To appear in *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1995.

On Characterizing Bandwidth Requirements of Parallel Applications[*]

Anand Sivasubramaniam *Aman Singla* *Umakishore Ramachandran* *H. Venkateswaran*

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280.
{*anand, aman, rama, venkat*}*@cc.gatech.edu*

Abstract

Synthesizing architectural requirements from an application viewpoint can help in making important architectural design decisions towards building large scale parallel machines. In this paper, we quantify the link bandwidth requirement on a binary hypercube topology for a set of five parallel applications. We use an execution-driven simulator called SPASM to collect data points for system sizes that are feasible to be simulated. These data points are then used in a regression analysis for projecting the link bandwidth requirements for larger systems. The requirements are projected as a function of the following system parameters: number of processors, CPU clock speed, and problem size. These results are also used to project the link bandwidths for other network topologies. Our study quantifies the link bandwidth that has to be made available to limit the network overhead in an application to a specified tolerance level. The results show that typical link bandwidths (200-300 MBytes/sec) found in current commercial parallel architectures (such as Intel Paragon and Cray T3D) would have fairly low network overhead for the applications considered in this study. For two of the applications, this overhead is negligible. For the other applications, this overhead can be limited to about 30% of the execution time provided the problem sizes are increased commensurate with the processor clock speed. The technique presented can be useful to a system architect to synthesize the bandwidth requirements for realizing well-balanced parallel architectures.

1 Introduction

Parallel machines promise to solve computationally intensive problems that may not be feasibly computed due to resource limitations on sequential machines. Despite this promise, the delivered performance of these machines often falls short of the projected peak performance. The disparity between the expected and observed performance may be due to application deficiencies such as serial fraction and work-imbalance, software slow-down due to scheduling and runtime overheads, and hardware slow-down stemming from the synchronization and communication requirements in the application. For building a general-purpose parallel machine, it is essential to identify and quantify the architectural requirements necessary to assure good performance over a wide range of applications. Such a synthesis of requirements from an application view-point can help us make cost vs. performance trade-offs in important architectural design decisions. The network is an important artifact in a parallel machine limiting the performance of many applications, and is the focus of our study. Using five parallel applications, we quantify bandwidth requirements needed to limit the overheads arising from the network to an acceptable level for these applications.

Latency and *contention* are two defining characteristics of a network from the application viewpoint. Latency is the sum of the time spent in transmitting a message over the network links and the switching time assuming that the message did not have to contend for any network links. Contention is the time spent by a message in the network waiting for links to become free. Both latency and contention 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 the architectural factors, link bandwidth and connectivity are the most crucial network parameters impacting latency and contention. Hence, in order to quantify requirements that limit network overheads (latency and contention) to an acceptable level, it is necessary to study the impact of link bandwidth and network connectivity on these overheads.

Dally [8] and Agarwal [2] present analytical models to study the impact of network connectivity and link bandwidth for k-ary n-cube networks. The results suggest that low-dimensional networks are preferred (based on physical and technological constraints) when the network contention is ignored or when the workload (the application) exhibits sufficient network locality; and that higher dimensional networks may be needed otherwise. Adve and Vernon [1] show using analytical models that network locality has an important role to play in the performance of the mesh. Since network requirements are sensitive to the workload, it is necessary to study them in the context of real applications.

The RISC ideology clearly illustrates the importance of using real applications in synthesizing architectural requirements. Several researchers have used this approach for parallel architectural studies [20, 7, 13]. Cypher et al. [7] use a range of scientific applications in quantifying the processing, memory, communication and I/O requirements. They present the communication requirements in terms of the number of messages exchanged between processors and the volume (size) of these messages. As identified in [22], communication in parallel applications may be categorized by the following attributes: communication volume, the communication pattern, the communication frequency and the ability to overlap communication with computation. A static analysis of the communication as conducted in [7] fails to capture the last two attributes, making it very difficult to quantify the contention in the system.

The importance of simulation in capturing the dynamics of parallel system (an application-architecture combination) behavior has been clearly illustrated in [12, 22, 25]. In particular, using an execution-driven simulator, one can faithfully capture all the attributes of communication that are important to network requirements synthesis. For example, in [12] the authors use an execution-

[*]This work has been funded in part by NSF grants MIPS-9058430 and MIPS-9200005, and grants from DEC, SYSTRAN, and IBM.

driven simulator to study k-ary n-cube networks in the context of applications drawn from image understanding, and show the impact of application characteristics on the choice of the network topology. We take a similar approach to deriving the network requirements in this study.

Using an execution-driven simulation platform called SPASM [27, 25], we simulate the execution of five parallel applications on an architectural platform with a binary hypercube network topology. We vary the link bandwidth on the hypercube and quantify its impact on application performance. From these results, we derive link bandwidths that are needed to limit network overheads to an acceptable level. We also study the impact of the number of processors, the CPU clock speed and the application problem size on bandwidth requirements. Using regression analysis and application knowledge, we extrapolate requirements for larger systems of 1024 processors and other network topologies. The results suggest that existing link bandwidth of 200-300 MBytes/sec available on machines like Intel Paragon [14] and Cray T3D [17] can easily sustain the requirements of two applications (EP and FFT) even on high-speed processors of the future. For the other three, one may be able to maintain network overheads at an acceptable level if the problem size is increased commensurate with the processing speed.

The technique outlined in this paper may be used to derive the bandwidth requirements of an application as a function of the system parameters (such as processor clock speed, number of processors, and expected problem size of applications). This information can then be used by an architect to deduce the network requirements for a specific set of system parameters. When cost and technological factors prohibit supporting the required bandwidth, the information may be useful to find out the efficiency that would result from a lower bandwidth or the factor by which the problem size needs to be scaled up to maintain reasonable efficiency.

Section 2 gives an overview of our approach and details of the simulation platform; section 3 briefly describes the hardware platform and applications used in this study; section 4 presents results from our experiments along with an analysis of bandwidth requirements as a function of system parameters; section 5 summarizes the implication of these results; and section 6 presents concluding remarks.

2 Approach

As observed in [22], communication in an application may be characterized by four attributes. *Volume* refers to the number and size of messages. The *communication pattern* in the application determines the source-destination pairs for the messages, and reflects on the application's ability to exploit network locality. *Frequency* pertains to the temporal aspects of communication, i.e., the interval between successive network accesses by each processor as well as the interleaving in time of accesses from different processors. This temporal aspect of communication would play an important role in determining network contention. *Tolerance* is the ability of an application to hide network overheads by overlapping computation with communication. Modeling all these attributes of communication in a parallel application is extremely difficult by simple static analysis. Further, the dynamic access patterns exhibited by many applications makes modeling more complex. Several researchers [22, 25, 12] have observed the importance of simulation for capturing the communication behavior of applications.

In this study, we use an execution-driven simulator called SPASM (Simulator for Parallel Architectural Scalability Measurements) [27, 25] that enables us to accurately model the behavior of applications on a number of simulated hardware platforms. SPASM has been written using CSIM [16], a process oriented sequential simulation package, and currently runs on SPARCstations. The input to the simulator are parallel applications written in C. These programs are preprocessed (to label shared memory accesses), the compiled assembly code is augmented with cycle counting instructions, and the assembled binary is linked with the simulator code. As with other recent simulators [5, 9, 6, 19], bulk of the instructions is executed at the speed of the native processor (the SPARC in this case) and only instructions (such as LOADs and STOREs on a shared memory platform or SENDs and RECEIVEs on a message-passing platform) that may potentially involve a network access are simulated. The input parameters that may be specified to SPASM are the number of processors, the CPU clock speed, the network topology, the link bandwidth and switching delays. We can thus vary a range of system parameters and study their impact on application performance. The main problem with the execution-driven simulation approach is the tremendous resource (both space and time) requirement in simulating large parallel systems. Related studies [28, 24] address this problem and show how it may be alleviated by augmenting simulation with other evaluation techniques.

SPASM gives a wide range of statistical information about the execution of the program. The novel feature of SPASM is its ability to provide a complete isolation and quantification of different overheads in a parallel system that limit its scalability. The algorithmic overhead (arising from factors such as the serial part and work-imbalance in the algorithm) and the network overheads (latency and contention) are the important overheads that are of relevance to this study. SPASM gives the *total time* (simulated time) which is the maximum of the running times of the individual parallel processors. This is the time that would be taken by an execution of the parallel program on the target parallel machine. SPASM also gives the *ideal time*, which is the time taken by the parallel program to execute on an ideal machine such as the PRAM [31]. This metric includes the algorithmic overheads but does not include any overheads arising from architectural limitations. Of the network overheads, the time that a message would have taken in a contention free environment is charged to the latency overhead, while the rest of the time is accounted for in the contention overhead.

To synthesize the communication requirements of parallel applications, the separation of the overheads provided by SPASM is crucial. For instance, an application may have an algorithmic deficiency due to either a large serial part or due to work-imbalance, in which case 100% efficiency[1] is impossible regardless of other architectural parameters. The separation of overheads, provided by SPASM, enables us to quantify the bandwidth requirements as a function of acceptable network overheads (latency and contention). We thus quantify the bandwidth needed to limit the network overheads to 10%, 30% and 50% of the overall execution time. This also amounts to quantifying the bandwidth needed to attain an efficiency which is 90%, 70% and 50% of the ideal efficiency (on an ideal machine with zero network overhead).

3 Experimental Setup

We have chosen a CC-NUMA (Cache Coherent Non-Uniform Memory Access) shared memory multiprocessor as the architectural platform for this study. Since uniprocessor architecture is getting standardized with the advent of RISC technology, we fix most of the processor characteristics by using a SPARC chip as the baseline for each processor in a parallel system. But to study the impact of processor speed on network requirements we allow the clock speed of a processor to be varied. Each node in the system has a piece of the globally shared memory and a 2-way set-associative private cache (64KBytes with 32 byte blocks). The cache is maintained sequentially consistent using an invalidation-based fully-mapped directory-based cache coherence scheme. Rothberg et al. [20]

[1] *Efficiency* is defined as $speedup(p)/p$ where p is the number of processors. *Speedup(p)* is the ratio of the time taken to execute the parallel application on 1 processor to the time taken to execute the same on p processors.

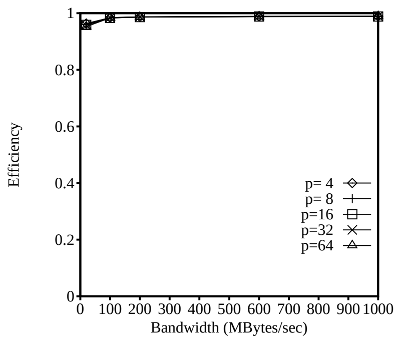

Figure 4: FFT (n=64K, s=33 MHz)

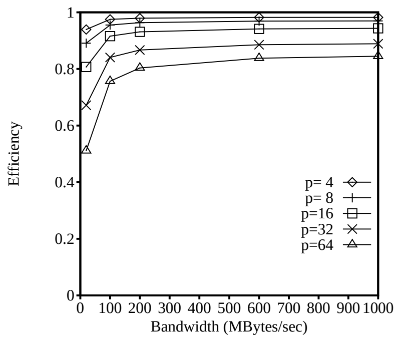

Figure 5: CG (n=1400*1400, s=33 MHz)

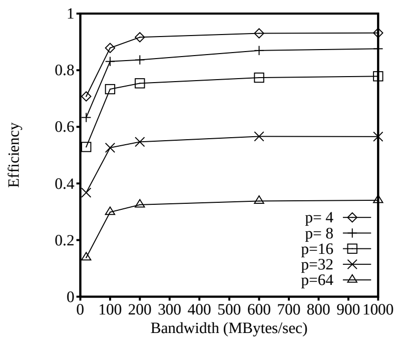

Figure 6: CHOLESKY (n=1806*1806, s=33 MHz)

Figures 2, 3, 4, 5 and 6 show the impact of varying link bandwidth on the efficiency of EP, IS, FFT, CG and CHOLESKY respectively, across different number of processors (p). The knees for EP and FFT, which display a high computation to communication ratio, occur at low bandwidths and are hardly noticeable in these figures. The algorithmic overheads in these applications is negligible yielding efficiencies that are close to 100%. For the other applications, the knee occurs at a higher bandwidth. Further, the curves tend to flatten at different efficiencies suggesting the presence of algorithmic overheads. For all the applications, 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.

Impact of CPU clock speed

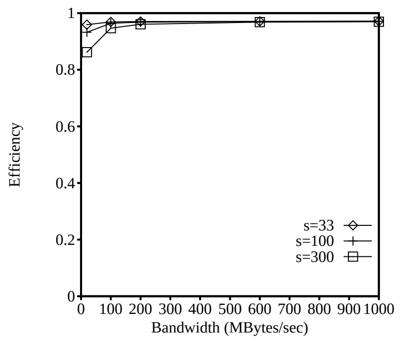

Figure 7: EP (p=64, n=128K)

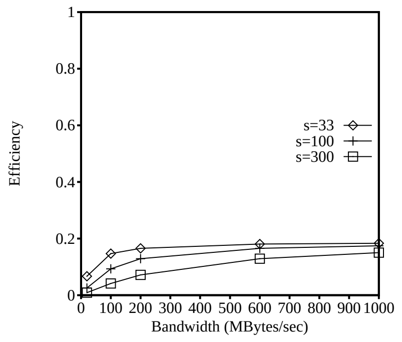

Figure 8: IS (p=64, n=64K)

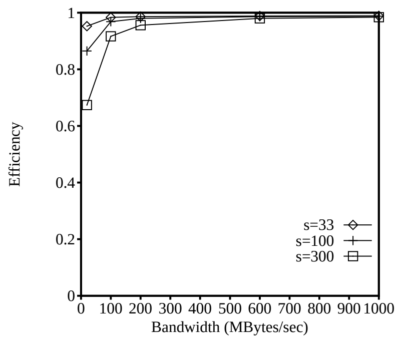

Figure 9: FFT (p=64, n=64K)

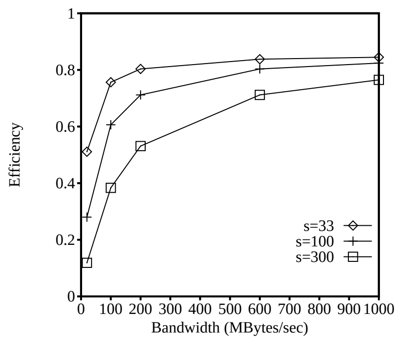

Figure 10: CG (p=64, n=1400*1400)

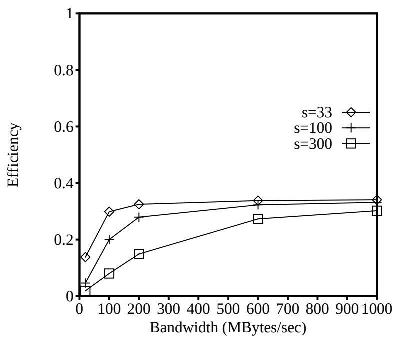

Figure 11: CHOLESKY (p=64, n=1806*1806)

Figures 7, 8, 9, 10 and 11 show the impact of link bandwidth on the efficiency of EP, IS, FFT, CG and CHOLESKY respectively, across different CPU clock speeds (s). 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 to the right uniformly across all applications.

Figures 12, 13, 14, and 15 show the impact of link bandwidth on the efficiency of EP, IS, FFT, and CG respectively, across different problem sizes. 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 and CG, 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.

Impact of Problem Size

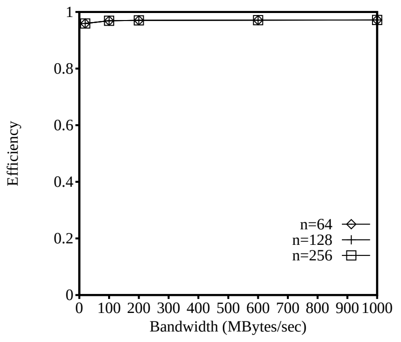

Figure 12: EP (p=64, s=33 MHz)

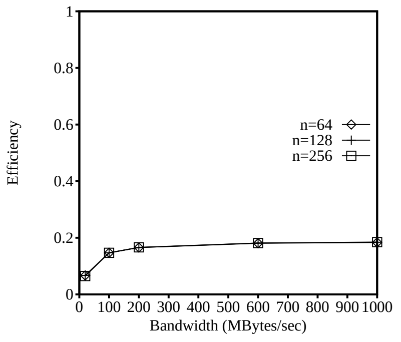

Figure 13: IS (p=64, s=33 MHz)

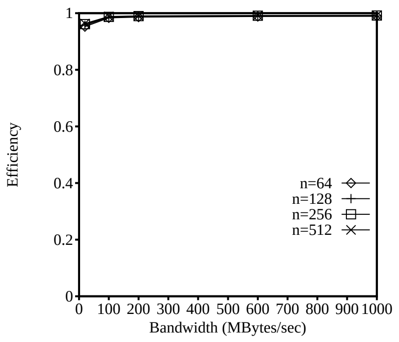

Figure 14: FFT (p=64, s=33 MHz)

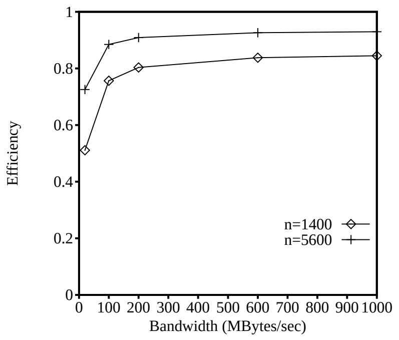

Figure 15: CG (p=64, s=33 MHz)

4.2 Quantifying Link Bandwidth Requirements

We analyze bandwidth requirements using the above simulation results in projecting requirements for large-scale parallel systems. We track the change in the knee with system parameters by quantifying 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 an ideal machine with zero network overhead). Ideally, one would like to operate as close to the knee as possible. But owing to either cost or technological constraints, one may be forced to operate at a lower bandwidth and it would be interesting to investigate if one may still obtain reasonable efficiencies under these constraints. Figure 16 shows the trade-off between the tolerable network overhead and the resulting bandwidth that needs to be sustained to maintain the overhead within the specified level. With the ability to tolerate a larger network overhead, the bandwidth requirements are expected to decrease as shown by the curve labeled “Actual” in Figure 16. To calculate the bandwidth requirement needed to limit the network overhead (both the latency and contention component) to a certain value, we simulate the applications and the network in their entirety over a range of link bandwidths. We plot the bandwidths and the resulting network overheads as shown by the curve labeled “Simulated” in Figure 16. We perform a linear interpolation between these data points to calculate the bandwidth (b_s) required to limit the network overhead to x% of the total execution time. This bandwidth would represent a good upper bound on the corresponding “Actual” bandwidth (b_a) required. Instead of presenting all the interpolated graphs, we simply tabulate the requirements for x = 10%, 30% and 50% in the following discussions. These requirements are expected

cause no change in communication in the above mentioned phases of IS, while the computation is expected to grow as $O(n)$. Hence, if we employ such a scaling strategy and increase the problem size linearly with the CPU clock speed, we may be able to limit the network overheads to within 30-50% for this application with existing technology.

	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 Fns.	$23.60p^{0.28} - 28.79$	$74.41p^{0.22} - 91.21$	$88.68p^{0.34} - 82.12$
p=1024	143.25	251.91	907.80

n=128K, s=33 MHz

Table 3: IS: Impact of Processors on Link Bandwidth (in MBytes/sec)

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

p=64, n=64K

Table 4: IS: Impact of CPU speed 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 5: IS: Impact of Problem Size on Link Bandwidth (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

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

Table 6: IS: Link Bandwidth Projections (in MBytes/sec)

FFT

The implementation of FFT has been optimized so that all the communication takes place in one phase where every processor communicates with every other processor, and the communication in this phase is skewed in order to reduce contention. 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 the number of processors. As the number of processors is increased, the contention encountered by each message in the network is expected to grow. However, due to the implementation strategy the bandwidth requirements of the network grow slowly with the number of processors as is shown in Table 4.2. These requirements can be satisfied even for faster processors (see Table 8). 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 9 confirms this observation. Hence, 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 10). The results shown in the above tables agree with theoretical results presented in [10] where the authors show that FFT is scalable on the cube topology and the achievable efficiency is only limited by the ratio of the CPU clock speed and the link bandwidth.

	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

Table 7: FFT: Impact of Processors on Link Bandwidth (in MBytes/sec)

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

p=64, n=64K

Table 8: FFT: Impact of CPU speed 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 9: FFT: Impact of Problem Size on Link Bandwidth (in MBytes/sec)

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

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

Table 10: FFT: Link Bandwidth Projections (in MBytes/sec)

CG

The main communication in CG occurs in the multiplication of a sparse matrix with a dense vector [26]. 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 at the same column position. As the number of processors is increased, the number of rows allocated to a processor decreases thus decreasing the computation that it performs. Increasing the number of processors 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 [4]. The results from our simulation are given in Table 11. 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. Increasing the clock speed has an almost linear impact on increasing bandwidth requirements as given in Table 12. As the problem size 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 13 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 [4] is 0.1 and we scale down the bandwidth requirements accordingly in Table 14 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 overheads to within 30%. But 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 may 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.

	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 11: CG: Impact of Processors on Link Bandwidth (in MBytes/sec)

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

p=64, n=1400*1400

Table 12: CG: Impact of CPU speed 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 13: CG: Impact of Problem Size on Link Bandwidth (in MBytes/sec)

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

p=1024, n=14000*14000

Table 14: CG: Link Bandwidth Projections (in MBytes/sec)

CHOLESKY

This application performs a Cholesky factorization of a sparse positive definite matrix [26]. Each processor while working on a column will need to access the non-zero elements in the same row position of other columns. Once a non-local element is fetched, the processor can reuse it for the next column that it has to process. The communication pattern in processing a column is similar to that of CG. The difference is that the allocation of columns to processors in CHOLESKY is done dynamically. As with CG, an increase in the number of processors is expected to decrease the computation to communication ratio as well as the probability of data reuse. Further, the network overheads for implementing dynamic scheduling are also expected to increase for larger systems. Table 15 reflects this trend, showing that bandwidth requirements for CHOLESKY grow modestly with increase in processors. Still, the requirements may be easily satisfied with existing technology for 1024 processors. Even with a 300 MHz clock, one may be able to limit network overheads to around 30% as shown in Table 16. Owing to resource constraints, we have not been able to simulate other problem sizes for CHOLESKY in this study. But an increase in problem size is expected to increase the the computation to communication ratio and has been experimentally verified on the KSR-1, which suggests that bandwidth requirements are expected to decrease with problem size. Hence, as the processors get faster, one may still be able to maintain network overheads at an acceptable level with existing technology if the problem size is increased correspondingly.

	50% ovhd.	30% ovhd.	10% ovhd.
p=4	5.62	11.98	76.56
p=8	6.86	13.11	78.32
p=16	7.77	14.48	80.83
p=32	8.91	16.02	84.12
p=64	10.49	17.48	87.35
B/w Fns.	$1.47p^{0.37} + 3.43$	$2.14p^{0.33} + 8.82$	$6.77p^{0.27} + 66.60$
p=1024	23.44	31.26	110.70

n=1806*1806, s=33 MHz

Table 15: CHOLESKY: Impact of Processors on Link Bandwidth (in MBytes/sec)

	50% ovhd.	30% ovhd.	10% ovhd.
s=33MHz	10.49	17.48	87.35
s=100MHz	16.56	60.13	278.92
s=300MHz	29.51	171.29	712.60

p=64, n=1806*1806

Table 16: CHOLESKY: Impact of CPU speed on Link Bandwidth (in MBytes/sec)

5 Discussion

In the previous section, we 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. Based on these results we also projected the requirements of large systems built with 1024 processors and CPU clock speeds upto 300 MHz. We observed that EP has negligible bandwidth requirements and FFT has moderate requirements that can be easily sustained. The network overheads for 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 provided the problem size is increased commensurately. The network overheads of IS are tolerable for slow processors, but the requirements become unmanageable as the clock speed increases. As we observed, the deficiency in this problem is in the way the problem is scaled (the number of buckets is scaled linearly with the size of the input list to be sorted). On the other hand, if the number of buckets is maintained constant, it may be possible to sustain bandwidth requirements by increasing the problem size linearly with the processing speed.

In [18], the authors show that the applications EP, IS, and CG scale well on a 32-node KSR-1. Although our results suggest that these applications may incur overheads affecting their scalability, this does not contradict the results presented in [18] since the implications of our study are for larger systems built with much faster processors.

All of the above link bandwidth 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 [11], cost and technology factors often play an important role in its physical realization. Agarwal [2] and Dally [8] 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 [15] and 3-dimensional [17, 3] 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. 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. Such a projection assumes that the communication in an application is devoid of any network locality and that each message crosses the bisection. But we know that applications normally tend to exploit network locality and the projection can thus become very pessimistic [1]. 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 every other processor, and thus only 50% of the messages cross the bisection. Similarly, instrumentation in our simulation showed that 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. But 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 method ignores the temporal aspect of communication. 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 would be interesting to conduct a detailed simulation for these network topologies to confirm these projections.

6 Concluding Remarks

In this study, we undertook the task of synthesizing the bandwidth requirements of five parallel applications. Such a study can help in making cost-performance trade-offs in designing and implementing networks for large scale multiprocessors. One way of conducting such a study would be to examine the applications statically, and develop simple analytical models to capture their communication requirements. But as we mentioned in Section 2, it is difficult to faithfully model all the attributes of communication by a simple static analysis for all applications. On the other hand, simulation can faithfully capture all the attributes of communication. We used an execution-driven simulator called SPASM for simulating the applications on an architectural platform with a binary hypercube topology. The link bandwidth of the simulated platform was varied and its impact on application performance was quantified. From these results, the link bandwidth requirements for limiting the network overheads to a specified tolerance level were identified. We also studied the impact of system parameters (number of processors, processing speed, problem size) on link bandwidth requirements. Using regression analysis and analytical techniques, we projected requirements for large scale parallel systems with 1024 processors and other network topologies. The results show that existing link bandwidth of 200-300 MBytes/sec available on machines like Intel Paragon [14] and Cray T3D [17] can sustain high speed applications with fairly low network overhead. For applications like EP and FFT, this overhead is negligible. For the other applications, this overhead can be limited to about 30% of the execution time provided the problem sizes are increased commensurate with the processor clock speed.

Using the technique outlined in this paper, it would 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 he needs to support 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.

References

[1] V. S. Adve and M. K. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):225–246, March 1994.

[2] A. Agarwal. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[3] R. Alverson et al. The Tera Computer System. In *Proceedings of the ACM 1990 International Conference on Supercomputing*, pages 1–6, Amsterdam, Netherlands, 1990.

[4] D. Bailey et al. The NAS Parallel Benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73, 1991.

[5] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl. PROTEUS : A high-performance parallel-architecture simulator. Technical Report MIT-LCS-TR-516, Massachusetts Institute of Technology, Cambridge, MA 02139, September 1991.

[6] 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*, pages 4–11, Santa Fe, NM, May 1988.

[7] 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*, pages 2–13, May 1993.

[8] W. J. Dally. Performance analysis of *k*-ary *n*-cube interconnection networks. *IEEE Transactions on Computer Systems*, 39(6):775–785, June 1990.

[9] 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*, pages II 99–107, 1991.

[10] A. Gupta and V. Kumar. The scalability of FFT on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, August 1993.

[11] 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*, 9(4):609–638, 1988.

[12] W. B. Ligon III and U. Ramachandran. Simulating interconnection networks in RAW. In *Proceedings of the Seventh International Parallel Processing Symposium*, April 1993.

[13] W. B. Ligon III and U. Ramachandran. Evaluating multigauge architectures for computer vision. *Journal of Parallel and Distributed Computing*, 21:323–333, June 1994.

[14] Intel Corporation, Oregon. *Paragon User's Guide*, 1993.

[15] D. Lenoski, J. Laudon, K. Gharachorloo, W-D Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam. The Stanford DASH multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[16] Microelectronics and Computer Technology Corporation, Austin, TX 78759. *CSIM User's Guide*, 1990.

[17] W. Oed. The Cray Research Massively Parallel Processor System Cray T3D, 1993.

[18] 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*, pages I–237–240, August 1993.

[19] 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*, pages 48–60, Santa Clara, CA, May 1993.

[20] 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*, pages 14–25, May 1993.

[21] SAS Institute Inc., Cary, NC 27512. *SAS/STAT User's Guide*, 1988.

[22] 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.

[23] J. P. Singh, W-D. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. Technical Report CSL-TR-91-469, Computer Systems Laboratory, Stanford University, 1991.

[24] A. Sivasubramaniam, U. Ramachandran, and H. Venkateswaran. A comparative evaluation of techniques for studying parallel system performance. Technical Report GIT-CC-94/38, College of Computing, Georgia Institute of Technology, September 1994.

[25] 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*, pages 171–180, May 1994.

[26] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran. On characterizing bandwidth requirements of parallel applications. Technical Report GIT-CC-94/31, College of Computing, Georgia Institute of Technology, July 1994.

[27] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran. A Simulation-based Scalability Study of Parallel Systems. *Journal of Parallel and Distributed Computing*, 22(3):411–426, September 1994.

[28] 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*, pages 54–63, January 1995.

[29] H. Sullivan and T. R. Bashkow. A large scale, homogenous, fully-distributed parallel machine. In *Proceedings of the 4th Annual Symposium on Computer Architecture*, pages 105–117, March 1977.

[30] D. A. Wood et al. Mechanisms for cooperative shared memory. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 156–167, May 1993.

[31] J. C. Wyllie. *The Complexity of Parallel Computations*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1979.