

An Application-driven Study of Parallel System Overheads and Network Bandwidth Requirements

Anand Sivasubramaniam, Member, IEEE,
Aman Singla,
Umakishore Ramachandran,
H. Venkateswaran

Anand Sivasubramaniam is with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA. Email: anand@cse.psu.edu

Aman Singla is with Silicon Graphics Inc., Mountain View, CA. Email: aman@engr.sgi.com

Umakishore Ramachandran is with the College of Computing, Georgia Institute of Technology, Atlanta, GA. Email: rama@cc.gatech.edu

H. Venkateswaran is with the College of Computing, Georgia Institute of Technology, Atlanta, GA. Email:venkat@cc.gatech.edu

Acknowledgement: This work has been funded in part by NSF grants MIPS-9058430, MIPS-9200005, MIPS-9630145, and an equipment grant from DEC. Anand Sivasubramaniam has been supported in part by a NSF Career Award MIPS-9701475.

Contact Information: Anand Sivasubramaniam, Department of Computer Science and Engineering,
220 Pond Lab, The Pennsylvania State University, University Park, PA 16802. Tel: (814) 865-1406
Fax: (814) 865-3176 Email: anand@cse.psu.edu

Abstract

Evaluating and analyzing the performance of a parallel application on an architecture to explain the disparity between projected and delivered performance is an important aspect of parallel systems research. However, conducting such a study is hard due to the vast design space of these systems. In this paper, we study two important aspects related to the performance of parallel applications on shared memory parallel architectures. First, we quantify the *overheads* observed during the execution of these applications on three different simulated architectures. We next use these results to synthesize the bandwidth requirements for the applications with respect to different network topologies. This study is performed using an execution-driven simulation tool called SPASM, which provides a way of isolating and quantifying the different parallel system overheads in a non-intrusive manner. The first exercise shows that in shared memory machines with private caches, as long as the applications are well-structured to exploit locality, the key determinant that impacts performance is network contention. The second exercise quantifies the network bandwidth needed to minimize the effect of network contention. Specifically, it is shown that for the applications considered, as long as the problem sizes are increased commensurate with the system size, current network technologies supporting 200-300 MBytes/sec link bandwidth are sufficient to keep the network overheads (such as latency and contention) within acceptable bounds.

Index Terms: parallel systems overheads, execution-driven simulation, interconnection network, application-driven study.

LIST OF FIGURES

1	Overheads in a Parallel System	3
2	Validation Experiments	10
3	Speedup	12
4	Performance of IS	14
5	Performance of FFT	16
6	Performance of CG	18
7	Link Bandwidth vs. Efficiency	19

LIST OF TABLES

I	IS : Overhead Functions	15
II	FFT : Overhead Functions	15
III	CG : Overhead Functions	17
IV	IS: Impact of Processors on Link Bandwidth (in MBytes/sec)	22
V	IS: Impact of CPU speed on Link Bandwidth (in MBytes/sec)	22
VI	IS: Impact of Problem Size on Link Bandwidth (in MBytes/sec)	23
VII	IS: Link Bandwidth Projections (in MBytes/sec)	23
VIII	FFT: Impact of Processors on Link Bandwidth (in MBytes/sec)	24
IX	FFT: Impact of CPU speed on Link Bandwidth (in MBytes/sec)	24
X	FFT: Impact of Problem Size on Link Bandwidth (in MBytes/sec)	25
XI	FFT: Link Bandwidth Projections (in MBytes/sec)	25
XII	CG: Impact of Processors on Link Bandwidth (in MBytes/sec)	26
XIII	CG: Impact of CPU speed on Link Bandwidth (in MBytes/sec)	26
XIV	CG: Impact of Problem Size on Link Bandwidth (in MBytes/sec)	27
XV	CG: Link Bandwidth Projections (in MBytes/sec)	27

I. INTRODUCTION

Parallel computing is being used increasingly in a variety of domains including scientific, technical, and interactive applications. It is important to structure applications such that they exploit the full computational capacity of parallel machines. By the same token, the architecture of parallel machines should address the computational and communication demands of the applications. This suggests that the design of applications and architectures are complementary processes. For this process, we need enabling tools and studies to analyze the *parallel system*¹ *overheads* that come into play when an application is implemented on a parallel architecture. The synchronization and communication that are inherent in implementing an application on a parallel architecture manifest themselves as different types of overheads during the execution of the application. First, there may be algorithmic bottlenecks causing overheads that would be incurred regardless of the underlying architecture. Further, there may be architectural overheads due to the memory hierarchy, synchronization primitives, and the interconnection network. It is important to understand how and to what extent each of these overheads contribute towards the overall execution time of the application. Higher algorithmic overheads would give a cue to the programmer to design alternate algorithms/implementations. From the architectural side, finding out the overheads due to each hardware component would help the system architect derive alternate architectural mechanisms in a cost-effective manner.

In this paper, we undertake a study that quantifies the algorithmic and network overheads for a set of applications on three simulated shared memory parallel architectures. We use the results of this study to synthesize the bandwidth requirements for these applications with respect to different network topologies.

To perform the study, we use an execution-driven simulator called SPASM (Simulator for Parallel Architectural Scalability Measurements) that allows performance evaluation of an application on a range of hardware platforms. SPASM provides detailed statistics about the application execution to help quantify the different parallel system overheads. SPASM can be used to study any aspect of a parallel system such as the memory hierarchy, synchronization primitives, and interconnection network. In this study, we focus on topology and bandwidth, two important attributes of the interconnection network, to quantify the network overheads. These two attributes contribute towards the latency and

¹We use the term, *parallel system*, to denote an application-architecture combination.

contention that is experienced by the applications. This paper makes the following three contributions:

- We first present the design and implementation of an execution-driven simulator that allows studying any aspect of a parallel system. In particular, we show its use in quantifying the algorithmic and network related overheads for a set of applications on three shared memory parallel architectures. Further, we use the measured overheads to discuss the growth of these overheads for larger problem sizes and larger configurations of these architectures.
- It is shown that in shared memory machines with private caches, as long as the applications are well-structured to exploit locality, the key determinant that impacts performance is network contention.
- We synthesize the bandwidth requirements for the applications on the binary hypercube network topology, and use them to project similar requirements for other topologies. In particular, it is shown that for the applications considered, as long as the problem sizes are increased commensurate with the system size, current network technologies supporting 200-300 MBytes/sec link bandwidth are sufficient to keep the network overheads (such as latency and contention) within acceptable bounds.

The rest of this paper is organized as follows. We first discuss metrics that would be desirable for a detailed understanding of the performance of parallel systems in Section II, and review related work in Section III. We next describe the SPASM simulator and its capability in providing these metrics in Section IV. The details of the experimental platform (simulated hardware and applications) are also given in this section. Section V quantifies the overheads for the set of applications using SPASM. Section VI synthesizes the network bandwidth requirements using SPASM. Finally, Section VII summarizes the contributions of this paper.

II. PERFORMANCE METRICS

Metrics which capture processor characteristics in terms of the clock speed (MHz), the instruction execution speed (MIPS), the floating point performance (MFLOPS), and the execution time for standard benchmarks have been widely used in modeling uniprocessor performance. However, in a parallel system, the hardware specification (processor, memory hierarchy, and interconnection network) may never be a true indicator of the performance delivered by the system. This is due to overheads arising

from complex interactions between application characteristics and architectural features. These overheads often result in a significant disparity between *available*² and *delivered*³ performance in a parallel system. Metrics for parallel system performance evaluation should quantify this gap between available and delivered compute power since understanding the application and architectural bottlenecks is crucial for application restructuring and architectural enhancements. Many performance metrics [1], [2], [3], [4], [5], [6] have been proposed to quantify the match between application and architecture in a parallel system. While these metrics are useful for tracking overall performance trends, they provide little additional information about where performance is lost. Some of these metrics [4], [5], [6] attempt to identify the cause (the application or the architecture) of the problem when the parallel system does not perform as expected. Once the problem is identified, it is essential to find the individual application and architectural artifacts that lead to these bottlenecks and quantify their relative contribution towards limiting the overall performance of the system. Traditional metrics [4], [5], [6] do not help further in this regard.

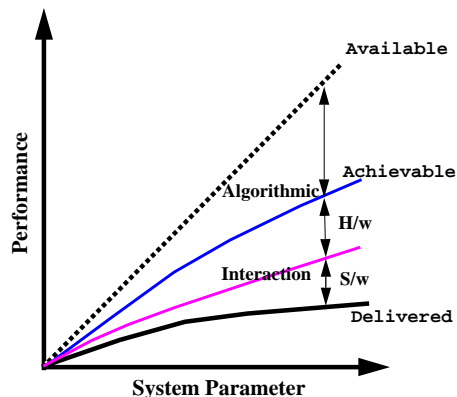


Fig. 1. Overheads in a Parallel System

Let us assume that we are tracking performance as a function of some system parameter as shown in Figure 1. For instance, the number of processors can be the system parameter (x-axis) and speedup can be the performance metric (y-axis). If the application execution is limited by only the available processing power, then we would get a speedup that increases linearly with an increase in the number of processors (curve labeled “available” in Figure 1). However, there are usually many overheads in a parallel system that affect its performance resulting in a speedup curve (labeled “delivered” in Figure

²We define *available* performance as the product of the number of nodes and the peak processing rate per node in a parallel machine.

³We define *delivered* performance as the actual performance obtained for an application on a parallel machine.

1) that grows much slower. The disparity between the “available” and “delivered” curves is due to growth of overheads in the system. Parallel system overheads may be broadly classified into a purely algorithmic component (*algorithmic overhead*), a component arising from the interaction of the application with the system software (*software interaction overhead*), and a component arising from the interaction of the application with the hardware (*hardware interaction overhead*). Algorithmic overheads arise from the inherent serial part in the application, the work-imbalance between the executing threads of control, any redundant computation that may be performed, and additional work introduced by the parallelization. The achievable performance of an application even on an ideal machine such as the PRAM [7] would be lower than the “available” curve as depicted by the “achievable” curve in Figure 1 due to algorithmic overheads. The implementation of the application on an actual machine would result in further slowdown due to software and hardware interaction overheads. Software interaction overheads such as overheads for scheduling, message-passing, and software synchronization arise due to the interaction of the application with the system software. Hardware slowdown due to network latency (the transmission time for a message in the network), network contention (the amount of time spent in the network waiting for availability of network resources), synchronization and cache coherence actions, contribute to the hardware interaction overhead. To fully understand the performance of a parallel system, it is important to isolate and quantify the impact of each of these components on the overall execution as shown in Figure 1. We have proposed a concept of *overhead functions* [8], [9] to capture the growth of particular system overheads with respect to specific system parameters. Crovella and LeBlanc [10] propose a similar set of metrics called Lost Cycles. Both these metrics quantify the contribution of each overhead towards the overall execution time. The studies differ in the techniques used to quantify these metrics. Crovella and LeBlanc [10] use experimentation, while simulation is used in our approach.

III. RELATED WORK

There have been a number of studies addressing architectural issues such as network latency and contention [11], [12], [13], [14], and synchronization [15], [16] in isolation. While such issues are extremely important, their performance impact should be put in perspective by considering them in the context of the overall application. Recognizing this importance, the current trend in architectural

evaluation is to use real applications for analyzing the performance of parallel systems (see for e.g. [17], [18], [19], [10]).

Performance measurement by experimentation, analytical modeling and simulation are three well-known techniques that can be used for quantifying parallel system overheads. Experimentation involves implementing the application on the actual hardware and measuring its performance. Analytical models abstract hardware and application details in a parallel system and capture complex system features by simple mathematical formulae. These formulae are usually parameterized by a limited number of degrees of freedom so that the analysis is kept tractable. Simulation is a valuable technique which exploits computer resources to model and imitate the behavior of a real system in a controlled manner. Each technique has its own limitations. The amount of statistics that can be gleaned by experimentation is limited by the monitoring/instrumentation support provided by the underlying system. Additional instrumentation can sometimes perturb the evaluation. Further, experimentation may not permit the variation of certain system parameters, which is a necessity for the study outlined here. Analytical models are often criticized for the unrealism and simplifying assumptions made in expressing the complex interaction between the application and the architecture. Simulation of realistic computer systems demand considerable resources in terms of space and time. A related survey [20] compares the pros and cons of these techniques in terms of the amount of *statistics* that can be collected from the evaluation, the *accuracy* of the results, and the *cost* in terms of computer and human resources expended in the evaluation. It has been shown [20], [21] that execution-driven simulation of parallel applications is invaluable for obtaining detailed and accurate statistics required to quantify parallel system overheads, and is the technique of choice in this paper.

Simulation tools have been successfully used to study specific parallel systems [22], [23], [24]. For instance, MemSpy [22] is a performance debugging tool that is used in conjunction with the Tango simulation platform to locate and fix memory system bottlenecks in applications. ParaView [23] presents the times spent in computation, synchronization and memory hierarchy by simulation. The AIMS toolkit that comes with the Axe [24] simulation platform supports automatic instrumentation, run-time monitoring and graphical analysis of performance for message-passing parallel programs. Performance debugging and visualization tools are not uncommon [25], [26], [27] and have been shown to be invaluable for analyzing and consequently improving specific architectural features or enhancing

application performance. SPASM [9], [8] (to be discussed in the next section) is perhaps the first general purpose execution-driven simulator that has been used to study a wide range of applications (shared memory and message passing) and architectures (including the details of the interconnection network, and memory hierarchy) for the purpose of isolating and quantifying overheads.

IV. SPASM

SPASM is an execution-driven simulator written in CSIM [28] used for simulating the execution of a parallel program on a parallel machine. As with other recent simulators [29], [30], [31], [32], [33], the bulk of the instructions in the parallel program is executed at the speed of the native processor (SPARC in our studies) and only instructions such as LOADs/STOREs on a shared memory platform, and SENDs/RECEIVEs on a message passing platform, that may potentially involve a network access are simulated. On a message passing system, the calls (SENDs/RECEIVEs) which trap to the simulator are inserted into the application program explicitly by the programmer. On a shared memory system, a pre-processor inserts code into the application program to trap to the simulator on a shared memory reference. On both systems, the compiled assembly code is augmented with cycle counting instructions which is used to keep track of the time spent in the application program since the last trap to the simulator. At the trapped instruction, SPASM reconciles the simulated time for the processor issuing the instruction since the last trap using the cycle counts. This technique has been popular in other execution-driven simulators [29], [30], [31], [33], [34] as well. Finally, the assembled binary is linked with the rest of the simulator code.

A simulation platform like SPASM allows us to vary a wide range of hardware parameters such as the number of processors, CPU clock speed, network topology, bandwidth of the links in the network, network switching delays, and cache parameters (the block size, cache size, associativity, etc). We have used SPASM to study both message passing [8] and shared memory platforms [9], [35], [36], [37]. Within the scope of shared memory platforms, we have used SPASM to study different network topologies [9], [35], memory consistency models and cache coherence protocols [37]. We confine ourselves to the shared memory platform in this paper and we present the details of SPASM relevant to this platform.

A. Statistics

Each simulated processor in SPASM passes through a set of system-defined and user-defined modes while executing a shared memory parallel program. Detailed statistics are collected by SPASM for each processor individually for each of these modes and the results are presented for a representative processor specified by the user. These statistics are used to compute the overhead functions. The system-defined modes provided by SPASM are:

- **BARRIER:** Mode corresponding to a barrier synchronization operation. SPASM provides a tournament barrier [16] as a library routine to support this operation.
- **MUTEX:** Mode corresponding to a mutual exclusion lock. The library routine provided by SPASM implements this using a test-test&set [15] operation.
- **PGM_SYNC:** Parallel programs may use Signal-Wait semantics for pairwise synchronization. A lock is unnecessary for the Signal variable since only 1 processor writes into it and the other reads from it. This mode is used to differentiate such accesses from normal load/store accesses.
- **NORMAL:** Mode corresponding to the actual work done in the application. A program is in the NORMAL mode if it is not in any of the other modes. A user may choose to specify user-defined modes within this mode to get detailed statistics for specific regions of the application. Such a feature can narrow down bottlenecks in the execution and help in application restructuring.

For each executing processor, SPASM accumulates the following statistics for each mode of execution:

- *computation time:* This is the time taken to execute on an ideal machine such as the PRAM [7].
- *latency:* Accesses to variables in a shared memory system may involve the network, and the physical limitations of the network tend to contribute to overheads in the execution. The latency overhead is defined as the total amount of time spent by a processor waiting for messages due to the transmission time on the links and the switching overhead in the network assuming that the messages did not have to contend for any link.
- *contention:* The contention overhead is the total amount of time incurred by a processor due to the time spent waiting for availability of network resources by its messages.

It is worthwhile to mention that SPASM provides statistics for both the latency and contention incurred by a message as well as the latency and contention that a processor may choose to see. Even though a message may incur a certain latency and contention, a processor may choose to hide all or part

of it by overlapping computation with communication. Such a scenario may arise with a non-blocking message operation where the processor is not stalled during message transfer. SPASM also provides statistical information about the network. It gives the utilization of each link in the network and the average queue lengths of messages at any particular link. This information can be useful for identifying network bottlenecks and comparing relative merits of different networks and their capabilities.

Since the focus of this study is primarily on hardware interaction overheads, particularly the communication overhead, we do not address software interaction overheads such as scheduling⁴ in this paper. However, the modularity in the implementation of SPASM makes it easy to investigate these other overheads as is illustrated in a related study [8] where we use it to model software message passing overheads.

The *total time* (simulated time) for a given application is the maximum of the *execution times* of the individual parallel processors. This is the time that would be taken by an execution of the parallel program on the target machine. The execution time for a processor is the sum of its execution times over all the modes. And the execution time for a processor in any particular mode is the sum of the computation time, latency and contention overheads. *Speedup* using p processors is measured as the ratio of the total time on 1 processor to the total time on p processors (“delivered” curve in Figure 1).

Ideal time is the total time taken by a parallel program to execute on an ideal machine such as the PRAM. It includes the algorithmic overhead but does not include the interaction overhead. SPASM simulates an ideal machine to provide this metric and the resulting speedup gives the “achievable” curve shown in Figure 1. As we mentioned earlier, the difference between the ideal time and the execution time corresponding to the “available” speedup curve gives the algorithmic overhead, and the difference between the total time and the ideal time gives the interaction overhead (see Figure 1).

The computation time, latency and contention overheads capture interesting facets in each mode of execution. Computation time in the NORMAL mode is the time spent in local computation which is a measure of the actual work in the application. The latency and contention overheads in the NORMAL mode reflect the network overheads for ordinary data accesses which is determined by the data access pattern of the application. For the BARRIER and PGM_SYNC modes, the computation time is the time incurred by a processor waiting for one or more processors to arrive at the synchronization point

⁴We do not distinguish between the terms, *process*, *processor*, and *thread*, and use them synonymously.

as a consequence of the algorithmic work imbalance. The computation time in the MUTEX mode is the time spent waiting to acquire a lock and represents the serial part in an application arising due to critical sections. For the BARRIER and MUTEX modes, the computation time also includes the cost of implementing the synchronization primitive and other residual effects due to latency and contention for prior accesses. In all three synchronization modes, the latency and contention overheads together represent the network overheads incurred in accessing synchronization variables. The metrics identified by SPASM thus quantify the interesting components of the algorithmic and interaction overheads.

B. Application and Hardware Characteristics

We have used a set of applications with diverse characteristics for this study. Three of the applications (EP, IS and CG) are from the NAS parallel benchmark suite [38]; CHOLESKY is from the SPLASH benchmark suite [39]; and FFT is the well-known Fast Fourier Transform algorithm. EP and FFT are well-structured applications with regular communication patterns determinable at compile-time, with the difference that EP has a higher computation to communication ratio. IS (a bucket sort of integers) has a regular communication pattern also, but it uses locks, in addition, for mutual exclusion during certain phases of the execution. CG and CHOLESKY are different from the other applications in that their communication patterns are not regular (both use sparse matrices) and cannot be determined at compile time. While a certain number of rows of the matrix in CG is assigned to a processor at compile time (static scheduling), CHOLESKY uses a dynamically maintained queue of runnable tasks. Further details on these applications are given in [40].

We have used three Cache Coherent Non-Uniform Memory Access (CC-NUMA) shared memory platforms. Each node in the simulated CC-NUMA hierarchy is assumed to have a sufficiently large piece of the globally shared memory such that for the applications considered, the data-set assigned to each processor fits entirely in its portion of shared memory. The private cache modeled at each processing node is a 2-way set-associative cache (64 KBytes with 32 byte blocks) that is maintained sequentially consistent using an invalidation-based (Berkeley protocol) full-mapped directory-based cache coherence scheme. We use a SPARC chip as the baseline for fixing the processor characteristics. For the interconnection network, we choose three different network topologies (a *fully connected network*, a *binary hypercube* and a *2-D mesh*) from the connectivity spectrum. All three networks use serial (1-bit

wide) unidirectional links but the bandwidth of these links can be varied. The fully connected network models two links (one in each direction) between every pair of processors in the system. The cube platform connects the processors in a binary hypercube topology and uses the e-cube algorithm [41] for routing messages. The 2-D mesh resembles the Intel Paragon system. Links in the North, South, East and West directions, enable a processor in the middle of the mesh to communicate with its four immediate neighbors. Processors at corners and along an edge have only two and three neighbors respectively. Equal number of rows and columns is assumed when the number of processors is an even power of 2. Otherwise, the number of columns is twice the number of rows (we restrict the number of processors to a power of 2). A wormhole routing strategy is employed and the message-size can vary upto 32 bytes (with an 8 byte header). The switching delay is assumed to be negligible compared to the transmission time and is ignored.

The parallel system parameters in this study (both hardware and application problem sizes) have been chosen such that they strike the right balance between computation and communication. With technological advances, so long as these parameters are scaled up retaining this balance, the results from this study (in terms of relative apportionment of overheads) will continue to apply.

C. Validation

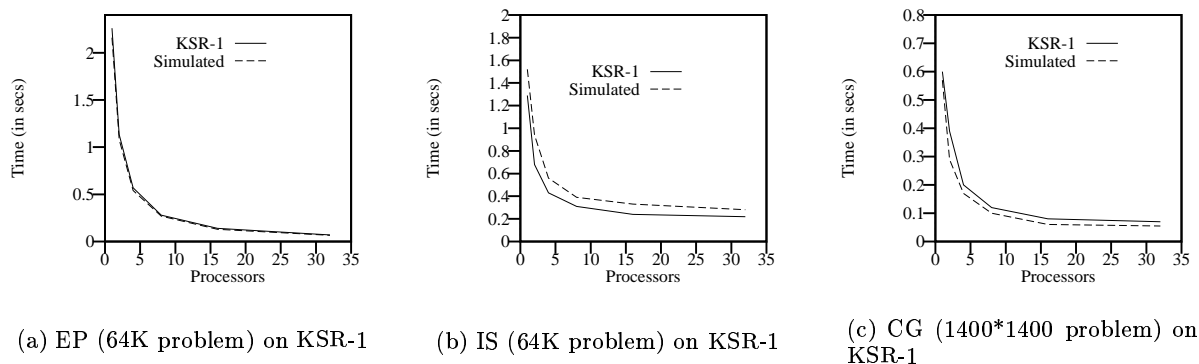


Fig. 2. Validation Experiments

We first validate the SPASM simulator by comparing the results that we get from SPASM to those obtained by implementing the applications on a KSR-1 [42]. Such a comparison is done at a “macro” level since KSR-1 does not offer assistance for measuring all the individual parallel system overheads that are obtainable from SPASM⁵. The KSR-1 results are drawn from our previously published studies

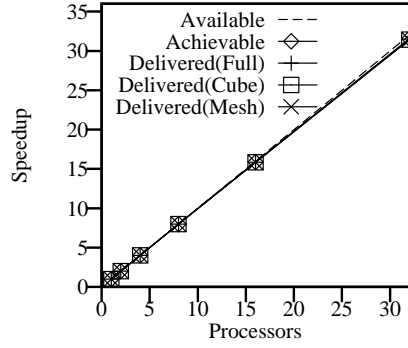
⁵KSR-1, as well as other machines such as the IBM SP-2, offer hardware and software performance monitoring tools that can help measure selected hardware and software overheads [10], [43], [44].

reported in [45], and are compared against a comparable model on SPASM. Figure 2 shows that our simulation model closely captures the scalability trends of EP (an application with negligible overheads), IS (an application with fairly high algorithmic and interaction overheads for a small problem size of 64K chosen in these experiments), and CG (an application with overheads lying between those for EP and IS). These figures reflect the general behavior observed over the chosen applications.

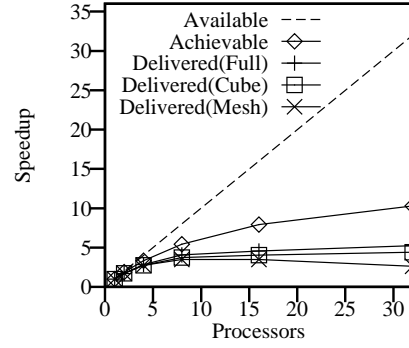
V. QUANTIFYING PARALLEL SYSTEM OVERHEADS

Having described the SPASM tool, we now illustrate its role in quantifying parallel system overheads for the application programmer and system architect. We quantify the parallel system overheads (as a function of the number of processors p) for the five applications (EP, IS, FFT, CG and CHOLESKY) on the CC-NUMA hardware platform outlined in section IV-B with the fully connected, mesh and hypercube network topologies. We fix the clocks of the processors in our simulated machine at 33 MHz and the bandwidth of the links in the network at 20 MBytes/sec. For the applications, we use a problem size of 64K for EP, IS and FFT, a sparse matrix of size 1400X1400 containing 100,300 non-zeroes for CG, and a 1806-by-1806 matrix with 30,824 non-zeros for CHOLESKY. We also briefly discuss the impact of larger problem sizes.

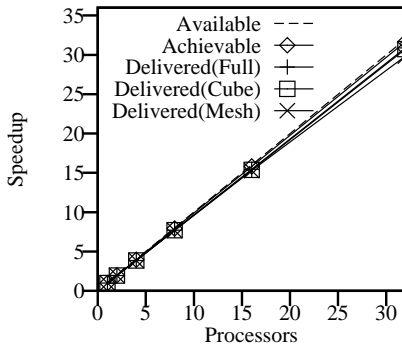
Figures 3 (a), (b), (c), (d) and (e) show the “delivered” speedup curves (see section II) for EP, IS, FFT, CG and CHOLESKY respectively on the three hardware platforms. If we were to simply run these applications on the actual hardware without any performance debugging tool, this is all the information that would be available to us. An application programmer or a system architect would be interested in finding out what causes the disparity between the “available” and “delivered” speedup (for IS and CHOLESKY at least where the disparity is significant). The detailed statistics provided by SPASM help us in this regard. SPASM shows that even the “achievable” (whose disparity from the “available” curve is purely due to algorithmic overheads) deviates significantly from the “available” curve for IS and CHOLESKY. Further, when we look at the deviation of the “delivered” curve from the “achievable” curve, there is negligible deviation for EP on the three hardware platforms; a marginal deviation for FFT and CG; and a significant deviation for IS and CHOLESKY. SPASM also helps us analyze these individual algorithmic and architectural components as is discussed below for IS, FFT, and CG. Details on the other applications can be found in [40].



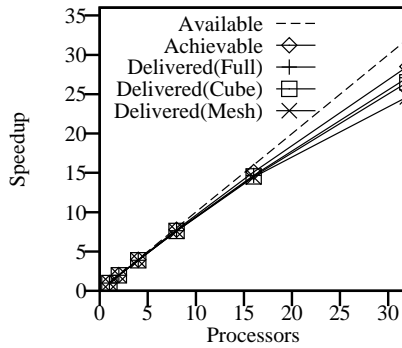
(a) EP



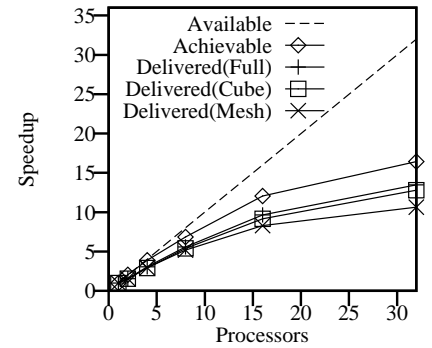
(b) IS



(c) FFT



(d) CG



(e) CHOLESKY

Fig. 3. Speedup

For each application, we show the execution time, the latency, and the contention overhead graphs for the mesh platform since the interaction overhead is the maximum on this topology. In each of the graphs, we present these statistics for the individual modes of execution. We also present the latency overhead and contention overhead curves for the three network topologies. The latency incurred by a single message is relatively independent of the network topology due to the wormhole routing strategy used and the relatively small system configurations. The number of messages in the NORMAL mode (i.e. due to ordinary data access) is determined by the memory reference pattern of the application as well as cache line replacements due to capacity misses. Discounting data dependent executions, the number of messages in the NORMAL mode is also expected to be independent of network topology. Due to the vagaries of the synchronization accesses, it is conceivable that the number of messages could differ across network platforms for the other modes. However, in our experiments we have not seen any significant deviation. As a result, the latency overhead curves for all the applications look

alike across network platforms and we present only one latency curve. On the other hand, it is to be expected that the contention overhead will increase as the connectivity in the network decreases.

Simulation of large parallel systems is not feasible because of the tremendous resources that are needed. As a result, we have restricted our simulations to a maximum of 32 processors. To make projections for larger processor configurations, we have taken the simulation results and performed a non-linear regression analysis using a multivariate secant method with a 95% confidence interval in the SAS [46] statistics package. Further, in many cases, we intentionally omitted one or two data points in fitting the regression models, and then confirmed that the resulting model does indeed capture these left out points with a high degree of confidence. These regression models are individually presented for the computation time, the network latency and contention as a function of the number of processors.

IS

The isolation of parallel system overheads from the detailed statistics provided by SPASM shows that there is a significant algorithmic overhead in the application (the “achievable” curve in Figure 3 (b)). Regardless of the architecture, we cannot hope to get better performance than this achievable curve for the IS application. Parallelization of IS increases the total amount of work to be done across all processors because there is a phase in the execution where the work done by a processor does not decrease when p is decreased [40]. For small problem sizes (64K in this exercise), this inherent algorithmic overhead causes a significant deviation of the achievable curve from the available speedup curve. The capability of SPASM in providing statistics for different segments of program execution by identifying user-defined modes helps in identifying where the algorithmic overhead is incurred. This information can provide useful hints to the programmer suggesting alternate implementations.

There is also a significant interaction overhead component for IS which may be analyzed by considering the different modes of execution. NORMAL and MUTEX are the only significant modes of execution (see Figure 4 (a)) in IS. The latency and contention overheads incurred in the MUTEX mode for accessing lock variables is higher than the corresponding components in the NORMAL mode for accessing ordinary data (see Figures 4 (b) and (c)). As a result, the total execution time in the MUTEX mode surpasses that in the NORMAL mode beyond 20 processors (Figure 4 (a)), which also explains the dip in the speedup curve for mesh (Figure 3 (b)). Since the problem rests within the

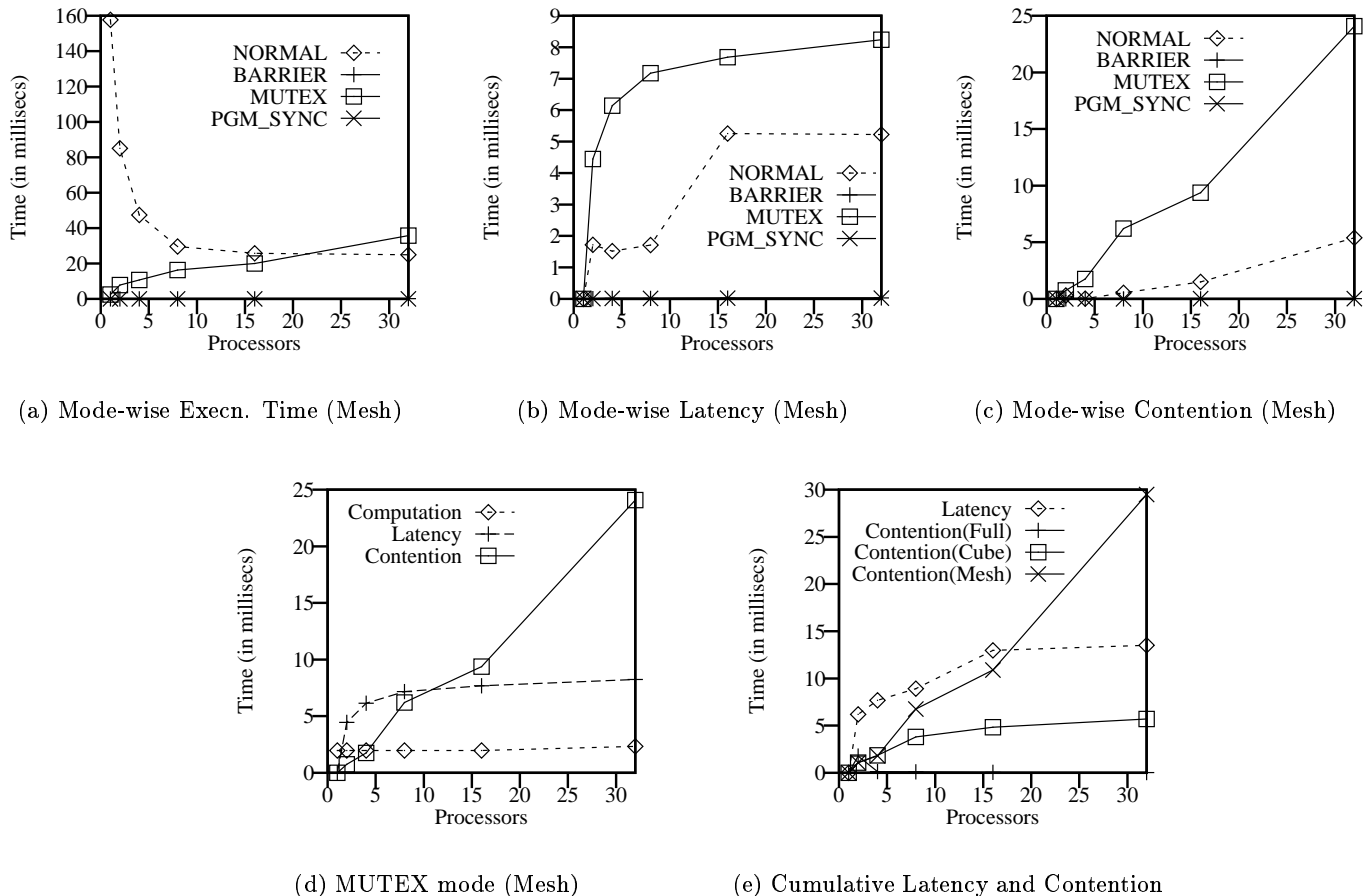


Fig. 4. Performance of IS

MUTEX mode, we can focus in on this mode and analyze the execution (Figure 4 (d)). The computation time in this mode, which reflects the time spent spinning to acquire a lock (implemented as a test-test&set), remains constant with increasing p . To the application programmer, this indicates that the serial portion of the algorithm is not a bottleneck. To the architect, it indicates that the contention for a lock does not increase with p . On the other hand, the contention in the network is the main problem, implying that the architect should improve the network bandwidth to get maximum rewards.

Figure 4 (e) shows the latency overhead and contention overheads for the three network topologies. In IS, since every processor needs to access data from all other processors, and since the data is equally partitioned among the executing processors, the number of accesses to remote locations grows as $(p-1)/p$ [40]. For large p , the latency thus converges to a constant explaining the flattening of the latency overhead curve. On the mesh network, the contention overhead is a severe problem surpassing

the latency overhead at around 18 processors. Table I summarizes the overheads for IS obtained by regression analysis of the datapoints from our simulation.

TABLE I
IS : OVERHEAD FUNCTIONS

IS	Full	Cube	Mesh
Comp. Time (ms)	$129.3/p^{0.7}$	$129.3/p^{0.7}$	$129.3/p^{0.7}$
Latency (ms)	$13.2(1 - 1/p)$	$13.2(1 - 1/p)$	$13.2(1 - 1/p)$
Contention (ms)	<i>Negligible</i>	$4.0 \log p$	$0.9p$

As we mentioned, the algorithmic overhead in parallelizing IS is responsible for a computation time which does not decrease linearly with p (Table I). Table I shows that the contention overhead is negligible and the latency overhead converges to a constant with a sufficiently large p on a fully connected network. Thus for a fully connected network, the performance of IS is expected to closely follow the achievable curve. For the cube and mesh platforms, the contention overhead grows logarithmically and linearly with p , respectively. Therefore, the performance of IS on these two platforms is likely to be worse than for the fully connected network. From the above observations, we can conclude that IS is not expected to scale well for the chosen problem size on larger configurations of the three hardware platforms. However, if the problem is scaled up, the coefficient associated with the computation time will increase allowing IS to scale better with larger configurations of these architectures.

FFT

TABLE II
FFT : OVERHEAD FUNCTIONS

FFT	Full	Cube	Mesh
Comp. Time (s)	$2.5/p$	$2.5/p$	$2.5/p$
Latency (ms)	$49.9/p^{0.9}$	$49.9/p^{0.9}$	$49.9/p^{0.9}$
Contention (us)	<i>Negligible</i>	<i>Small</i>	$63.5p$

The algorithmic and interaction overheads for FFT are marginal. Thus the delivered curves for all three platforms, as well as the achievable curve, are close to the linear one as shown in Figure 3 (c). The execution time is dominated by the NORMAL mode (Figure 5 (a)). The latency and contention overheads (Figures 5 (b) and (c)) incurred in this mode are insignificant compared to the

total execution time for upto 32 processors.

The communication in FFT has been optimized as suggested in [47] into a single phase where every processor accesses the data of all the other processors in a skewed manner. The number of such non-local accesses incurred by a processor grows as $O((p-1)/p^2)$ with p , and the latency overhead curve reflects this behavior. As a result of skewing the communication among the processors, the contention is negligible on the full and the cube platforms (Figure 5 (d)). On the mesh, the contention surpasses the latency overhead at around 28 processors. Table II summarizes the overheads for FFT obtained by regression analysis of the datapoints from our simulation.

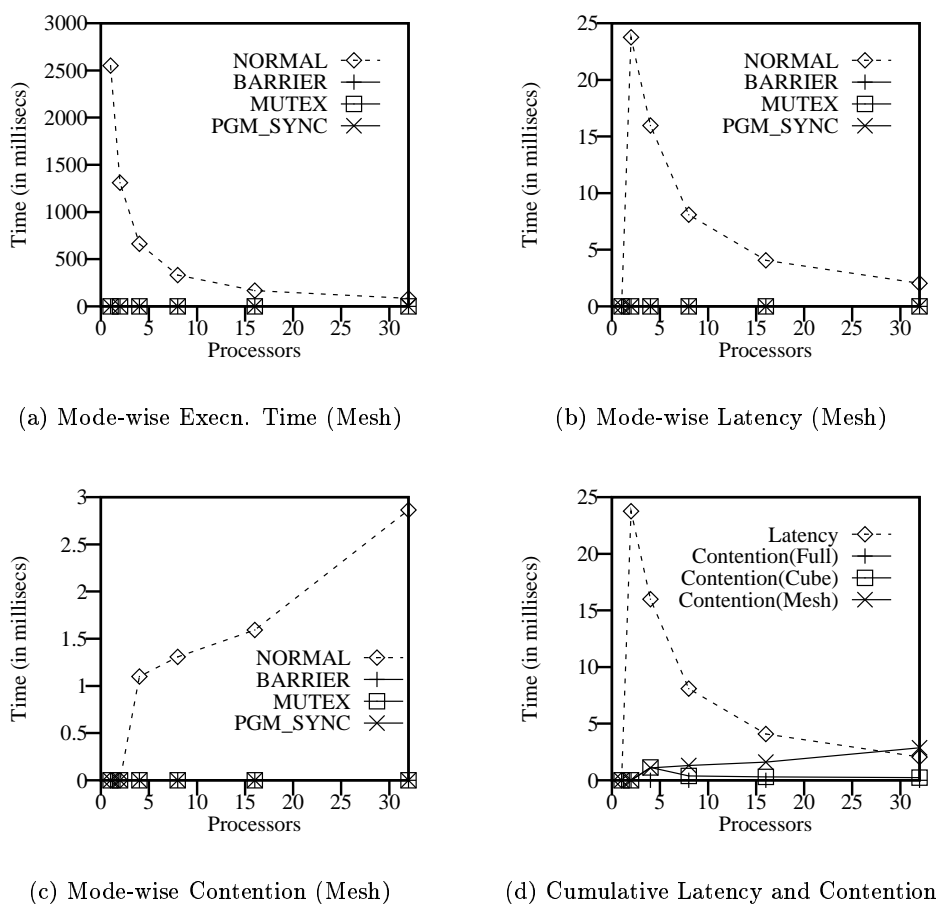


Fig. 5. Performance of FFT

A linear decrease in the computation time with p indicates that the algorithmic overhead is marginal. Of the interaction overheads, the latency component does not play an important role since the number of network accesses incurred by a processor decreases as p is increased for the same problem size. The contention overhead is thus the only artifact that can limit how well FFT can scale with larger

architectural configurations. With skewed communication accesses, the contention overhead has also been minimized and begins to show only on the mesh network where it grows linearly (see Table II). Thus we can conclude that FFT scales well for the fully-connected and cube platforms. For the mesh platform, it would take 200 processors before the contention overhead starts dominating for the 64K problem size. With increase in problem size (n), the local computation that performs a radix-2 Butterfly is expected to grow as $O((n \log n)/p)$ while the communication for a processor is expected to grow as $O(n(p-1)/p^2)$. Hence, increase in data size increases the computation to communication ratio allowing FFT to scale well on larger instances of all three hardware platforms.

CG

The algorithmic and interaction overheads for CG are larger than the corresponding overheads for EP, but not as dominant as in IS (Figure 3 (d)). The NORMAL mode is the only dominant mode of execution as depicted in Figure 6 (a). The communication in the NORMAL mode for data accesses, incurred in calculating a matrix-vector product (the main part of the application), outweighs the overhead in accesses for synchronization variables during the BARRIER and PGM_SYNC modes (Figures 6 (b) and (c)). However, the communication is still insignificant compared to the overall execution time for the range of processors considered.

The fewer the rows assigned to a processor, the fewer will be the number of elements of the vector that may need to be accessed for the matrix-vector product. Therefore, as p increases, the number of rows of the sparse matrix allocated to a processor decreases, thereby decreasing the likelihood of non-local memory references. Hence, the latency overhead decreases with an increase in p . The contention overhead increases from the full to the cube topology and surpasses the latency overhead for the mesh (Figure 6 (d)) at around 17 processors. Table III summarizes the overheads for CG.

TABLE III
CG : OVERHEAD FUNCTIONS

CG	Full	Cube	Mesh
Comp. Time (ms)	$571.2/p$	$571.2/p$	$571.2/p$
Latency (ms)	$2.9/p^{0.4}$	$2.9/p^{0.4}$	$2.9/p^{0.4}$
Contention (us)	<i>Negligible</i>	$8.1p$	$68.2p$

The algorithmic overhead for CG is marginal. Of the interaction overheads, the latency overhead

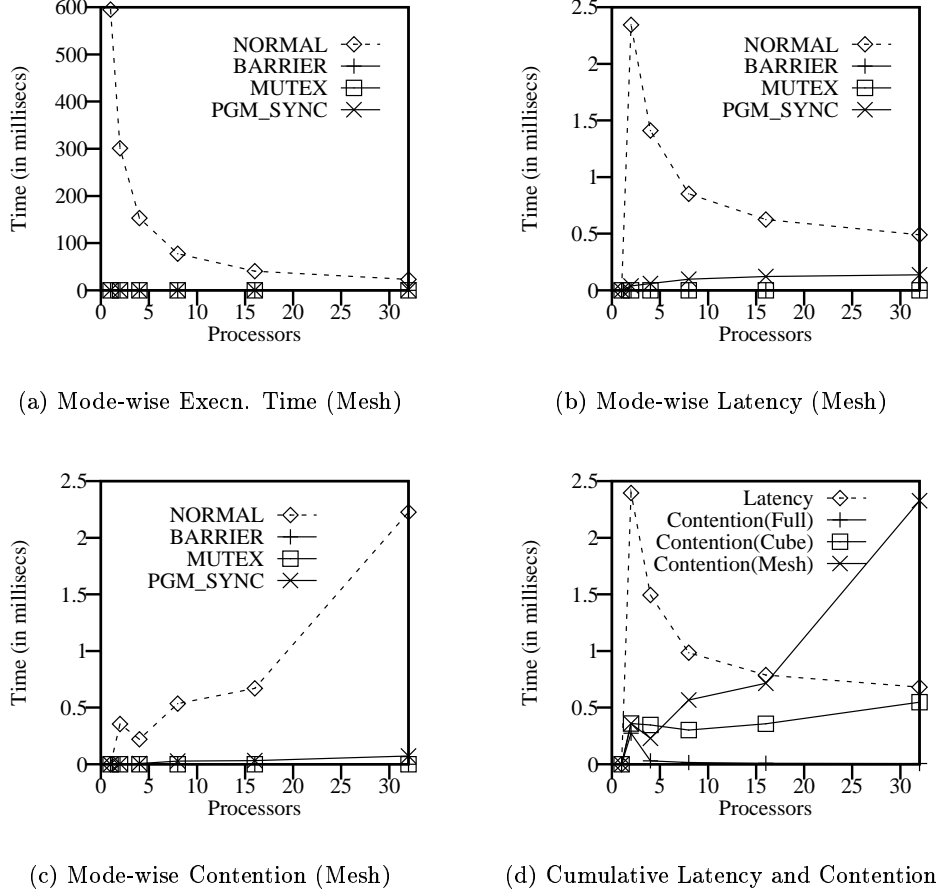


Fig. 6. Performance of CG

decreases with increasing p and the contention overhead is more pronounced. The contention overhead is negligible for the fully-connected network, grows linearly for the cube and the mesh with a larger coefficient for the mesh compared to the cube. CG scales well for the fully-connected network and less so for networks with lower connectivity like the cube and the mesh. Since the NORMAL mode dominates the total execution time, the time spent in the matrix-vector product determines the performance of CG. Scaling the problem size increases the number of non-local memory accesses linearly, while the increase in local computation would be at least linear if not larger. Thus an increase in problem size is likely to make CG scale well for larger instances of all three platforms.

A. Summary

In this section, we have used SPASM to quantify the overheads of specific application-architecture pairs, and to project the growth of these overheads as a function of the problem size of the application as well as the number of processors in the architecture. Algorithmic overheads such as the parallelization

overhead are dominant in limiting the performance of IS and CHOLESKY. SPASM’s ability to pinpoint the region of execution where these overheads are incurred provides useful hints to the programmer for application restructuring. For instance, an initial implementation of IS yielded a contention overhead that far exceeds the one shown in Figure 4 (e). SPASM helped narrow down the problem and discover an alternate implementation where the communication accesses between processors were skewed to reduce the overhead to the one shown in Figure 4 (e).

Isolation of different components of the interaction overheads give feedback to the system architect for making architectural enhancements. On the architectural side, this exercise shows that in shared memory machines with private caches, as long as the applications are well-structured to exploit locality, the key determinant to performance is network contention. This observation leads us to the next exercise in which we use SPASM to determine the bandwidth requirements for specific application-architecture pairs such that the contention is kept within tolerable limits.

VI. SYNTHESIZING BANDWIDTH REQUIREMENTS

Latency and contention, that an application experiences, depend on a number of factors including the connectivity, the bandwidth of the links in the network, the switching delays, and the length of the message. Of these factors, link bandwidth and connectivity are the most crucial network parameters [48]. In this section, we synthesize the bandwidth requirements of the chosen applications for the binary hypercube platform as a function of the following system parameters: the number of processors, CPU clock speed, and application problem size. We also discuss how these results can be used to project the requirements for the other two topologies.

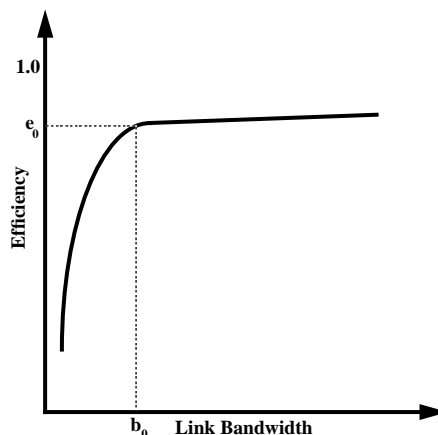


Fig. 7. Link Bandwidth vs. Efficiency

We conduct experiments over a range of processors ($p=4, 8, 16, 32, 64$), CPU clock speeds ($s=33, 100, 300$ MHz) and link bandwidths ($b=1, 20, 100, 200, 600$ and 1000 MBytes/sec). In studying the effect of each parameter, we keep the other two constant. The problem size n of the applications has been varied as 16K, 32K, 64K, 128K and 256K for EP, IS and FFT, 1400×1400 and 5600×5600 for CG, and 1806×1806 for CHOLESKY. We first summarize here the general observations drawn from the experiments.

One would expect that as the link bandwidth is increased, the efficiency of the system would also increase as shown in Figure 7. However, we soon reach a point of diminishing returns beyond which increasing the bandwidth does not have a significant impact on application performance (the curve flattens) since the network overheads are sufficiently low at this point. In all our results, we observe such a distinct knee. Note that the efficiency beyond this knee does not necessarily have to be close to 100% owing to algorithmic overheads (no fault of the hardware). These algorithmic overheads may also cause the curves for each configuration of system parameters to flatten out at entirely different levels in the efficiency spectrum. The bandwidth corresponding to the knee (b_0) still represents an ideal point at which we would like to operate since the network overheads beyond this knee are minimal and the network is no longer the bottleneck for any loss of efficiency.

For all the applications, as expected, the knee shifts to the right as the number of processors is increased indicating the need for higher bandwidth. As the number of processors is increased, the network accesses incurred by a processor in the system may increase or decrease depending on the application, but each such access would incur a larger overhead from contending for network resources (due to the larger number of messages in the network as a whole for the chosen applications). Further, the computation performed by a processor is expected to decrease, lowering the computation to communication ratio, thus making the network requirements more stringent. As the CPU clock speed is increased, the computation to communication ratio decreases. In order to sustain the same efficiency, communication has to be sped up to keep pace with the CPU speed thus shifting the knee in Figure 7 to the right uniformly across all applications. An increase in problem size is likely to increase the amount of computation performed by a processor. At the same time, a larger problem may also increase the network accesses incurred by a processor. In EP, FFT, CG, and CHOLESKY the former effect is more dominant thereby increasing the computation to communication ratio, making the knee

move to the left as the problem size is increased. The two counteracting effects nearly compensate each other in IS showing negligible shift in the knee.

Our objective is to calculate the link bandwidth needed to limit the network overheads to a certain fraction of the overall execution time. This fraction would determine the closeness of the operating point to the knee. For instance, if the network overhead is less than 10% of the overall execution time, then it amounts to saying that we are achieving an efficiency that is within 90% of the ideal efficiency (on a machine with zero network overhead). Ideally, one would like to operate as close to the knee as possible. However, owing to cost or technological constraints, one may be forced to operate at a lower bandwidth and it would be interesting to investigate if it may still be possible to obtain reasonable efficiencies under these constraints. Therefore, we adopt the following methodology. Using the datapoints obtained from the experiments we calculate the bandwidth needed to limit the network overheads to $x\%$ of the total execution time. We consider the requirements for $x = 10\%$, 30% and 50% in the following discussion. In cases where the analysis is simple, we use our knowledge of the application and architectural characteristics in extrapolating the performance for larger systems. In cases where such a static analysis is not possible (due to the dynamic nature of the execution), we perform a non-linear regression analysis of the simulation results using a multivariate secant method with a 95% confidence interval in the SAS [46] statistics package. As mentioned earlier, we intentionally omitted one or two data points in fitting the regression models, and then confirmed that the resulting model does indeed capture these left out points with a high degree of confidence.

In the following discussion, we quantify the bandwidth requirements of IS, FFT and CG using the above methodology, and subsequently summarize the results from studying all five applications. Detailed results for the other applications are given in [40]. For each application we present the intrinsic characteristics that impact its communication and computation requirements; quantify the link bandwidth requirements as a function of increasing number of processors (p), CPU clock speed (s), and problem size (n); and project the requirements for a 1024-node system with a problem size appropriate for such a system. The results presented in the following discussions are for a binary hypercube topology. In section VI-A we discuss how these requirements would change for other topologies.

There are two dominant phases in the execution of IS that account for the bulk of the communication [40]. In both these phases, the communication grows with p . Further, the computation performed by a processor decreases with an increase in processors, but the rate is less than linear owing to algorithmic deficiencies in the problem. These factors combine to yield a considerable bandwidth requirement for larger systems (see Table IV), if we are willing to tolerate less than 10% network overheads. The bandwidth function has been obtained by performing a non-linear regression analysis of the simulation data points for the given system parameter, and the resulting function has been used to calculate the requirements for the 1024 node system. As s is increased, the computation to communication ratio decreases, making the requirements more stringent as shown in Table V. As n is increased, the communication increases linearly. The local computation also increases, but the former effect is more prominent as is shown in Table VI, where the bandwidth requirements grow moderately with problem size.

TABLE IV
IS: IMPACT OF PROCESSORS ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$p=4$	7.75	12.91	68.69
$p=8$	13.38	30.75	92.87
$p=16$	22.00	66.44	168.71
$p=32$	38.65	78.61	211.45
$p=64$	47.03	84.61	293.44
B/w F _{ns.}	$23.60p^{0.28} - 28.79$	$74.41p^{0.22} - 91.21$	$88.68p^{0.34} - 82.12$
$p=1024$	143.25	251.91	907.80

$n=128K, s=33$ MHz

TABLE V
IS: IMPACT OF CPU SPEED ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$s=33$ MHz	47.03	84.61	293.44
$s=100$ MHz	102.49	224.69	770.16
$s=300$ MHz	356.14	649.95	1344.72

$p=64, n=64K$

Using these results, the bandwidth requirements for IS are projected in Table VII for a 1024 node

TABLE VI
IS: IMPACT OF PROBLEM SIZE ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$n=16K$	46.60	83.80	270.08
$n=32K$	47.16	84.52	286.98
$n=64K$	47.03	84.61	293.44
$n=128K$	47.48	85.09	303.41
$n=256K$	48.67	85.53	307.75
B/w Fns.	$0.007n^{1.00} + 46.61$	$0.006n^{0.99} + 84.10$	$19.57n^{0.26} + 230.19$
$n=8192K$	110.75	133.19	441.66

$p=64, s=33$ MHz

TABLE VII
IS: LINK BANDWIDTH PROJECTIONS (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$s=33MHz$	337.34	396.55	1366.34
$s=100MHz$	735.15	1053.08	3586.08
$s=300MHz$	> 5000	> 5000	> 5000

$p = 1024, n = 2^{23}$

system and a problem size of 2^{23} that is representative of a real world problem [38]. This table shows that bandwidth requirements of IS are considerably high. We may at best be able to operate at around 50% network overhead range with 33 MHz processors given that link bandwidth of state-of-the-art networks is around 200-300 MBytes/sec. With faster processors, the network becomes an even bigger bottleneck for this application.

In projecting the above bandwidth requirements of IS (which performs a bucket sort of a list of integers) with 1024 processors, both the number of buckets as well as the number of list elements to be sorted have been increased for the larger problem. Bucket sort is, however, frequently used in cases where the number of buckets is relatively independent of the number of elements in the list to be sorted. A scaling strategy where the size of the list is increased and the number of buckets is maintained constant would cause no change in communication in the above mentioned phases of IS, while the computation is expected to grow as $O(n)$. Hence, only under such a scaling strategy, would we be able to limit the network overheads to within 30-50% for this application with existing technology.

FFT

The computation performed by a processor in FFT grows as $O((n \log n)/p)$ while the communication grows as $O(n(p-1)/p^2)$. Thus, these components decrease at comparable rates with an increase in p . As p is increased, the contention encountered by each message in the network is expected to grow. However, due to skewing of the communication accesses [47] the bandwidth requirements of the network grow slowly as is shown in Table VIII. These requirements can be satisfied even for faster processors (see Table IX). As we mentioned earlier, the computation to communication ratio is proportional to $O(\log n)$, and the network requirements are expected to become even less stringent as the problem size is increased. Table X confirms this observation.

TABLE VIII
FFT: IMPACT OF PROCESSORS ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$p=4$	< 1.0	6.40	16.35
$p=8$	< 1.0	6.52	16.40
$p=16$	< 1.0	7.52	16.75
$p=32$	< 1.0	7.83	16.87
$p=64$	< 1.0	8.65	17.19
B/w Fns.	-	$0.75p^{0.36} + 5.11$	$0.01p^{0.99} + 16.37$
$p=1024$	-	14.85	29.93

$n=64K, s=33$ MHz

In projecting the requirements for a 1024-node system, link bandwidths of around 100-150 MBytes/sec would suffice to limit the network overheads to less than 10% of the execution time (see Table XI). The results shown in the above tables confirm the theoretical results presented in [49] where the authors show that FFT scales well on the hypercube topology and the achievable efficiency is only limited by the ratio of the CPU clock speed and the link bandwidth.

TABLE IX
FFT: IMPACT OF CPU SPEED ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$s=33$ MHz	< 1.0	8.65	17.19
$s=100$ MHz	8.65	13.81	29.86
$s=300$ MHz	17.19	29.20	88.81

$p=64, n=64K$

TABLE X
FFT: IMPACT OF PROBLEM SIZE ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$n=16K$	< 1.0	9.42	17.63
$n=32K$	< 1.0	9.03	17.45
$n=64K$	< 1.0	8.65	17.19
$n=128K$	< 1.0	8.38	17.03
$n=256K$	< 1.0	7.97	16.84
B/w Fns.	-	$11.02 - 0.4 \log n$	$18.43 - 0.2 \log n$
$n=2^{30}$	-	3.02	14.43

$p=64, s=33$ MHz

TABLE XI
FFT: LINK BANDWIDTH PROJECTIONS (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$s=33$ MHz	-	5.15	21.64
$s=100$ MHz	-	8.22	48.58
$s=300$ MHz	-	17.38	144.50

$p = 1024, n = 2^{30}$

CG

The main communication in CG occurs in the multiplication of a sparse matrix with a dense vector. Each processor performs this operation for a contiguous set of rows allocated to it. The elements of the vector that are needed by a processor to perform this operation depend on the distribution of non-zero elements in the matrix and may involve external accesses. Once an element is fetched, a processor may reuse it for a non-zero element in another row of the same column. As p is increased, the number of rows allocated to a processor decreases thus decreasing the computation that it performs. Increasing p has a dual impact on communication. Since the number of rows that need to be computed decreases, the probability of external accesses decreases. There is also a decreased probability of reusing a fetched data item for computing another row. These complicated interactions are to a large extent dependent on the input data and are difficult to analyze statically. We use the data sets supplied with the NAS benchmarks [38]. The results from our simulation are given in Table XII. We observe that the effect of lower local computation, and lesser data reuse has a more significant impact in increasing the communication requirements for larger systems. The rate at which the bandwidth requirements increase for higher clock speeds is shown in Table XIII. As n is increased, the local computation

increases, and the probability of data re-use also increases. The rate at which these factors impact the requirements depends on the sparsity factor of the matrix. Table XIV shows the requirements for two different problem sizes. For the 1400×1400 problem, the sparsity factor is 0.04, while the sparsity factor for the 5600×5600 problem is 0.02. The corresponding factor for the 14000×14000 problem suggested in [38] is 0.1 and we scale down the bandwidth requirements accordingly in Table XV for a 1024 node system. The results suggest that we may be able to limit the overheads to within 50% of the execution time with existing technology. As the processors get faster than 100 MHz, it would need a considerable amount of bandwidth to limit the overheads to within 30%. With faster processors, and larger system configurations, one may expect to solve larger problems as well. If we increase the problem size (number of rows of the matrix) linearly with the clock speed of the processor, one can expect the bandwidth requirements to remain constant, and we may be able to limit network overheads to within 30% of execution time even with existing technology.

TABLE XII
CG: IMPACT OF PROCESSORS ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$p=4$	1.74	2.90	8.71
$p=8$	3.25	5.41	16.23
$p=16$	5.81	9.68	52.03
$p=32$	9.73	16.22	82.39
$p=64$	15.63	46.10	124.19
B/w Fns.	$1.25p^{0.62} - 1.28$	$0.04p^{1.63} + 3.61$	$18.80p^{0.51} - 33.07$
$p=1024$	94.79	393.32	618.28

$n=1400*1400, s=33$ MHz

TABLE XIII
CG: IMPACT OF CPU SPEED ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$s=33$ MHz	15.63	46.10	124.19
$s=100$ MHz	43.50	96.75	386.12
$s=300$ MHz	120.89	262.84	1022.14

$p=64, n=1400*1400$

TABLE XIV
CG: IMPACT OF PROBLEM SIZE ON LINK BANDWIDTH (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$n=1400*1400$	15.63	46.10	124.19
$n=5600*5600$	9.48	25.47	78.55

$p=64, s=33$ MHz

TABLE XV
CG: LINK BANDWIDTH PROJECTIONS (IN MBYTES/SEC)

	50% ovhd.	30% ovhd.	10% ovhd.
$s=33$ MHz	34.87	120.04	247.33
$s=100$ MHz	97.05	251.93	1200.49
$s=300$ MHz	269.7	684.41	2035.64

$p=1024, n=14000*14000$

A. Summary

We have quantified the link bandwidth requirements of five applications for the binary hypercube topology as a function of the number of processors, CPU clock speed and problem size [40]. Based on these results we have projected the requirements of large systems built with 1024 processors and CPU clock speeds upto 300 MHz. EP has negligible bandwidth requirements and FFT has moderate requirements that can be easily sustained. The network overheads for IS, CG and CHOLESKY may be maintained at an acceptable level for current day processors, and as the processor speed increases, one may still be able to tolerate these overheads with existing link bandwidth technologies of 200-300 MBytes/sec provided the problem size is increased commensurately.

The results have been presented for the binary hypercube network topology. The cube represents a highly scalable network where the bisection bandwidth grows linearly with the number of processors. Even though cubes of 1024 nodes have been built [2], cost and technology factors often play an important role in its physical realization. Agarwal [11] and Dally [12] show that wire delays (due to increased wire lengths associated with planar layouts) of higher dimensional networks make low dimensional networks more viable. The 2-dimensional [50] and 3-dimensional [51], [52] toroids are common topologies used in current day networks, and it would be interesting to project link bandwidth requirements for these topologies.

A metric that is often used to compare different networks is the bisection bandwidth available per

processor. On a k -ary n -cube, the bisection bandwidth available per processor is inversely proportional to the radix k of the network. One may use a simple rule of thumb of maintaining per processor bisection bandwidth constant in projecting requirements for lower connectivity networks. This argument is very similar to the rationale behind the g parameter of LogP [47] which models network contention. For example, considering a 1024-node system, the link bandwidth requirement for a 32-ary 2-cube would be 16 times the 2-ary 10-cube bandwidth; similarly the requirement for a 3-D network would be around 5 times the 10-D network. However, such a projection becomes pessimistic since it assumes that the communication in an application is devoid of any network locality and that each message crosses the bisection. With a little knowledge about the communication behavior of applications, one may be able to reduce the degree of pessimism. In both FFT and IS, every processor communicates with all other processors, and thus only 50% of the messages cross the bisection. Similarly, instrumentation in our simulation showed that only around 50% of the messages in CG and CHOLESKY traverse the bisection. To reduce the degree of pessimism in these projections, one may thus introduce a correction factor of 0.5 that can be multiplied with the above-mentioned factors of 16 and 5 in projecting the bandwidths for 2-D and 3-D networks respectively. EP would still need negligible bandwidth and we can still limit network overheads of FFT to around 30% on these networks with existing technology. The problem sizes for IS, CG and CHOLESKY would have to grow by a factor of 8 compared to their hypercube counterparts if we are to sustain the corresponding efficiency on a 2-D network with current technology. Despite the correction factor, these projections are still expected to be pessimistic since the temporal aspect of communication is ignored. The projection assumes that every message in the system traverses the bisection at the same time. If the message pattern is temporally skewed, then a lower link bandwidth may suffice for a given network overhead. It is almost impossible to determine these skews statically, especially for applications like CG and CHOLESKY where the communication pattern is dynamic. It is necessary to conduct a detailed simulation for these network topologies to confirm these projections, but that is beyond the scope of this paper.

Using the outlined technique, it would thus be possible for an architect to synthesize the bandwidth requirements of an application as a function of system parameters. For instance, given a set of applications, the system size (number of processors) and the CPU speed, an architect may use this technique to calculate the bandwidth that needs to be supported in hardware. In cases where

cost/technological problems prohibit supporting this bandwidth, the architect may use the results to find out the efficiency that would result from a lower hardware bandwidth or the factor by which the problem size needs to be scaled to maintain good efficiency. The results may also be used to quantify the rate at which the network (which is often custom-built) capabilities have to be enhanced in order to accommodate the rapidly improving off-the-shelf components used in realizing the processing nodes.

VII. CONCLUDING REMARKS AND FUTURE WORK

Evaluating and analyzing the performance of an application on an architectural platform has widespread applicability in parallel systems research. However, conducting such a study is hard because of the complex interaction between application characteristics and architectural features. In this paper, we considered five applications on three architectural platforms. We quantified the algorithmic and network overheads for these combinations of application-architecture pairs. The key result of this part of the study revealed that network contention is a dominant impediment to performance for these combinations. Armed with this knowledge, we synthesized the bandwidth requirements for these application on the binary hypercube network topology as a function of various system parameters. We also projected the bandwidth requirements for other topologies from these results. We showed that a link bandwidth of 200-300 MBytes/sec, available with current network technologies is sufficient in most situations. Wherever this does not suffice, we have quantified the rate at which application problem sizes must be scaled up to keep network overheads within a tolerable factor.

We also presented the design and implementation of an execution-driven simulator called SPASM, which was used in this study. SPASM is unique in that it provides a separation of all the parallel system overheads, a feature that is crucial in understanding the complex interaction between applications and architectures.

As we mentioned in Section III, execution-driven simulation of large parallel systems with real applications may become impractical because of the tremendous computational and storage resources that is needed. In such cases, experimentation (measurement) on the actual system or analytical modeling techniques are more useful. However, these techniques have their own limitations as discussed in Section III. We are investigating an integrated evaluation strategy that combines the strengths of all three techniques.

REFERENCES

- [1] G. M. Amdahl, "Validity of the Single Processor Approach to achieving Large Scale Computing Capabilities," in *Proceedings of the AFIPS Spring Joint Computer Conference*, April 1967, pp. 483–485.
- [2] J. L. Gustafson, G. R. Montry, and R. E. Benner, "Development of Parallel Methods for a 1024-node Hypercube," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 4, pp. 609–638, 1988.
- [3] X-H. Sun and J. L. Gustafson, "Towards a better Parallel Performance Metric," *Parallel Computing*, vol. 17, pp. 1093–1109, 1991.
- [4] V. Kumar and V. N. Rao, "Parallel Depth-First Search," *International Journal of Parallel Programming*, vol. 16, no. 6, pp. 501–519, 1987.
- [5] A. H. Karp and H. P. Flatt, "Measuring Parallel processor Performance," *Communications of the ACM*, vol. 33, no. 5, pp. 539–543, May 1990.
- [6] D. Nussbaum and A. Agarwal, "Scalability of Parallel Machines," *Communications of the ACM*, vol. 34, no. 3, pp. 57–61, March 1991.
- [7] J. C. Wyllie, *The Complexity of Parallel Computations*, Ph.D. thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1979.
- [8] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran, "A Simulation-based Scalability Study of Parallel Systems," *Journal of Parallel and Distributed Computing*, vol. 22, no. 3, pp. 411–426, September 1994.
- [9] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran, "An Approach to Scalability Study of Shared Memory Parallel Systems," in *Proceedings of the ACM SIGMETRICS 1994 Conference on Measurement and Modeling of Computer Systems*, May 1994, pp. 171–180.
- [10] M. E. Crovella and T. J. LeBlanc, "Parallel Performance Prediction Using Lost Cycles Analysis," in *Proceedings of Supercomputing '94*, November 1994.
- [11] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, October 1991.
- [12] W. J. Dally, "Performance analysis of k -ary n -cube interconnection networks," *IEEE Transactions on Computer Systems*, vol. 39, no. 6, pp. 775–785, June 1990.
- [13] V. S. Adve and M. K. Vernon, "Performance analysis of mesh interconnection networks with deterministic routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 3, pp. 225–246, March 1994.
- [14] S. L. Scott and J. R. Goodman, "Impact of pipelined channels on k -ary n -cube networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 1, pp. 2–16, January 1994.
- [15] T. E. Anderson, "The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1, pp. 6–16, January 1990.
- [16] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors," *ACM Transactions on Computer Systems*, vol. 9, no. 1, pp. 21–65, February 1991.
- [17] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural requirements of parallel scientific applications with explicit communication," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993, pp. 2–13.
- [18] E. Rothberg, J. P. Singh, and A. Gupta, "Working sets, cache sizes and node granularity issues for large-scale multiprocessors," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993, pp. 14–25.
- [19] W. B. Ligon III and U. Ramachandran, "Toward a more realistic performance evaluation of interconnection networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 7, July 1997.
- [20] A. Sivasubramaniam, U. Ramachandran, and H. Venkateswaran, "A comparative evaluation of techniques for studying parallel system performance," Tech. Rep. GIT-CC-94/38, College of Computing, Georgia Institute of Technology, September 1994.
- [21] J. P. Singh, E. Rothberg, and A. Gupta, "Modeling communication in parallel algorithms: A fruitful interaction between theory and systems?," in *Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures*, 1994.
- [22] M. Martonosi, A. Gupta, and T. Anderson, "MemSpy: Analyzing Memory System Bottlenecks in Programs," in *Proceedings of the ACM SIGMETRICS 1992 Conference on Measurement and Modeling of Computer Systems*, June 1992, pp. 1–12.
- [23] E. Speight, "Paraview: Performance Debugging Through Visualization of Shared Data," M.S. thesis, Department of Electrical and Computer Engineering, Rice University, Houston, TX, September 1993.
- [24] P. Mehra, C. H. Schulbach, and J. C. Yan, "A comparison of two model-based performance-prediction techniques for message-passing parallel programs," in *Proceedings of the ACM SIGMETRICS 1994 Conference on Measurement and Modeling of Computer Systems*, May 1994, pp. 181–190.
- [25] M. T. Heath and J. A. Etheridge, "Visualizing the Performance of Parallel Programs," *IEEE Software*, vol. 8, no. 9, pp. 29–39, September 1991.
- [26] B. P. Miller et al., "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, vol. 28, no. 11, pp. 37–46, November 1995.
- [27] D. A. Reed et al., "Virtual Reality and Parallel Systems Performance Analysis," *IEEE Computer*, vol. 28, no. 11, pp. 57–67, November 1995.
- [28] Microelectronics and Computer Technology Corporation, Austin, TX 78759, *CSIM User's Guide*, 1990.
- [29] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl, "PROTEUS : A high-performance parallel-architecture simulator," Tech. Rep. MIT-LCS-TR-516, Massachusetts Institute of Technology, Cambridge, MA 02139, September 1991.
- [30] R. G. Covington, S. Madala, V. Mehta, J. R. Jump, and J. B. Sinclair, "The Rice parallel processing testbed," in *Proceedings of the ACM SIGMETRICS 1988 Conference on Measurement and Modeling of Computer Systems*, Santa Fe, NM, May 1988, pp. 4–11.
- [31] H. Davis, S. R. Goldschmidt, and J. L. Hennessy, "Multiprocessor Simulation and Tracing Using Tango," in *Proceedings of the 1991 International Conference on Parallel Processing*, 1991, pp. II 99–107.

- [32] J. E. Veenstra and R. J. Fowler, "MINT: A Front End for Efficient Simulation of Shared Memory Multiprocessors," in *Proceedings of MASCOTS '94*, February 1994, pp. 201–207.
- [33] S. K. Reinhardt et al., "The Wisconsin Wind Tunnel : Virtual prototyping of parallel computers," in *Proceedings of the ACM SIGMETRICS 1993 Conference on Measurement and Modeling of Computer Systems*, Santa Clara, CA, May 1993, pp. 48–60.
- [34] M. Rosenblum, S. Harrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology*, Fall 1995.
- [35] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran, "Abstracting network characteristics and locality properties of parallel systems," in *Proceedings of the First International Symposium on High Performance Computer Architecture*, January 1995, pp. 54–63.
- [36] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran, "On characterizing bandwidth requirements of parallel applications," in *Proceedings of the ACM SIGMETRICS 1995 Conference on Measurement and Modeling of Computer Systems*, May 1995, pp. 198–207.
- [37] U. Ramachandran, G. Shah, A. Sivasubramaniam, A. Singla, and I. Yanasak, "Architectural mechanisms for explicit communication in shared memory multiprocessors," in *Proceedings of Supercomputing '95*, December 1995.
- [38] D. Bailey et al., "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, 1991.
- [39] J. P. Singh, W-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," Tech. Rep. CSL-TR-91-469, Computer Systems Laboratory, Stanford University, 1991.
- [40] A. Sivasubramaniam, *A Framework for Evaluating Architectural Issues of Parallel Systems*, Ph.D. thesis, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, August 1995.
- [41] H. Sullivan and T. R. Bashkow, "A large scale, homogenous, fully-distributed parallel machine," in *Proceedings of the 4th Annual Symposium on Computer Architecture*, March 1977, pp. 105–117.
- [42] Kendall Square Research, "Technical summary," 1992.
- [43] E. H. Welbon, C. C. Chan-Nui, D. J. Shippey, and D. A. Hicks, "The POWER2 performance monitor," *IBM Journal of Research and Development*, vol. 38, no. 5, pp. 545–554, September 1994.
- [44] J. Maki, "POWER2 Hardware Performance Monitor Tools," November 1995.
- [45] U. Ramachandran, G. Shah, S. Ravikumar, and J. Muthukumarasamy, "Scalability study of the KSR-1," in *Proceedings of the 1993 International Conference on Parallel Processing*, August 1993, pp. 1–237–240.
- [46] SAS Institute Inc., Cary, NC 27512, *SAS/STAT User's Guide*, 1988.
- [47] D. Culler et al., "LogP : Towards a realistic model of parallel computation," in *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993, pp. 1–12.
- [48] A. Vaidya, A. Sivasubramaniam, and C. Das, "Performance Benefits of Virtual Channels and Adaptive Routing: An Application-driven Study," in *Proceedings of the ACM 1997 International Conference on Supercomputing*, July 1997, pp. 140–147.
- [49] A. Gupta and V. Kumar, "The scalability of FFT on parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 8, pp. 922–932, August 1993.
- [50] D. Lenoski et al., "The Stanford DASH multiprocessor," *IEEE Computer*, vol. 25, no. 3, pp. 63–79, March 1992.
- [51] Cray Research, Inc., Minnesota, *The Cray T3D System Architecture Overview Manual*, 1993.
- [52] R. Alverson et al., "The Tera Computer System," in *Proceedings of the ACM 1990 International Conference on Supercomputing*, Amsterdam, Netherlands, 1990, pp. 1–6.

Biographical Sketch

Anand Sivasubramaniam received his B.Tech in computer science from the Indian Institute of Technology, Madras in 1989, and the MS and Ph.D. degrees in computer science from the Georgia Institute of Technology in 1991 and 1995 respectively. Currently, he is an Assistant Professor of Computer Science and Engineering at the Pennsylvania State University. His research interests are in the areas of computer architecture, operating systems and performance evaluation, with a focus on parallel and distributed computing. He is the recipient of a NSF Career Award, and is a member of ACM, IEEE and IEEE Computer Society.

Aman Singla currently works in the Video Computing Research and Development Division at Silicon Graphics, Inc. He received his Ph.D. in Computer Science from the Georgia Institute of Technology, Atlanta, in 1997. His research interests are primarily in the area of high performance parallel and distributed systems.

Umakishore Ramachandran received his Ph.D. in Computer Science from the University of Wisconsin, Madison in 1986. He has been with the Georgia Institute of Technology since then, where he is an Associate Professor in the College of Computing. He has worked extensively in the architectural design, programming, and analysis of parallel and distributed systems. His current research interests include cluster computing techniques for interactive multimedia applications. He jointly initiated the Stampede project at Compaq CRL, while on sabbatical leave from Georgia Tech in 1996-97. He is the recipient of a Presidential Young Investigator Award from the National Science Foundation, and the 1993 Georgia Tech doctoral thesis advisor award.

H. Venkateswaran is on the faculty of the College of Computing at the Georgia Institute of Technology which he joined after getting his Ph.D. from the University of Washington, Seattle in 1986. His research interests are in the areas of computational complexity theory and parallel computation.