Architectural Requirements and Scalability of the NAS Parallel Benchmarks

Frederick C. Wong, Richard P. Martin, Remzi H. Arpaci-Dusseau and David E. Culler
Computer Science Division
University of California
Berkeley, CA 94720

January 14, 1998

Abstract

We present a study of the architectural requirements and scalability of the NAS Parallel Benchmarks. We find that the local processor and memory systems dominate both the overall performance and scalability of the NAS parallel benchmarks, as opposed to the communication architecture. Improvements in computational efficiency due to the parallel machines' increasing global cache size often offset the costs of communication and synchronization, hence yielding linear or even super-linear speedups. Communication performance is often much lower than that would be expected from micro-benchmarks, as a result of queuing effects in the both the software and hardware. We also find that network bisection bandwidth is the limiting factor for two of the benchmarks because of their all-to-all communication patterns.

1 Introduction

The NAS Parallel Benchmarks (NPB) are widely used to evaluate parallel machines. To date, every vendor of large parallel machines has presented NPB version 1.0 results [9]. With the recent convergence of parallel machines and the introduction of a standard programming model (MPI), the NAS group created version 2.2 of the benchmark suite [3]. In contrast to the vendor-specific implementations of version 1.0, NPB 2.2 presents a consistent, portable, and readily available workload to parallel machine designers, analogous to the SPEC benchmarks for single-processor machines. Although much has been written about the mathematical techniques in these codes, an understanding of their architectural requirements is at best incomplete.

The NAS parallel benchmarks use Constant Problem Size (CPS) scaling as the method of evaluating parallel machine performance [19]. Under CPS scaling, the problem size is fixed as processors are added. Figure 1 shows the speedup of three machines on the class A and class B problem sizes of the NPB 2.2. The

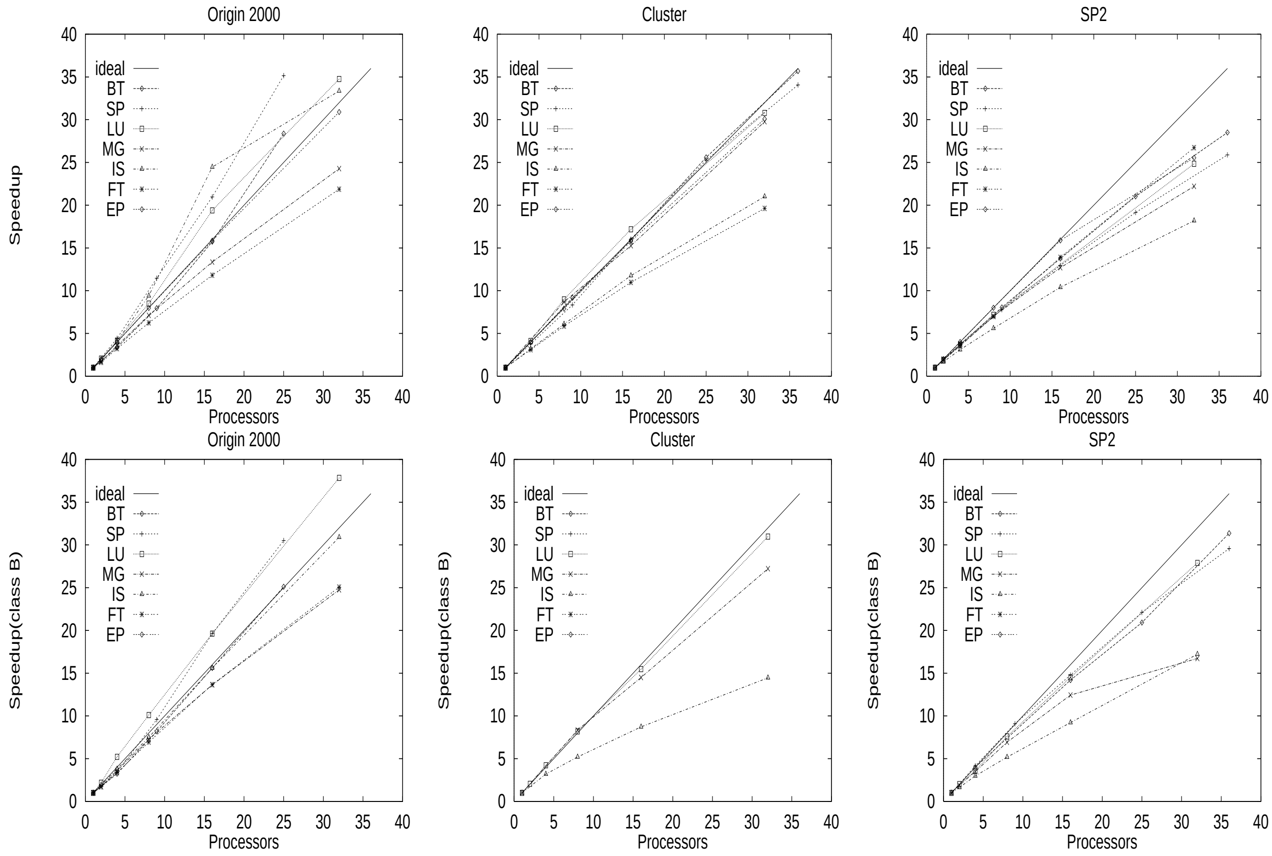


Figure 1: **Speedup on Three Platforms** *This figure shows the speedup of the NPB codes for the class A input (top) and class B (bottom) on three machines: The SGI Origin 2000, (left), a cluster of Sun Ultra-1 workstations (center) and the IBM SP2-66/Wide (right).*

results show that scaling characteristics across platforms are quite different. The SGI Origin has outstanding speedup, with many benchmarks achieving super-linear performance. The cluster of UltraSPARCs, the focus of our study, attains surprisingly good performance, with most benchmarks attaining near-ideal speedup. Finally, the speedup curves for the SP2 are all noticeably below ideal, though we should note that the absolute performance of the single-processor SP2 on the NAS benchmarks is comparable to the Origin, and better than the UltraSPARC. In this paper we quantify the characteristics of each benchmark that lead to such dramatically different performance results. More detail on the benchmarks can be found in Figure 2.

In order to establish a framework within which we can understand the performance of the NAS benchmarks, we employ a simple model of speedup. Scaling is a function of computation, communication and synchronization:

$$\text{Speedup}_p = \frac{T_1}{\max\left(T_{computation_i} + T_{communication_i} + T_{wait_i}\right)_{1 \leqslant i \leqslant p}}$$

Thus, speedup is determined by the time to perform the necessary computation on each of the p processors, as well as the time spent communicating and synchronizing. Because any time spent in synchronization and communication reduce speedup, we conclude from Figure 1 that the total computation time may *decrease* under scaling; though extra work may be required to parallelize a code, the efficiency of computation can increase – a result of the increase in total available cache. Linear, or even super-linear speedups are thus possible even with non-trivial communication and synchronization requirements. To compute this effect, we further define the time spent in computation as:

$$T_{computation_i} = \left(\left(\frac{Work_1}{p} \right) + ExtraWork_i \right) \times ComputeEfficiency_i$$

Both the SGI and UltraSPARC platforms experience an improvement in $ComputeEfficiency$; we seek to understand exactly when this benefit takes place, and how significant is its impact.

In this work, we present a quantitative study of computation, communication, and synchronization costs under CPS scaling. To approach this difficult problem, we employ a *hybrid* method, combining direct measurements from a real machine with parallel trace-driven simulations. Not only does this allow us to understand the performance characteristics of an actual platform, it shows us how different architectural parameters affect scaling.

We find that cache effects, and the resulting increase in processor efficiency, are often the dominate effect for many of the benchmarks. The effect is pronounced for both class A and B problem sizes The CPI improves dramatically under CPS scaling on the UltraSPARC platform, up to 50% for some benchmarks when scaling from 1 to 32 processors. We show this benefit occurs only on systems with large caches, usually 256 KB per processor or more in size. In systems with small caches, scaling can have the reverse effect of worsening memory-system behavior, due to the extra work involved in managing parallelism; hence linear speedup is not possible on such systems. We also find that communication and synchronization play important roles in benchmark scalability. Subtle effects such as queuing behind large messages lead to less than optimal communication efficiency. Finally, for those benchmarks that perform all-to-all communication, bisection bandwidth is critical to performance.

The three main contributions of this work are: (1) a characterization of the complex interactions of software and hardware effects on speedup, (2) a methodology for understanding speedup in the CPS domain, and (3) a quantitative analysis of the architectural requirements of the NAS Parallel Benchmark suite version 2.2, including the first detailed study of NAS working set behavior under scaling.

The rest of the paper is as follows. In Section 2, we describe our methodology. Section 3 presents the inherent characteristics of the NAS benchmarks, first in the single-processor case, and then as processors are

[1] Both $Work_1$ and $ExtraWork_i$ are expressed in instructions. $ComputeEfficiency_i$ is expressed in Cycles Per Instruction (CPI) times the machine's cycle time.

Code	Description	Class A Size	Class B Size
BT	Block Triangular Solver	64x64x64 doubles	102x102x102 doubles
SP	Scalar Pentadiagonal Solver	64x64x64 doubles	102x102x102 doubles
LU	Lower-Upper Symmetric Gauss-Seidel using SOR	64x64x64 doubles	102x102x102 doubles
MG	Multi-grid method to solve Poisson Equation	256x256x256 doubles	256x256x256 doubles
IS	Integer Sort	8 million integers	32 million integers
FT	Spectral Method FFT to solve Laplace Equation	256x256x128 doubles	512x256x256 doubles

Figure 2: **NPB Benchmark Overview.** *This table presents an overview of the NAS benchmarks. All benchmarks except* **IS** *are FP-intensive codes. The programs are derived from the field of computational fluid dynamics. Of the codes,* **BT**, **SP**, *and* **LU** *are considered pseudo-applications, whereas the rest are smaller, computational kernels. Problems come in three classes: A, B, and C. In this study, we concentrate on class A and B, as shown in the table. The* **EP** *program, not listed in the table, is an embarrassingly parallel random number generator; we do not include it in our study. None of the benchmarks performs a significant amount of I/O. In addition, both* **BT** *and* **SP** *require a square number of processors (e.g. 1,4,9,16,25,36), while the rest of the benchmarks required a power of two (e.g. 1,4,8,16,32) number of processors. More detail on all benchmarks can be found in [3].*

scaled. We proceed with an in-depth study of machine efficiency, showing how the changing characteristics of the benchmarks under scaling affects delivered performance in Section 4. Section 5 presents related work, and in Section 6, we conclude.

2 Methodology

Understanding the performance under scaling of large, parallel codes is a difficult problem [16]. Ideally, one would like to run the benchmarks on real machines, but this option precludes a detailed study of differing hardware characteristics such as cache size and other parameters. Simulations are problematic, because they limit the size of the problem and can potentially miss long-term effects. An entire run of the NPB for the just the class A size is over 1 trillion instructions. Observing effects with simulations alone would be prohibitive. For comparison, the Splash-2 suite, which can be run entirely under simulation, is about 10 billion instructions.

To remedy this situation, we apply a *hybrid* approach; where possible, we use an instrumented communication library in tandem with hardware counters to measure the execution characteristics of the benchmarks on a real machine. When necessary, we trace and simulate the benchmarks, which allows us to vary certain architectural parameters. Details for both of these methods are given below.

2.1 Direct Measurement

In this study, all real executions of the NAS benchmarks are performed on a cluster of 36 UltraSPARC Model 170 workstations (167 MHz clock, 128 MB memory, 16 KB L1 and 512 KB L2 cache). Each has a single Myricom network interface card attached to the S-Bus I/O bus. The machines are interconnected with ten 8-port Myrinet [4] switches in a fat-tree like topology.

All the NAS benchmarks communicate via MPI, version 1. Our implementation of MPI is based on the MPICH reference implementation [2]. All ADI calls are mapped to Active Messages [20], and the layer is highly tuned for performance. The resulting MPI implementation has a fixed set-up cost of 50 microseconds and a peak bandwidth of 30 MB/s.

To break down the performance of the benchmarks, we add instrumentation code to the MPI layer. At each MPI call, we record a desired set of statistics and at the end of execution, we write the results to disk. In all cases the instrumentation adds less than 5% to overall execution time and in most cases, less than 1%. In this manner, we gather information such as message sizes, destinations, and time stamps of when communication events occur.

To measure instruction count and CPI, we use the two available UltraSPARC performance counters. At the beginning of the run, we configure one counter to count cycles, and the other to count instructions. At the start and end of each MPI event, we record both these values; at the end of the run, we can de-construct not only the amount of time spent, but also the number of instructions executed inside and outside of MPI routines. All measurements run for the the full number of iterations specified by the NPB class A problem sizes.

2.2 Simulation

While the instrumented MPI layer can give us fairly accurate breakdowns of where time is spent in the code, it can not give us all the information we desire. The main pieces of missing data are the working sets for the benchmarks; because we are running on a fixed platform, with one cache size for all executions, we can not understand the true working set characteristics with direct execution.

To solve this problem, we employ the Shade simulation environment [6]. Shade provides a SPARC-based virtual machine to an application program, as well as certain kernel functionality required to run most programs. We use Shade in a novel manner, running independent instances of the simulator across the workstations of our cluster. Inside this "virtual cluster", communication between processes of a parallel program takes place on the real Myrinet network. We have written a Shade analyzer that outputs the data cache address trace of one process on the simulated machine.

For all traces, the benchmarks are run for a single time-step. Other experiments have revealed that behavior across processors and time steps is nearly identical for these benchmarks. After a trace is produced, we use the Dinero cache simulator [12] to simulate the desired cache configuration.

3 Algorithmic Characteristics

In this section, we describe the baseline characteristics and algorithmic scaling properties of the NAS benchmarks. We find that the NAS codes are well-tuned and mostly floating-point intensive. Scaling has the side-effect of increasing the amount of work (in instructions) performed by the benchmarks; one of the pro-

	FP Inst (Reported)	FP Inst (Measured)	Inst Count (Measured)	FP Inst (%)	ALU (%)	Loads (%)	Stores (%)	Branches (%)
BT	168.30	170.90	379.30	45.1	10.9	26.4	16.1	1.5
SP	85.00	80.10	234.20	34.2	22.7	30.2	10.8	2.1
LU	119.20	117.40	222.80	52.7	7.1	26.7	12.7	0.8
MG	3.89	4.17	11.83	36.1	20.3	31.8	8.9	2.9
IS	0.00	0.00	2.38	0.00	54.0	24.4	15.3	6.3
FT	7.14	8.56	19.70	43.5	16.1	18.7	18.3	3.4

Figure 3: **NPB Single-Processor Computational Characteristics.** *This table shows the single processor computational characteristics for the NPB codes on class A problem sizes. All instruction and floating point counts are given in billions. We present the total number floating point operations reported by the program, as well as the measured count. We also show the percentage of ALU operations, loads, stores, and branches.*

grams (**IS**) sees a 30% increase in aggregate instruction count at 32 processors. Communication volume for those programs performing nearest-neighbor exchanges (all but two) grows with the surface-to-volume ratio. The other two applications perform all-to-all exchanges, and the amount of communicated data remains constant under scaling. Given all of these factors, communication is still relatively coarse-grained; however, with scaling, the average message size may decrease by two orders of magnitude (*e.g.,* from 10 MB per message to 100 KB).

3.1 Single Node Characteristics

Figure 3 shows single-processor program characteristics for the NAS benchmarks on our cluster platform for the Class A size of the benchmark. Qualitatively, with the exception of **IS** , the benchmarks are overwhelmingly floating point intensive. The benchmarks not only compute, but also move a substantial amount of data through the CPU, as evidenced by the high frequency of load/stores. For reference, a single run of the class A benchmarks streams well over 1 terabyte of data through the CPU. The low frequency of branches suggests that the codes consist of large basic blocks. The low ratio of integer to FP operations suggests that the overhead of integer work per-FLOP is low.

3.2 Computational Scaling

One key factor in speedup is the total amount of work across all processors that is performed, that is, $Work_1+ ExtraWork$ in our model. Figure 4 plots the measured percent increase in total instruction count, (i.e. $100\times\frac{ExtraWork}{Work_1}$) as we increase the number of processors. The results show that $ExtraWork$ grows slowly for most codes: benchmarks execute 5%-10% more instructions over the entire run of the program. One exception is **IS**, which has a large initial penalty of 35% more instructions for the parallel version of the program then diminishes. The growth rate of **SP** is also substantial, executing 20% more instructions at 36 processors. Most of the other programs reveal a slow growth in instruction count as processors scale, with less than a 5% increase at 32 to 36 processors. The one exception is **FT**, which exhibits a fixed cost of parallelism in a 5% increase in total instruction count. The data implies that there is an instruction cost to

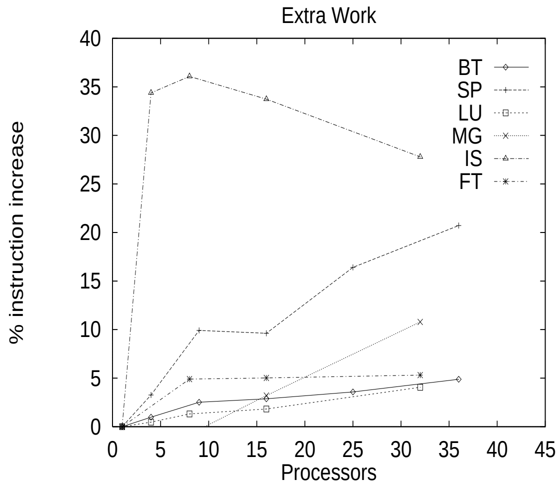


Figure 4: **Extra Work.** *The graph shows the percentage of extra instructions executed across all processors under scaling, as compared to the single processor case.*

parallelism in the NAS benchmarks. Without any compensating factor, this cost will have a negative impact on speedup.

3.3 Communication Scaling

Figure 5 shows the baseline communication characteristics of the NAS benchmarks on 4 processors. All other communication scaling is relative to this base case. The table shows that in the 4 processors case, the benchmarks send very few messages. The size of the messages are quite large; some of the benchmarks send messages that are megabytes in size. Both **BT** and **SP** use the `Isend` primitive, which is a non-blocking send, whereas **LU** and **MG** use the traditional blocking `send`. The number of collective communication operations is also quite low. Surprisingly, no benchmark uses barriers in the benchmarked portion of the code. Most benchmarks use reduce operations, but the reduce operations do not contribute to overall performance significantly. Only **FT** and **IS** communicate primarily with collective communication; both do a series of all-to-all exchanges.

3.3.1 Communication Balance

Figure 6 shows the number of bytes sent and received by each processor throughout the duration of one iteration on 32 or 36 processors. We define communication as balanced when each processor sends and receives the same amount of data. The figure shows that communication patterns of the NAS benchmarks are mostly balanced at the algorithmic level. Assuming that communication is well scheduled, there are few obvious points of contention, or "hot-spots". However, the probabilistic bucket algorithm used in **IS**

	4-Processor Data					
	Point-to-Point		**Collective**			**Total Volume**
	Isend	**Send**	**Alltoall**	**AllReduce**	**Barrier**	**(MB)**
BT	9600	0	0	0	0	1072.9
SP	19200	0	0	0	0	1876.2
LU	0	126000	0	2	0	459.1
MG *	0	4464	0	4	0	104.4
IS	0	0	20	10	0	320.2
FT	0	0	7	(Reduce) 6	0	896.0

Figure 5: **NPB Communication Baseline Characteristics.** *This table shows the baseline communication characteristics for the NPB codes on class A problem sizes. The table shows the number of messages sent breakdown by type, and the total number of bytes sent. FT uses Reduce instead of AllReduce.* * *The results for* **MG** *are for the 8-processor case.*

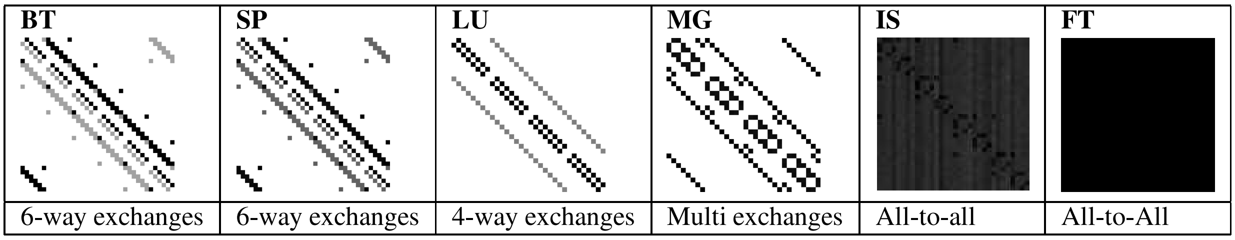
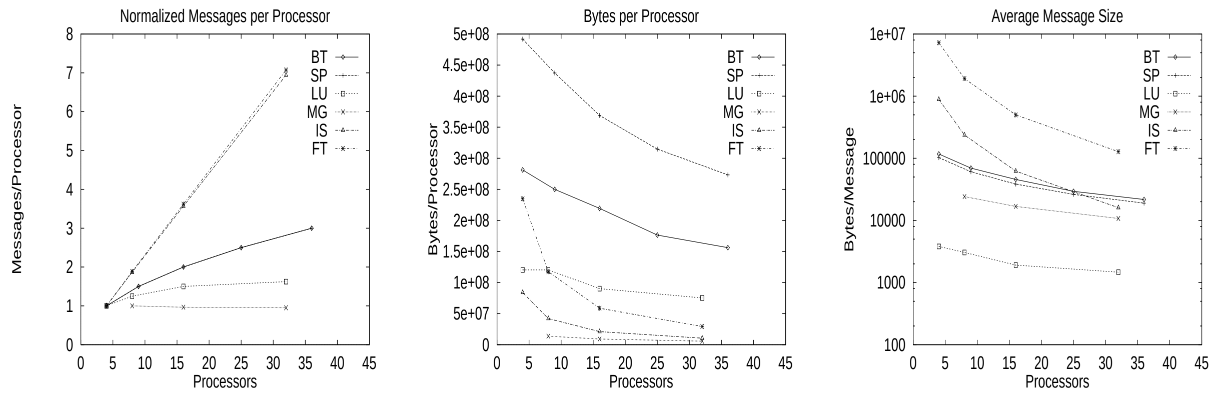


Figure 6: **Communication Balance** *This figure presents the communication balance between each of the 32 or 36 processors. The grey scale for each pixel represents a byte count. Each application is individually scaled from white, representing zero bytes, to black, representing the maximum byte count over all processors. The y-coordinate tracks the message sender and the x-coordinate tracks the receiver. The darkness of cell (i, j) indicates the fraction of bytes sent from processor i to processor j.*

results in somewhat unbalanced communication. The reduction pattern creates a few dark spots along the diagonal, but the darker spots along the other corners indicate that a few processors receive more data than the others, up to 20% in some cases.

Several communication patterns are visible in Figure 6. **LU** has the simplest pattern with a 4-way neighbor exchange. The multi-partition scheme used in **BT** and **SP** results in a six-way exchange because each processor owns 'cubes' of the data-set [10]. Each processor exchanges boundary conditions along the six faces of its cubes. **MG** presents a more complex multi-way exchange, but it is clear from the figure that the communication is also balanced. Finally, the even coloring of **FT** and **IS** correspond to all-to-all exchanges.

The communication patterns of **FT** and **IS** stress the bisection bandwidth of a machine. The other benchmarks, because of the locality of their communication, may perform adequately on machines with reduced bisection bandwidth.



(a) (b) (c)

Figure 7: **Message Scaling.** *This figure shows how the number of messages sent normalized to the 4 processor case (left), number of bytes sent (center) and average size of a message (right) scale as a function of processors.*

3.3.2 Communication Volume and Distribution

In addition to the communication pattern, changes in communication volume and message count affect performance under scaling. Figure 7(a) plots the change in total message count as a function of the number of processors, normalized to the message count on 4 processors. The figure shows that the number of messages for all programs increases as the machine scales. Figure 7(b) shows the byte count per processor as a function of scaling. The byte count per processor for all benchmarks decreases. Finally, Figure 7(c) shows the resulting average size message per processor.

Within the realm of interest, there is an order of magnitude difference in the average size of a message. Interestingly, the smallest messages are still on the order of 1000 bytes, which is a substantially larger grain than found in many other parallel benchmarks [13, 21].

Because of their all-to-all pattern, for both **IS** and **FT** , the total number of messages increases as the square of the number of processors, and thus the per-processor message count is linear in the number of processors. However, the total byte volume remains constant for both of these codes and therefore the bytes per processor decrease by $O(\frac{1}{p})$ and the message size decreases as a function of $O(\frac{1}{p^2})$. Although for the range of processors studied the absolute number of messages remains relatively small, the squaring of the message count and resulting decrease in message size has important implications for machine designers. For machines of up to 100's of processors, efficient transfer mechanisms must exist for messages ranging from a few hundred bytes to megabytes in size.

For **LU**, **BT**, **SP**, and **MG**, the amount of total communication follows surface to volume ratio as we scale the number of processors. For example, in both **BT** and **SP** , each processor owns $\sqrt{p}$ cubes of space in the physical domain. Since the number of messages is constant for each face of the cube, the number of messages per processor grows with $O(\sqrt{p})$, and thus the total number of messages increases as $O(p\sqrt{p})$.

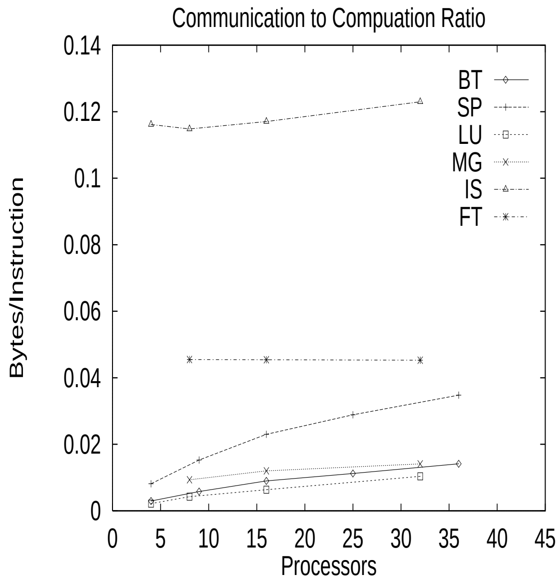


Figure 8: **Communication to Computation Ratio.** *This figure shows the number of bytes sent per instruction executed.*

The total volume of communication grows with $O(\sqrt{p})$, and thus the volume per-processor decreases as $O(\sqrt{p})$. For the range of processors of interest, the scaling of communication along these lines does not unduly limit speedup. The spatial decomposition keeps communication in the nearest-neighbor regime for these benchmarks as well.

3.4 Summary

Figure 8 shows how the communication to computation ratio scales with the number of processors. On average, the communication to computation ratio rises slowly. Indeed, for **FT** and **IS** , the ratio is constant. However, the there is an order-of-magnitude variation in communication to computation ratios across the benchmarks. The communication to computation ration, while non-trivial, is still on the lower end of those reported in the SPLASH-2 suite [21].

Overall, we have seen that most codes perform more work as processors increase, usually a 5% to 20% increase. Further, all programs perform a substantial amount of communication, and all but two show an increase in the total amount of data communicated under scaling. Both of these factors have a negative impact on scaling, and without any counter-balancing effect, would guarantee less than perfect speedup. Recall from Figure 1 that on both the cluster and the SGI Origin one or more benchmarks showed super-linear speedup. To understand this effect, we proceed to architectural efficiency.

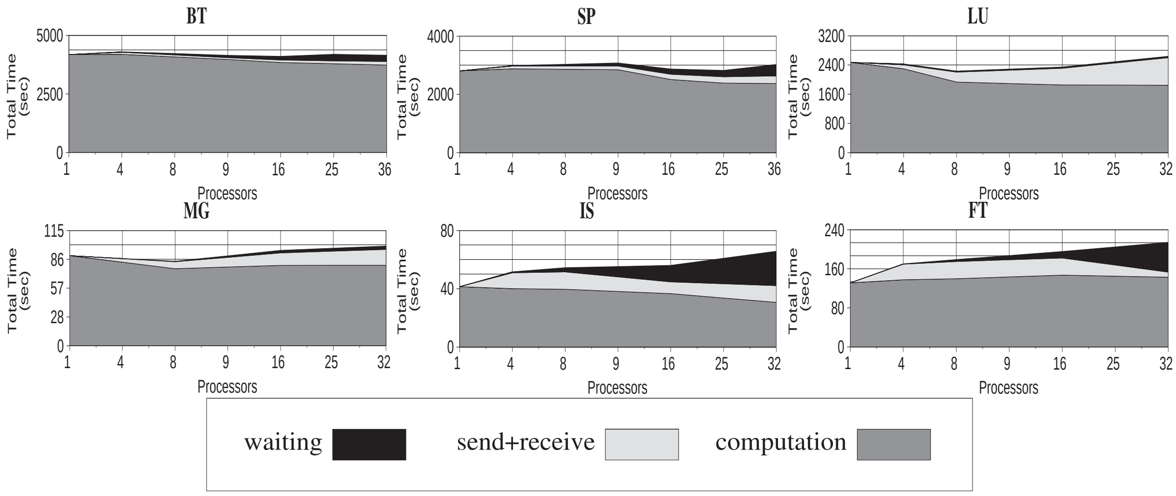


Figure 9: **Execution Time Summary.** *This figure shows the total amount of time spent in different tasks by the programs. The time spend in computation, communication, and waiting are shown as a function of the number of processors, summed across all processors on the cluster platform.*

4 Architectural Efficiency

This section examines the concrete interaction of the algorithmic factors previously described on a specific platform. Figure 9 shows the amount of time spent in computation, communication, and waiting as measured on our cluster platform. The times shown are summed across all processors; the total area in each graph thus measures the total amount of work and communication done by the programs. Assuming that the programs are well balanced, a flat line represents a linear speedup, a decreasing line is a super-linear speedup, and an increasing line is a sub-linear speedup. The figure shows that computation dominates the run time for all the applications. A closer inspection of Figure 9 and Figure 1 show a number of unusual deviations from simplistic models of speedup, however.

For all the applications but **FT** ,the amount of time spent in computation actually decreases under scaling. The occurs despite modest increases in the $ExtraWork$ involved in parallelism. Figure 9 shows that in the linear-speedup regime, the remaining time spent is spent in communication and synchronization.

An interesting deviation from the algorithmic level can be seen examining **IS** and **FT**. Both these benchmarks have a constant communication to computation ratio at the algorithmic level. This suggests that communication costs should be relatively constant with scaling; the message sizes remain relatively large for most machines and no extra bytes are sent. However, the communication performance of both of these programs degrades significantly with scale. We detail communication performance in Section 4.3.

	Single-Processor Performance		
	Origin	**UltraCluster**	**SP2**
BT	2488.3 s	4177.8 s	2574.2 s
SP	1652.2 s	2806.7 s	1817.2 s
LU	1373.2 s	2469.3 s	1870.7 s
MG	53.4 s	89.9 s	53.3 s
IS	36.7 s	41.4 s	29.1 s
FT	133.4 s	131.1 s	*139.0 s
SPECfp95	19.0	9.4	9.69
SPECint95	9.48	5.56	3.19
Triad	317.0 MB/s	253.8 MB/s	654.5 MB/s

	UltraSPARC Specific Data		
	MFLOPS (per proc)	**Instructions** (Billions)	**CPI**
BT	40.28	379.3	1.787
SP	30.29	234.2	1.560
LU	48.31	222.8	1.673
MG	43.30	11.83	1.213
IS	+2.03	2.38	1.431
FT	54.42	19.70	0.996

Figure 10: **Single-Processor Performance.** *The table on the left shows the single-processor computational performance for the NPB codes on class A problem sizes, the SPEC95 benchmarks, and the Triad benchmark of STREAM. Results are presented as follows: NAS baseline run time in seconds, SPEC95 results as a SPEC rating, and Triad results in MB/s. The table on the right shows achieved performance on the UltraSPARC-I, in terms of MFLOPS, and memory system efficiency in CPI.* * **FT** *results on the SP2 are reported as twice the 2-processor case (due to unavailability of the single-processor data).* + *Reported performance for* **IS** *is MOPS instead of MFLOPS.*

4.1 Single Processor Performance

Before scrutinizing the efficiency of various aspects of the cluster architecture, we present the raw performance on a single processor. Figure 10 shows the run time of the single node benchmarks, as well as the achieved MFLOPS and CPI for the UltraSPARC cluster. From the table, we see that the performance of both the Origin and the SP2 are roughly equivalent in most cases about twice as good as the raw performance of the UltraSPARC.

Figure 10 also presents the results of three common benchmarks of single-processor processor performance: SPECfp95, SPECint95, and the triad benchmark from STREAM [14]. From the SPECfp results, one might be tempted to conclude that the Origin processor will perform twice as well the SP2 processor on the NAS benchmarks. However, we can see that this is not the case. The triad benchmark gives some indication as to why this might be: the SP2 is optimized for high memory bandwidth. In the single-processor regime, most of the NAS problem sizes do not fit in cache, and therefore, memory bandwidth plays an important role to performance.

The right-most table in the figure presents measured performance and computational efficiency of the UltraSPARC platform; most CPI ratings are in the range of 1.4 to 1.8. Given that the machine can execute two floating point instructions every cycle, we conclude that the processor is operating well below peak performance.

4.2 Computational Efficiency

In this section, we examine the changes in computational characteristics under scaling for the cluster platform. Because we see super-linear speedup on the cluster (and on the SGI as well), we conclude that com-

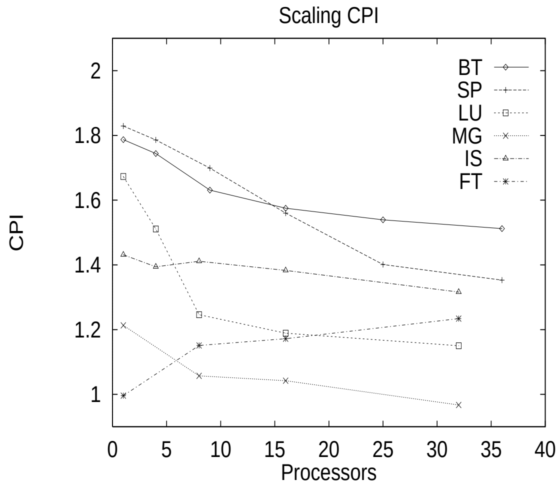


Figure 11: **Computational Efficiency.** *The graph shows the measured CPI under scaling.*

putational efficiency *must* improve under scaling; it is the only factor that can counteract the increase in instructions and the cost of communication. In this section, we quantify when and where these computational "boosts" occur, essentially where the $ComputeEfficiency$ term in the model improves.

We find that improvements in computational efficiency are a direct result of increased memory-system efficiency; because we are scaling a fixed problem, each processor operates on a smaller data set as processors increase. Therefore, scaling can cause a working-set threshold to be crossed, which lowers the cache miss rate, and improves processor utilization. We also find that machines with smaller per-processor caches (*i.e.,* older machines) would *not* realize these benefits; for the NAS benchmarks on the studied range of processors, the benefits of scaling occur with caches that are usually 256 KB or larger.

Figure 11 plots the effects of scaling on computational efficiency, by showing the CPI. From the graph, we can see that nearly all benchmarks see a significant improvement in CPI. In addition, there are several "knees" in the curve, were working-set are crossed. **LU** shows the most dramatic improvement, with a 35% decrease in CPI at 8 processors, and a 45% overall decrease at 32 nodes. The only exception to this is **FT**, which shows a 20% increase in CPI as processors are scaled.

4.2.1 Working Sets

To gain better insight into the memory access characteristics of the benchmarks under scaling, we simulate different cache sizes to discern the true working sets of the codes. Whereas many studies, such as those of SPLASH-2 [21], have shown the "knees" of the working set *on a given number of processors*, we also show how the working sets change as processors are scaled. This important effect is a fundamental property of CPS scaling; as the fixed problem size is spread across more processors, the load placed on each cache

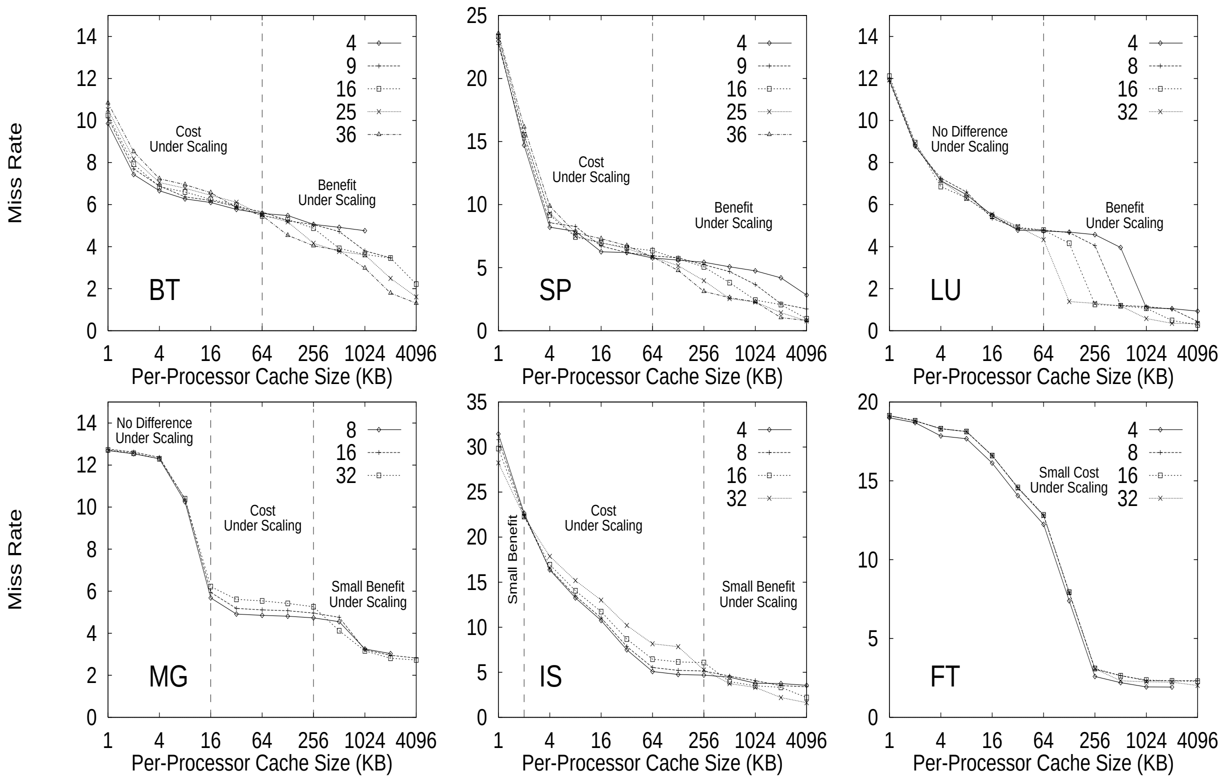


Figure 12: **Working Sets.** *These graphs show the working sets for the NAS benchmarks. Along the x-axis, we increase the size of the simulated cache. The y-axis plots miss rate. Each line shows the miss rate for a different machine size while changing the per-processor cache size. The benefit seen under scaling is the vertical distance between lines for a fixed size cache (the difference in miss rate). For all experiments, the cache parameters are: fully associative with LRU replacement, 64-byte blocks, write back, and write allocate.*

changes. Concurrently, the *global* amount of cache increases.

Figure 12 shows the data cache miss rate as a function of cache size for the NPB for cache sizes between 1 KB and 4 MB, on each of the machine sizes. The results show distinct differences in the cache behavior of the programs as the number of processors increases.

To demonstrate the effect of scaling on miss rates, let us concentrate on **LU**, where the effect is quite pronounced. If we examine the miss rate curve on the 4-processor machine, we see a gradual decrease in miss rate from 12% to roughly 4% when increasing per-processor cache size from 1 KB to 512 KB. At 1024 KB, a sharp drop occurs, where an important working set threshold has been crossed. The total amount of cache in the 4-processor system, or "global cache size", at that point is $4 \cdot 1024$ KB, or 4 MB. If we now focus our attention on the 32-processor system, we can see that the working set drop occurs at 128 KB of per-processor cache, or also at 4 MB of global cache. Thus, when the total amount of cache across processors reaches the 4 MB threshold in **LU**, a noticeable drop in miss rate will occur.

A real machine, however, has a fixed size cache. For example, when examining **LU**, a machine with 256 KB of per-processor cache can expect to see the cache miss rate drop from 4% to 1.5%, when scaling from 8 to 16 processors, almost a factor of three improvement. It is in scenarios such as this one where scaling increases memory system efficiency, potentially leading to super-linear speedup curves. In the graphs, these regions of potential benefit are labeled "Benefit Under Scaling."

For the rest of the benchmarks, we see a number of interesting effects. In **SP**, **BT**, and **MG**, machines with small caches would see only *increasing* miss rates under scaling, in the range of processors studied. Remember that for all of the benchmarks, scaling had the property of increasing the amount of work (in instructions) as processors are added. Therefore, the load on the memory system is likely to be higher. If no global cache size threshold is crossed, the end result is often a higher miss rate during scaling. Regions where scaling leads to higher miss rates are labeled in the graphs as "Cost Under Scaling."

Finally, we can see that **FT** shows no benefit from scaling. Though the important footprint of the program fits into a 256 KB cache, scaling does not improve the miss rate. In fact, it makes it slightly worse, due to the small amount of extra instructions performed in the parallel version.

4.2.2 Analysis

Now that we have seen how the working sets change under scaling, we can better understand the scaling performance of the cluster of UltraSPARCs, as well as both the Origin and SP2 systems. Figure 13 shows the cache parameters of the processors found in each system. The UltraSPARC I external cache is 512 KB; though it is direct-mapped, we can ascertain from the working set graphs that this is large enough to lead to a substantial improvement in miss rate. Figure 11 confirms this with actual measurements.

We saw in Figure 10 that the SP2 investment in memory bandwidth paid dividends for single-processor performance. However, the SP2 has a smaller external cache at 256 KB, and therefore probably misses out on some of the benefit that could have been available with a larger cache. It is therefore not surprising that none of the SP2 benchmarks achieve super-linear speedup. Finally, each processor in the Origin has a large 4 MB of cache per processor, and in this range, computational efficiency improves under scaling. In fact, the relative drop in miss rate may be more substantial than in the case of the UltraSPARC. The result is excellent (and often super-linear) speedup for the SGI machine.

4.3 Communication Efficiency

In this section, we explore the time spent in communication, $T_{communicate}$ and the time spent waiting, T_{wait}. We find that unlike computational efficiency, defining communication efficiency under a message passing model requires care.

Measured efficiency is surprising low given the well-structured nature of the NPB. However, efficiency stays relatively constant under scaling. We also find that complex and subtle effects in the network, in-

	MIPS R10000	UltraSPARC-I	Power2/590
Date	Nov 96	Nov 95	May 94
MHz	195	166.7	66.5
L1 size	32KB(i)+32KB(d)	16KB(i)i+16KB(d)	32KB(i)
L2 size	4MB(u)	512KB(u)	256KB(d)

Figure 13: **Machine Parameters.** *This table shows a comparison of three processors, the MIPS R10000, which forms the core of the SGI Origin 2000, a Sun UltraSPARC-I, the cluster processor, and the IBM Power2/590, the heart of the SP2 system.*

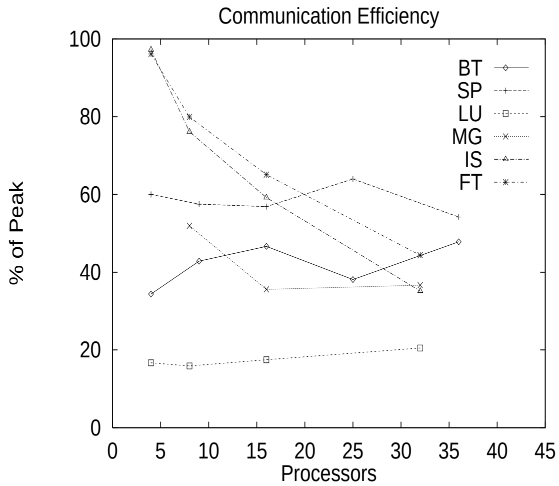


Figure 14: **Communication Efficiency.** *This figure shows the percentage of micro-benchmark peak bandwidth delivered by the MPI layer vs. the total time spent in sends and waiting for each application as a function of the number of processors.*

cluding queuing delays, cause severe performance degradations. Bisection bandwidth plays a strong role in determining scaling performance for **IS** and **FT**.

4.3.1 Delivered Efficiency

Message layer performance is often quoted as the delivered performance under a few communication micro-benchmarks [7, 11]. However, micro-benchmarks measure communication performance under ideal conditions, and we wish to quantify the communication performance under a real workload. Unfortunately, quantifying communication efficiency is difficult. Under a message-passing model, both communication and synchronization costs are intertwined at many levels. For example, when a program calls MPI `Isend`, the Active Message layer underneath MPI could store all communication into local buffers. The program meanwhile, would be blocked in the succeeding `wait`. To the program, the `Isend` call looks like it runs at the memory copy speed, and the synchronization cost appears large. However, the time spent in the `wait`

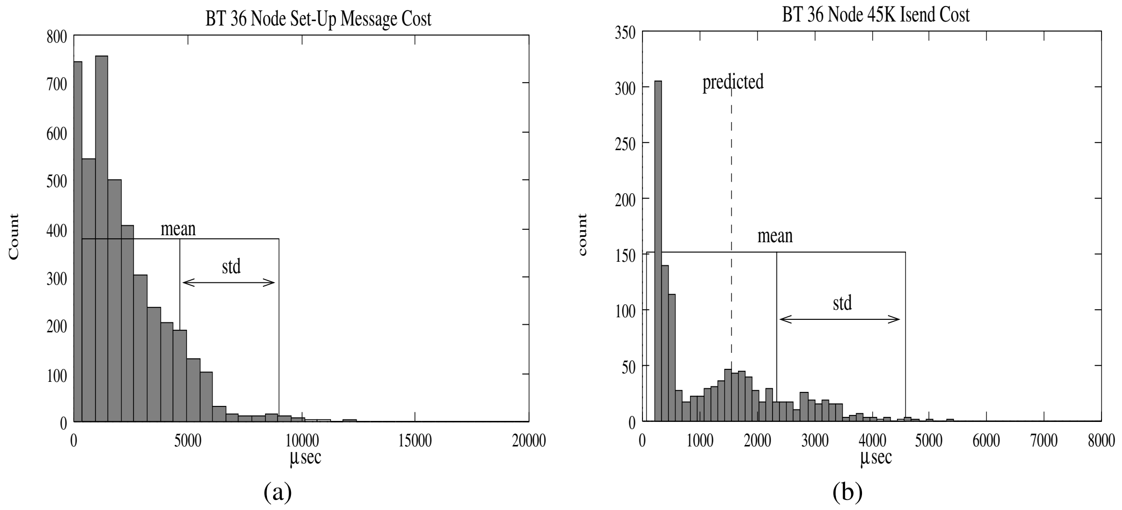


Figure 15: **Changes in Message Cost Using Different Protocols.** *These figures show the change in message costs with different MPI layers. Figure (a) plots the histogram of observed round trip times for the initial MPI layer's set-up message for 1 node of the* **BT** *benchmark on 36 processors. Figure (b) plots time taken in the one-way protocol MPI* `Isend` *for a 45K message size. The mean and standard deviation are shown for both graphs. The tails of the distributions (60,000 μsecs and 20,000 μsecs, respectfully) have been omitted for clarity.*

call is a function wire delay, rather than a true synchronization cost.

Given that the NPB are balanced algorithmically, this observation leads us to define communication efficiency as the *sum* of the time spent in MPI `Isend` plus `wait`Ġiven our zero algorithmic waiting-time cost assumption, our definition should reasonably approximate the wire cost.

Figure 14 plots communication efficiency for the NAS programs on the cluster. For all benchmarks except **IS** and **FT**, efficiency stays relatively constant during scaling. There is a wide range of efficiencies, from a low of 20% (**MG**), to around 60% (**SP**). The low efficiencies are surprising given the care the NPB 2.2 codes take to reduce communication costs by sending a limited number of high volume messages.

The inefficiencies can partly be explained by queuing effects. The additional inefficiencies for **IS** and **FT** are the result of bisection bandwidth limitations of the cluster. The efficiency of **IS** is further reduced (see Figure 9) by its algorithmic communication imbalance as seen in Figure 6. The next two sections describe these effects in more detail.

4.3.2 Queuing Delays

Our first attempt at an MPI layer resulted in efficiencies which were much lower than seen in Figure 14. However, micro-benchmarks indicated that the layer should have good performance. Further investigation revealed that the source of the problem was the internal protocol of the MPI layer.

Due to the availability of low latency communication, our older implementation MPI used a round-trip communication to initialize data structures at the receiver for message transfers larger than 8 KB. Inserting the round-trip protocol message simplified both the receive code and the message format. The design of the MPI layer assumed that the short Round Trip Time (RTT) could be amortized by the impending bulk transfer. Under micro-benchmarking conditions, this design does deliver good performance.

Performance will degrade quickly if the initial set-up message suffers from a long delay. We thus decided to test our assumption that actual RTT matched the micro-benchmarked time. Figure 15(a) plots the resulting histogram of measured round-trip times for the protocol message during a run of the **BT** benchmark on 36 nodes. Although in micro-benchmark tests the RTT is about 50 μsec, the actual mean cost is 5 ms! The variance is also quite large indicating that prediction of round-trip times would be difficult. Because of the large average message size of the NPB, the protocol messages often experienced long queuing delays.

The two orders of magnitude difference in micro-benchmark vs. actual RTT caused us to abandon using round-trip set up messages. We re-architected the MPI layer so it did not require the use of a round-trip set-up message. Instead, when performing a large data transfer, each of the 8 KB blocks contains enough information to identify itself to a reconstruction routine. At the cost of code complexity to re-sequence out-of-order messages, communication performance improved by over 100% on some of the NPB codes.

Examining the new MPI layer, we find that the mean performance is much closer to the micro-benchmark time, but substantial inefficiencies remain. Figure 15(b) plots the distribution of time in an `Isend` call for the new MPI layer for 45K messages. The distribution is very spread out. Even though most of messages fit into local buffers and run at the copy speed, the mean time of an `Isend` call is still significantly higher than the micro-benchmark prediction of 1560 μsec.

4.3.3 Bisection Bandwidth

In addition to queuing delays, bisection bandwidth constraints can precipitate another form of contention. Figure 16 shows how the per-processor delivered bisection bandwidth changes as the number of processors increases. Up to 25 processors, the delivered bandwidth per processor stays constant at 20 MB/s (or 40 MB/s for both send and receive). After 25 processors bandwidth rapidly trails off to 10 MB/s.

This behavior, though an artifact of our current cluster environment, has a noticeable impact on applications that perform all-to-all communication. If we glance back at Figure 9, we can see that wait time in both **FT** and **IS** increases noticeably when crossing the 25-processor threshold, a direct result of poor bisection bandwidth. The other benchmarks, which perform only localized, nearest-neighbor style communication, are unaffected.

4.3.4 Analysis

The observed queuing delays in the network has serious implications for the design of message passing layers. Even with very low latency networks, unless the entire path for short messages is separate from

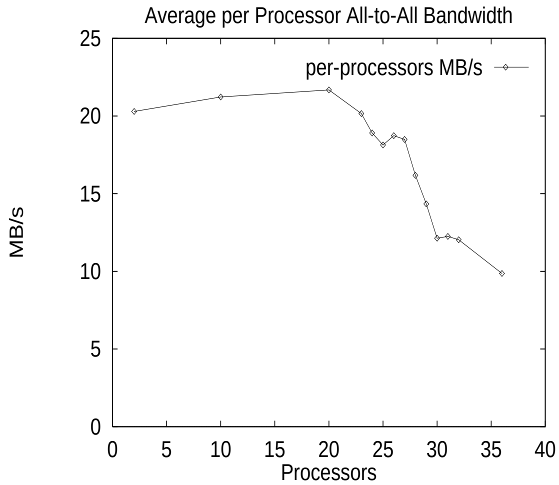


Figure 16: **Bisection Bandwidth on the Cluster.** *This figure shows how the average per-processor delivered send bandwidth changes for an MPI* `all-to-all` *call as a function of processors.*

long messages, using a round-trip protocol can result in poor performance. Although the average load on the network is low, the large messages sizes and bursty behavior of the NPB result in even small messages experiencing long delays. Better communication scheduling, perhaps with techniques found in [5], may be required to obtain higher efficiency.

5 Related Work

The performance of the NPB 2.1 codes is reported in [18] for several different platforms. Although the qualitative conclusions they reach are similar to ours, little quantitative data on the internal structure of the NPB is presented. As in this study, the NAS report attributes the SP2's good overall performance to high memory bandwidth and its poor scalability to lack of network bandwidth. Although two of the platforms in their study exhibited super-linear speedup (the SGI PowerChallenge and the Cray T3D), they did not investigate or explain this effect.

Perhaps the closest study to our own is the SPLASH-2 work [21]. The NAS suite, however, differs significantly from the SPLASH-2 benchmarks in two important ways. First, the NAS codes use explicit message passing, whereas SPLASH codes are for shared-memory machines. More importantly, we see that all NAS programs are highly structured: communication is infrequent and consists of large transfers. The communication to computation ratio is roughly equivalent to the less frequently communicating SPLASH-2 codes.

A study of various computational kernels' scaling behavior is presented in [17]. However, the study focuses on memory-constrained and time-constrained scaling, rather than the problem-size constrained scaling. Interestingly, the study found that much smaller grain sizes were acceptable up to a large (1000) number of nodes. This is direct opposition to the results we present here; under problem-constrained scaling, much larger grain sizes are necessary to amortize the overheads of parallelism.

Cypher [8] describes the characteristics of a set of substantial message passing applications. Qualitatively, the applications are similar to NAS in that they were developed in the context of heavy-weight message-passing libraries, and were highly floating-point intensive. However, it is unclear from the paper the nature of the computational performance of these applications, especially the interactions with the memory hierarchy. Also, the range of processor studies, into the 100's of nodes, and time-scales (hours) of the applications are quite different from those of the NPB presented here. The applications in [8] have a much wider range of communication to computation ratios than we found in the NPB.

A study of the NPB 1.0 suite for the Convex SPP [1] shows similar computational characteristics of those studied here. The Convex suite had a similar high percentage of load/store instructions, and an even lower frequency of branches. A direct comparison of communication costs is difficult because the codes were written using shared memory.

6 Conclusions

We have presented a study of Constant Problem Size (CPS) scaling of message passing parallel programs. Our study focuses on version 2.2 of the NAS Parallel Benchmarks, an important suite of scientific codes. We conclude by touching on the architectural requirements of these benchmarks, and by summarizing what we have learned and what remains as open research.

Architectural Requirements Our primary observation of the NPB shows that for both the class A size as well as the class B size, the benchmarks are highly computationally intensive. They also have low communication to computation rations. This leads to the direct conclusion that the *sequential node architecture*, that is the CPU, caches and local memory system, are the primary determinants of NPB performance.

In addition, we have have found the somewhat surprizing result that the primary factor affecting the scalability of the NPB is not the parallel architecture, i.e. the network fabric and interface into the node. Rather, the size of the cache per-node is the primary architectural feature which determines scalability.

These "cache-effects" are due to a significant shift in working-set characteristics under scaling. Even for as memory-intensive codes as the NPB, a machine can exploit these effects for the by having reasonably sized L2 caches. For the problem sizes we studied, 512 KB or larger captures a significant portion of the potential. Memory bandwidth, while useful when the code is operating out of cache, cannot provide a performance boost into the super-linear speedup regime in the parallel case. A comparison of the cluster and SP2 is instructive; even with it's outstanding memory bandwidth, the SP2 performs about on par with the Cluster at a 32 processor size.

In the communication regime, we have seen that messages are relatively coarse-grained across all benchmarks, with average sizes ranging from 1 KB to over 1 MB. However, we have seen a 100-fold decrease in average message size when scaling from 1 to 32 processors, implying that scaling beyond 32 processors will soon lead to relatively fine-grained communication. While perhaps tolerable on Origin-like systems, this will severely degrade performance on less tightly-coupled platforms such as the Cluster and SP2.

Even with relatively coarse-grained messages, we have seen that the efficiency of communication is lower than expected. Large messages often have deleterious side effects, including potential queuing delays at the network interface, a single point of contention. Providing multiple network ports per processor is one potential solution to this problem. The Origin, by using fine-grained shared memory for data-transport (even for MPI), probably can avoid many of these effects, as large messages will automatically be interleaved by the hardware. Fine-grained interleaving may also be possible on the Cluster and SP2 as well, with better software support.

Implications of CPS Scaling The designers of the NPB chose to use CPS scaling as the figure of merit to judge machine scalability. While to first order this may seem reasonable, we have shown that many of the effects shown in this work due to the basic nature of CPS scaling. If some other scaling rule were used, such as time-contrained or memory-constrained scaling, we may have drawn different conclusions. However, we believe many of the effects presented here would hold true for other CPS benchmarks.

Many of our results are based on the one outstanding implication of CPS scaling: the problem size per node changes by orders-of-magnitude per node as the machine scales. This under publized causatum has profound implications. For example, a parallel machine nodes must handle working sets that scale from 100's of MB to 10's of MB. These effects of these inherent working set size changes will become more pronounced over time as caches grow larger and memory systems grow relativly father away from the CPU.

The communication per node also changes by orders of magnitude. If we had scaled out to 100, or even 1000 processors, the communication volume would scale down to less than 100 byte messages per node. Clearly, the parallel communication architecture must be able to handle not just bimodal communcation, e.g. large and small, but a continuos wide range of message sizes.

Lessons and Open Problems We have seen that this level of detailed analysis can yield insight into the complex behavior of applications under scaling. In the realm of benchmarks, the NAS programs would be considered the most regular and predictable; we were ourselves surprised at the complex effects derived from these programs.

Given this, there is clearly a need for better performance tools to help the programmer understand and tune performance in the face of such complexity. Performance counters were helpful, and our parallel Shade environment was critical in aiding our understanding.

The interaction between communication layers, such as MPI and Active Messages, needs to be better understood. Programmers of the NAS suite spent a good deal of time optimizing the communication of these programs by blocking small messages into larger ones to ensure scalability. Yet this directly lead to massive queuing delays in the system. Communication scheduling is clearly an issue.

We have shown that scalability is a complex phenomenon. For example, the linear speedups observed on the cluster are not a result of well scheduled and optimized communication. In fact, communication efficiency is quite low. Rather, they are the direct result of improved machine efficiency under scaling. The crossing of global working set thresholds and the resulting improvement in memory system efficiency boosts performance into the acceptable range.

Communication performance is often much lower than what would be expected from micro-benchmark results. However, communication efficiency only decreases slightly as processors are scaled. Complex effects such as queuing in the message layer and contention are often seen in practice. The design of the message-passing layer protocol can have a non-trivial and difficult to avoid impact on program performance.

Finally, network bisection bandwidth is critical for benchmarks with all-to-all communication patterns. The lack of adequate bisection bandwidth will directly limit their scalability. For benchmarks performing mostly nearest-neighbor communication, the bisection bandwidth is not as critical.

References

[1] Gheith Abandah. Characterizing Shared-Memory Applications: A Case Study of the NAS Parallel Benchmarks. Technical Report HPL-97-24, Hewlett-Packard Laboratories, June 1997.

[2] Argonne National Laboratory. *MPICH-A Portable Implmentation of MPI*, 1997. http://www.mcs.-anl.gov/mpi/mpich.

[3] David H. Bailey, T. Harris, Rob Van der Wigngaart, William Saphir, Alex Woo, and Maurice Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-010, NASA Ames Research Center, 1995.

[4] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet—A Gigabet-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–38, February 1995.

[5] Eric Brewer and Bradley C. Kuszmaul. How to Get Good Performance from the CM-5 Data Network. In *International Parallel Processing Symposium*, volume 8, pages 858–67, Cancun, MX, April 1994.

[6] Bob Cmelik and Doug Keppel. Shade: A Fast Instruction-set Simulator for Execution Profiling. In *Proceedings of the 1994 ACM SIGMETRICS Conference*, pages 128–137, May 1994.

[7] David E. Culler, Lok Tin Liu, Richard P. Martin, and Chad O. Yoshikawa. Assessing Fast Network Interfaces. *IEEE Micro*, 16(1):35–43, February 1996.

[8] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Atchitectural Requirements of Parallel Scientific Applications with Explict Communication. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 2–13, May 1993.

[9] Leonardo Dagum David H. Bailey, Eric Barszcz and Horst D. Simon. NAS Parallel Benchmark Results. Technical Report RNR-93-016, NASA Ames Research Center, 1993.

[10] Rob F. Van der Wigngaart. Efficient Implementatin of a 3-Dimensional ADI Method. In *Proceedings of Supercomputing '93*, pages 102–111, 1993.

[11] Jack Dongarra and Tom Dunigan. Message-Passing Performance of Various Computers. Technical Report UT-CS-95-299, University of Tennessee, Department of Computer Science, July 1995.

[12] Mark Hill. The Dinero Cache Simulator, August 1985. http://www.cs.wisc.edu/ larus/warts.html.

[13] Richard P. Martin, Amin M. Vahdat, David E. Culler, and Thomas E. Anderson. The Effects of Latency, Overhead and Bandwidth in a Cluster Architecture. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

[14] John D. McCalpin. Sustainable Memory Bandwidth in Current High-Performance Computers. http://reality.sgi.com/employees/mccalpin, 1995.

[15] NASA Ames Research Center. *NPB 2 Detailed Results*, 1997. http://science.nas.nasa.gov/Software/-NPB/NPB2Results.

[16] Steven K. Reinhardt, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, and David A. Wood. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In *Proceedings of the 1993 ACM SIGMETRICS Conference*, May 1993.

[17] Edward Rotherberg, Jawwinder Pal Singh, and Anoop Gupta. Working Sets, Cache Sizes and Node Granuarity Issues for Large Scale Multiprocessors. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 14–25, May 1993.

[18] William Saphir, Alex Woo, and Maurice Yarrow. NAS Parallel Benchmark 2.1 Results. Technical Report NAS-96-010, NASA Ames Research Center, 1996.

[19] Jaswinder Pal Singh, John Hennessy, and Anoop Gupta. Scaling Parallel Programs for Multiprocessor: Methodolgy and Examples. *Computer*, 26(7):42–50, jul 1993.

[20] Thorsten von Eicken, David E. Culler, Steh C. Goldstein, and Klaus Erik Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proc. of the 19th Int'l Symposium on Computer Architecture*, May 1992.

[21] Steven Cameron Woo, Moriwoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.