Cactus Application: Performance Predictions in Grid Environments

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

[1] Departmet of Computer Science, The University of Chicago
1100 E. 58th Street, Chicago, IL 60637, USA
{matei, anda, foster}@cs.uchicago.edu
[2] Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, IL 60439, USA

Abstract. The Cactus software is representative for a whole class of scientific applications; typically those that are tightly coupled, have regular space decomposition, and huge memory and processor time requirements. Cactus proved to be a valuable tool for astrophysicists, who first initiated its development. However, today's fastest supercomputers are not powerful enough to perform realistically large astrophysics simulations with Cactus. The emergence of innovative resource environments like 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 [10], a new field that is distinguished from conventional distributed computing by its focus on 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, 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.

Experiments on large pools of Internet-connected resources have been successful. For example, [4] 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 seem, at first sight, to be capable of exploiting it fully. 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 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.

Cactus [1] was originally developed as a framework for finding numerical solutions to Einstein's equations 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. Its name 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, or standard computational tools, such as parallel I/O, data distribution, or checkpointing. We analyze an application that simulates the collision of two black holes.

A basic module (thorn) implements unigrid domain decomposition. It decomposes the global domain over processors and places an overlap region (referred to as *ghost-zone*) on each processor. This reduces the number of messages (and hence 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.

To better understand our test application, we study its sequential behavior and build and validate the performance model for two different supercomputer architectures (Section 2). This performance model is later adapted to a pool of supercomputers (Section 3.1). We predict the application efficiency in this environment and study the factors that limit performance. In Section 3.2 we move our application performance discussion to Internet computing.

2 Application Execution on Traditional Architectures

We analyzed the sequential and parallel execution of our application on two supercomputers: a shared memory machine (Silicon Graphics Origin 2000) and a message passing based supercomputer (IBM SP). We built a performance model for the parallel execution, validated it against real data, and used 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.

Details on the analysis of the sequential execution are presented in [12]. Relevant for our problem are memory usage and execution time per grid point. We learned that, for avoiding memory penalties, the problem space allocated on a processor with a RAM MB of associated memory should be $10^6 \times \frac{RAM-16}{512}$ grid points. We determined the time to process one grid-point is $t_c = 17\mu s$ on a RISC 10000 processor and $t_c = 24\mu s$ on a Power3 processor. In addition, we learned how to avoid cache conflicts (also presented in [12]) that strongly degrade performance.

Communication among processors is what differentiates the parallel 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 twice, 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 per 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.

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. 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. We consider $g_x = g_y = g_z = g$.

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. Moreover, we observe that the speed of the whole computation is the speed of the slowest processor, and model only interior processors, e.g., those processors that have 6 neighbors.

We consider a simple latency/bandwidth communication cost model: the cost of sending a message of length L between any two processors is:

$$t_{msg} = t_s + L \times t_w \tag{1}$$

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 $\frac{6I}{g}$ messages. For a ghost-zone size of g, δ bytes sent per grid point, and assuming the link connecting any two neighboring processors is not shared, the total time spent communicating is: $T_{comm} = \frac{6I}{g} t_s + 4It_w(yz + xz + xy)\delta$

Execution Time Each processor spends its time on *useful* work, communicating, and on *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 $\sum_{j=1}^{g-1} j(xy +$

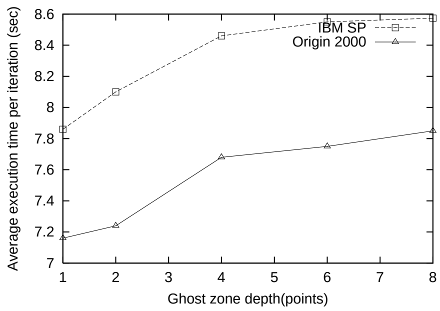

Fig. 1. Average time per iteration as a function of ghost-zone size.

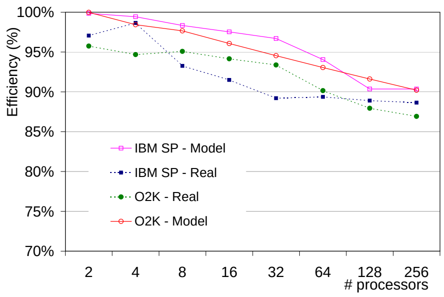

Fig. 2. Efficiency measured when problem size per processor is constant.

$yz+xz) = \frac{g(g-1)}{2}(xy+yz+xz)$ 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: $T_r = It_c(g-1)(xy+yz+xz)$.

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

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

Optimal Ghost-Zone Size We determine the optimal ghost-zone size for which the overhead introduced by parallelism is minimum. $T_{overhead}$ (2) 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 the ghost-zone size.

We have validated this theoretical conclusion by measuring the execution time on 8 processors on Origin2000 and IBM SP with different ghost-zone sizes. The experimental data (Figure 1) confirms our result: 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.

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:

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

Equation 3, 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 use a competition for bandwidth model [7] adapted to the interconnection characteristics of the two supercomputers: we identify shared hardware components, compute the number of competing flows, and use manufacturer's performance specifications. For these experiments we use a memory constrained model: the problem size per processor remains constant while the number of processors increases up to 256. Figure 2 compares experimental results with our predictions. Although our communication models are simplistic, the test results match the predictions within a 10% range. Other models, like hyperbolic [16], could lead to more accurate predictions.

3 Predicted Performance in Grid Environments

We consider two different computational Grids. The first is a collection of supercomputers connected by a Grid middleware infrastructure like the Globus toolkit [9]. 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.

3.1 Performance on a Pool of Supercomputers

We predict the application efficiency on a collection of Internet-connected supercomputers 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. 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.
- The problem space is decomposed using a 1D decomposition among supercomputers and a 3D decomposition among the processors of each supercomputer. 1D decomposition among supercomputers is realistic since a relatively small number of supercomputers (maybe hundreds) is likely to be simultaneously available to a group in the near future. However, it is straightforward to extend the model to a 2D or 3D decomposition.

– Supercomputers are connected through identical network links to the Internet. We use the same linear model (1) for communication costs. In addition, 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 each supercomputer a computational entity that obeys the performance model described in Section 2. Hence, we model a supercomputer as a 'faster' processor with a 'big' associated memory. This *super*processor will be characterized by the time θ_c needed to update one grid point (the equivalent of t_c presented in Section 2).

Execution Time. Using the model described in Section 2, the time spent for useful work on a problem of size $X \times Y \times Z$ on a supercomputer with P processors is: $\Theta_{seq} = \theta_c XYZ$. 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: $\Theta_{seq} = \frac{t_c}{E} \times \frac{X}{\sqrt[3]{P}} \frac{Y}{\sqrt[3]{P}} \frac{Z}{\sqrt[3]{P}}$ We can now compute θ_c the time it takes to process a grid-point: $\theta_c = \frac{t_c}{EP}$.

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 $\Theta_r = 2I\theta_c(G-g)XY$.

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:

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

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 communica-

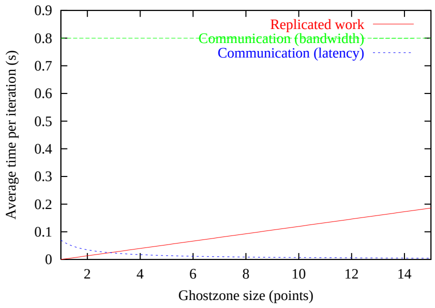

Fig. 3. Components of parallel overhead time. Gigabit links are assumed.

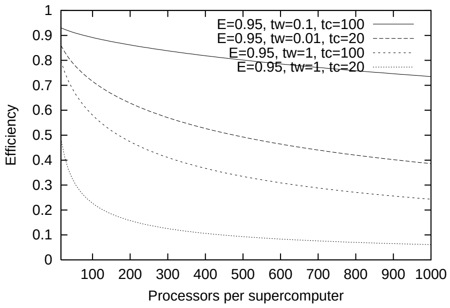

Fig. 4. Achievable efficiency on a pool of supercomputers with in-place networks.

tion link. Therefore, the time spent communicating is:

$$\Theta_{comm} = \frac{I}{G} \times 2(\theta_s + 2\theta_w GXY\delta) = \frac{2I\theta_s}{G} + 4I\theta_w XY\delta \tag{5}$$

Optimal Ghost-Zone Size. From (4) maximum efficiency E' is obtained for

$$G_{opt} = \frac{1}{x\sqrt[3]{P}}\sqrt{\frac{\theta_s}{\theta_c}} \tag{6}$$

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 60^3 grid-points per processor, the maximum overall efficiency is obtained for $G_{opt} = 2$.

These results suggest that using different ghost-zones sizes for supercomputer communication does increase overall efficiency. However, Figure 3 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 need one order of magnitude faster links to obtain significant savings due to deeper ghost-zones.

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:

$$E_{overall} = E\varepsilon = E\frac{\Theta_{seq}}{S \times \Theta_{par}} = \frac{1}{\frac{1}{E} + \frac{2\theta_s}{xyzt_c} + \frac{4\theta_w\delta\sqrt[3]{P^2}}{zt_c}} \tag{7}$$

We validated (7) with two experiments executed on supercomputers across the US. The experimental setup, including details on the middleware infrastructure used (Globus and MPICH-G2), is presented in [3]. The first experiment used 2 supercomputers with up to 64 processors each. The second, large-scale experiment involved 4 supercomputers of a total of 1500 processors (described in [2]). In both cases the results measured were at most 15% less than our predictions. We believe this difference is mostly due to the highly variable behavior of the wide area network.

In Figure 4 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 equation (7) and from Figure 4, 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.

3.2 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 [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 (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 caused backbone network capacity to grow of an even faster rate [15]. 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 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 system (*megacomputer*) closer to reality. Megacomputers might be the world's first Peta-op (10^{15} FLOPS) computing systems. However, for applications like Cactus, increased computational power may be less significant than the aggregated memory of millions of computers which will allow solving problems of unprecedented scale and resolution.

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 a cluster to have 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 ([14]) some mending is needed at the 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 analyzed the performance of a megacomputer using the same model as in previous sections. Instead of describing the whole process in detail, we just summarize our conclusions. Efficiency of 15-20% can be obtained 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.

To conclude, we estimate that Cactus could run at an execution rate of 20 TFLOPS on a megacomputer. This about 30 times faster than the best execution rate achievable now [13] on a supercomputer. Based on Moore's Law (which, judging by the power of the fastest supercomputer worldwide as per the annual Gordon Bell awards, still holds), it will take 6 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.

4 Summary

We provided a detailed performance model of a scientific application—a typical stencil computation code—and validated it on two architectures. 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.

Our prediction model shows that with better network connectivity than in place today and using computational Grids, scientists will shortly have a powerful computational platform at a 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*. Submitted to 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*, 2001, to appear.
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. IBM, *IBM SP POWER3 SMP node system architecture*, Technical report, IBM.
12. M. Ripeanu, A. Iamnitchi, *Performance Predictions for a Numerical Relativity Package in Grid Environments*, Technical Report TR-2001-23, U. of Chicago, 2001.
13. J. Makino, T. Fukushige, M. Koga, *A 1.349 TFLOPS simulation of black holes in a galactic center in GRAPE-6*, In Supercomputing 2000.
14. R. Morris, *TCP behavior with many flows*, IEEE International Conference on Network Protocols, Atlanta, Georgia, October, 1997.
15. A. Odlyzko, *Internet growth: Myth and reality, use and abuse*, Information Impacts Magazine, November, 2000.

16. 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.