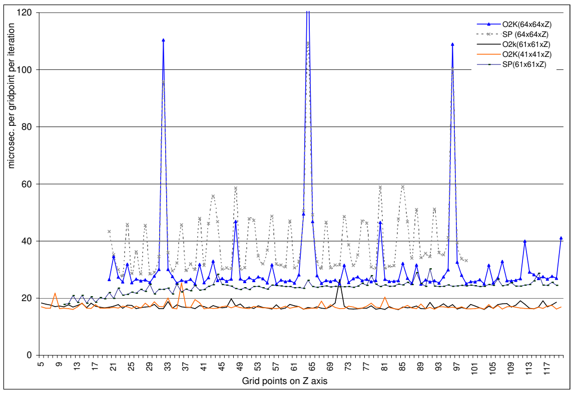

Figure 2: Execution time (in μs) per grid point on RISC10000 and Power3 processors. x and y dimensions of the problem space are constant while z varies.

To explore further this behavior, we traced execution time per grid point varying z and maintaining x and y constant (Figure 2). For most of the values of x and y the execution time is in the expected range. However, for some specific x and y values the execution time grows strongly on both architectures. 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. We tried to avoid in our following tests configurations that lead inefficient cache usage.

3.2 Parallel Execution

Communication among processors is what differentiates the parallel algorithm. The use of ghost-zones decreases the number of messages exchanged for the cost of replicated work: the grid points within a ghost-zone are computed twice, on different processors.

In the remaining of this section we analyze communication costs and execution time. We present the efficiency of the parallel algorithm as resulted from experiments and explain the differences from our theoretical model. We also 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 each iteration based on values of the neighboring grid points. Therefore, neighboring processors communicate their border values every 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 the 3 axis. 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 $N_i^p \in \{1, 2\}$ that 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. Assuming 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).

Replicated 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 2 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 determine 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, because $g \geq 1$ 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 the increasing of the ghost-zone size.

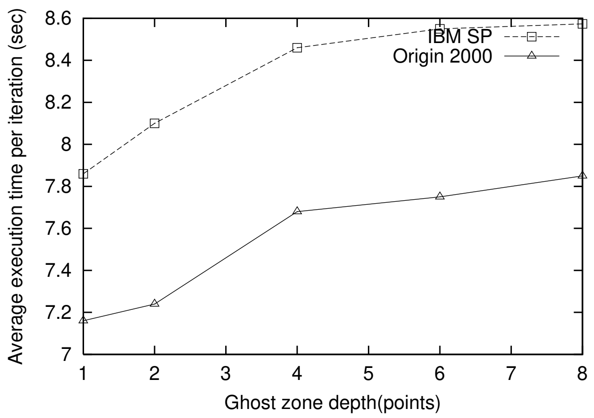

Figure 3: Average time per iteration as a function of ghost-zone size.

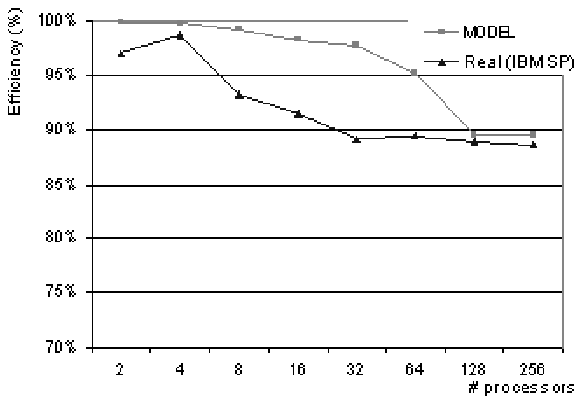

Figure 4: Efficiency: problem size per processor constant is kept constant, variable number of processors.

We have validated this theoretical conclusion by measuring execution time on 8 processors on Origin2000 and IBM SP with different ghost-zone sizes. The experimental data (Figure 3) confirm our result: the execution time grows with ghost-zone size on both supercomputers.

The intuition behind 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 in the order of tens of μs, ghost-zones with $g > 1$ make sense only in environments with very large (more than $100ms$) latency.

3.2.4 Efficiency

For $g = 1$, P processors, and a problem space of x^3 grid points per processor, maximum efficiency is:

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

It is here that the simplified communication model used shows its limitations. The model ignores the fact that links within supercomputer's interconnection network are shared. It also assumes that the interconnection switch scales linearly. More accurate models (logP [7, 1] or hyperbolic [16]) should be used for better prediction. A simpler approach that usually yields acceptable results is to use a competition for bandwidth model [8]. For the IBM SP machine we used, 8 processors that form a *node* share the link from that node to the switch ([6]). Knowing the mapping of the application's logical topology to physical processors allows us to compute the number of flows that share these links for each configuration. Thus, we can compute the bandwidth per flow (t_w) as the result of dividing link bandwidth (960MB/s) by the number of flows.

To measure the achieved efficiency (Figure 4) we used a memory constrained model: the problem size per processor remains constant while the number of processors increases up to 256. Figure 4 presents experimental results and our model prediction. For a large number of processors the model is close to measured data. For a smaller number, however, the limiting performance factor is not the shared node-to-switch link but perhaps some shared channel inside the nodes. We do not have enough information about the IBM SP node architecture to further model it.

4 Predicted Performance in a Grid Environment

We consider two different computational Grids. The first is a collection of supercomputers connected by a Grid software infrastructure like the middleware provided by the Globus toolkit [10]. This computational Grid already exists and we used it to validate our approach on performance predictions. The second Grid environment we analyze is a very large collection of workstations likely to be used in the near future.

4.1 Performance on a Pool of Supercomputers

Our objective is to predict Cactus performance on a pool of supercomputers. [4] presents an experiment where a small number of processors from supercomputers located 1000s miles apart are coupled together to perform a Cactus simulation. We imagine a similar architecture with potentially tens of supercomputers each providing hundreds of processors.

We predict the application efficiency on this architecture and study the application and environment characteristics that limit performance. We also investigate ways to increase performance by tuning application parameters and improving the code. For example, we evaluate the benefits of using larger ghost-zone size for inter-supercomputer communication to offset latency costs, while maintaining minimal ghost-zone size for intra-supercomputer communication.

We make the following assumptions and notations:

- Greek letters describe functions/values at supercomputer level while the corresponding English 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: the same number of processors with the same computational power. This assumption is realistic since 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 have 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 for 1D decomposition is 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 assume the traffic between any two supercomputers is limited by these links and not by the Internet backbone.
- We use 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 assume 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 assume a 3D regular partition at supercomputer level, each processor will be 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 assume S supercomputers, each having P processors. We assume ghost-zone depth (G) for inter-supercomputer communication and ghost-zone depth g for intra-supercomputer communication.

To build the performance model for the architecture described above, we consider that each supercomputer is a computational entity that obeys the performance model described in Section 3.2. Hence, we model a supercomputer as a 'faster' processor with a 'big' memory associated. This *super*processor will be 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, the time spent for useful work on a problem of size $X \times Y \times Z$ on a supercomputer with P processors is:

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

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):

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

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 (inter-supercomputer vs. intra-supercomputer). 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 inter-supercomputer 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 $G_{opt} = 2$.

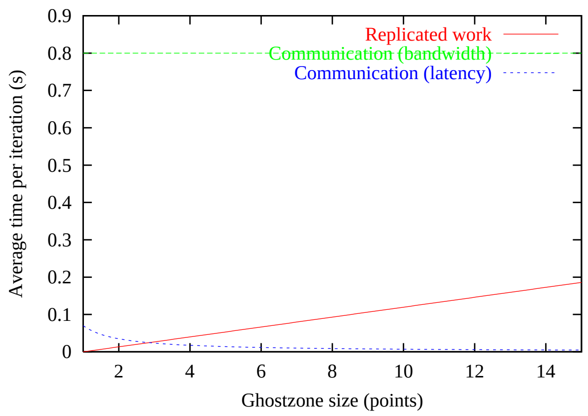

Figure 5: Components of parallel overhead time. Gigabit links connect supercomputers to Internet.

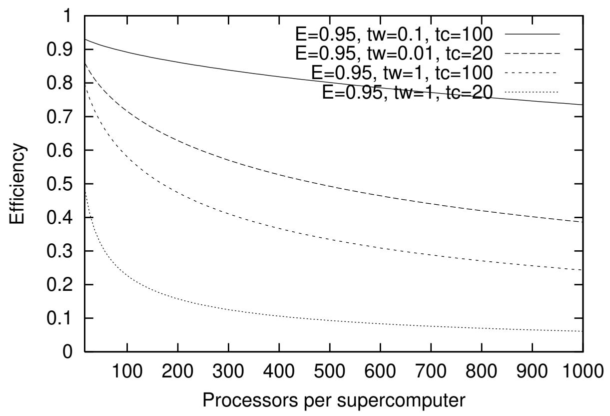

Figure 6: Cactus achievable efficiency on a pool of supercomputers with in-place networks.

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 a Gigabit connection to the outside world. We would need one order of magnitude faster links 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:

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

We validated (12) with small scale runs between two supercomputers across the US. We used up to 64 processors on each machine and obtained results within 10% of our predictions. We believe the difference between our test results and our prediction is due to the highly variable behavior of the wide area network.

In Figure 6 we consider the existing network infrastructure in our test-bed ($\theta_s = 35ms$ 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 5 times more computationally intensive ($t_c = 100\mu s$) and for an application-observed bandwidth of 10MB/s ($\theta_w = 0.1\mu s$). Although in our test we have not benefitted from 10MB/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 Internet using Gigabit links.

From the 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 400 million PCs around the world, many as powerful as early 1990s supercomputers. Every large institution has hundreds or thousands of such systems. Internet Computing [9] 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 (1) middleware infrastructure and (2) network connectivity.

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 driven backbone network capacity to grow of an even faster rate [14]. Moreover, it is estimated that in the near future we will witness an explosion in network capacity [5]. 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 multi-Gb/s traffic.

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 Peta-op (10^{15} FLOPS) computing systems. However, for application like Cactus, increased computational power might prove to be less significant than aggregating the memory of millions of computers which 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 here these issues but we estimate the performance Cactus could obtain in such an environment.

We consider 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 assume that communication within a cluster is low delay, high bandwidth. A shared hub allows communication with other clusters. We imagine the size of a cluster in the order of 100s to 1000s machines. To minimize communication, we use a 3D decomposition among clusters. Even with this problem decomposition aimed at minimizing inter-cluster communication (and thus the number of flows that traverse cluster boundaries), the networking infrastructure will have to deal with an enormous number of flows. For a cluster of size P there will be $6\sqrt[3]{P^2}$ communication flows going out. Given TCP's limited ability to fairly and efficiently deal with large number of simultaneous flows ([13, 15]) some mending is needed at network transport level. Possible solutions are: use TCP concentrator nodes for inter-cluster communication, use an improved TCP protocol (along the lines of RFC2140), or simply replace TCP with a future transport protocol.

We followed the same steps in drawing a performance model as in previous sections. We shall not describe in detail the whole process but we summarize our conclusions:

- Efficiency of 30-35% can be obtained in these environments even without modifying Cactus' 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 1000 clusters with 1000 machines each. We considered a 2-level hierarchy of Gigabit LANs within a cluster and non-shared OC48 links among clusters. We picked rather conservative values for application's uniprocessor execution rates (100MFLOPS) and grid-point processing time ($20\mu s$).

- We note that the application is extremely sensitive to communication costs. This means that simple improvements like overlapping computation and communication will bring up to 100% improvements in efficiency. This also means that we should consider network performance issues when taking load-balancing decisions.

To conclude, we estimate that Cactus could run at an execution rate of 40 TFLOPS on a megacomputer. This is more than 30 times faster than the best execution rate achievable now [12] on a supercomputer. Based on Moore's Law (validated also by the evolution of the fastest supercomputer worldwide as reported by Gordon Bell awards), it will take 7 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 sooner, to offer a more powerful, low cost alternative to supercomputers.

5 Summary

We provided a detailed model of a scientific application, a typical model of finite differences code, 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 determined that the limiting factor for efficiency is network bandwidth, which is going to improve dramatically over the next few years.

We predicted Cactus performance in 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.

References

[1] A. Alexandrov et al. LogGP: incorporating long messages into the LoGP model. In *SPAA'95*, 1995.

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

[3] Kurt Anstreicher, Nathan Brixius, Jean-Pierre Goux, and Jeff Linderoth. Solving large quadratic assignment problems on computational grids. *http://www-unix.mcs.anl.gov/metaneos/publications*, 2000.

[4] Werner Benger, Ian Foster, Jason Novotny, Edward Seidel, John Shalf, Warren Smith, and Paul Walker. Numerical Relativity in a Distributed Environment. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.

[5] Charlie Catlett and Ian Foster. The network meets the computer: Architectural implications of unlimited bandwidth. 2001. to appear.

[6] IBM Corp. IMB SP POWER3 SMP node system architecture. Technical report, IBM. http://http://www.rs6000.ibm.com/resource/technology/spwppwr3.html.

[7] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, 28(7):1–12, July 1993.

[8] Ian Foster. *Designing and Building Parallel Programs.* Addison-Wesley, 1995. Also at: `http://www.mcs.anl.gov/dbpp/`.

[9] Ian Foster. Internet Computing and the emerging Grid. *Nature*, vol 408 issue 6815, 2000.

[10] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[11] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco, CA, 1999.

[12] Junichiro Makino, Toshiyuki Fukushige, and Masaki Koga. A 1.349 tflops simulation of black holes in a galactic center in a GRAPE-6. In *Supercomputing 2000*, 2000.

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

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

[15] Lili Qiu, Yin Zhang, and Srinivasan Keshav. On individual and aggregate TCP performance. *Proceedings of 7th International Conference on Network Protocols, Toronto, Canada,*, 1999.

[16] Ion Stoica, Florin Sultan, and David Keyes. A hyperbolic model for communications in layered parallel processing environments. *Journal of Parallel and Distributed Computing*, 39(1):29–45, 25 November 1996.