The Effects of Communication Parameters on End Performance of Shared Virtual Memory Clusters

Angelos Bilas and Jaswinder Pal Singh

Department of Computer Science
35 Olden Street
Princeton University
Princeton, NJ 08544

{bilas, jps}@cs.princeton.edu

Abstract

Recently there has been a lot of effort in providing cost-effective Shared Memory systems by employing software only solutions on clusters of high-end workstations coupled with high–bandwidth, low–latency commodity networks. Much of the work so far has focused on improving protocols, and there has been some work on restructuring applications to perform better on SVM systems. The result of this progress has been the promise for good performance on a range of applications at least in the 16–32 processor range. New system area networks and network interfaces provide significantly lower overhead, lower latency and higher bandwidth communication in clusters, inexpensive SMPs have become common as the nodes of these clusters, and SVM protocols are now quite mature. With this progress, it is now useful to examine what are the important system bottlenecks that stand in the way of effective parallel performance; in particular, which parameters of the communication architecture are most important to improve further relative to processor speed, which ones are already adequate on modern systems for most applications, and how will this change with technology in the future. Such information can assist system designers in determining where to focus their energies in improving performance, and users in determining what system characteristics are appropriate for their applications.

We find that the most important system cost to improve is the overhead of generating and delivering interrupts. Improving network interface (and I/O bus) bandwidth relative to processor speed helps some bandwidth-bound applications, but currently available ratios of bandwidth to processor speed are already adequate for many others. Surprisingly, neither the processor overhead for handling messages

nor the occupancy of the communication interface in preparing and pushing packets through the network appear to require much improvement.

Keywords: Distributed Memory, Shared Memory, Communication Parameters, Bandwidth, Latency, Host Overhead, Network Occupancy, Interrupt Cost, Clustering.

1 Introduction

With the success of hardware cache-coherent distributed shared memory (DSM), a lot of effort has been made to support the programming model of a coherent shared address space using commodity-oriented communication architectures in addition to commodity nodes. The techniques for the communication architecture range from using less customized and integrated controllers [17, 2] to supporting shared virtual memory (SVM) at page level through the operating system [12, 8]. While these techniques reduce cost, unfortunately they usually lower performance as well. A great deal of research effort has been made to improve these systems for large classes of applications. Our focus in this paper is on SVM systems.

In the last few years there has been much improvement of SVM protocols and systems, and several applications have been restructured to improve performance [12, 8, 25, 14]. With this progress, it is now interesting to examine what are the important **system** bottlenecks that stand in the way of effective parallel performance; in particular, which parameters of the communication architecture are most important to improve further relative to processor speed, which are already adequate on modern systems for most applications, and how will this change with technology in the future. Such studies can hopefully assist system designers in determining where to focus their energies in improving performance, and users in determining what system characteristics are appropriate for their applications.

This paper examines these questions through detailed architectural simulation using applications with widely different behavior. We simulate a cluster architecture with SMP nodes and a fast system area interconnect with a programmable network interface (i.e. Myrinet). We use a **home-based** SVM protocol that has been demonstrated to have comparable or better performance than other families of SVM protocols. The base case of the protocol, called home-based lazy release consistency (HLRC) does not require any additional hardware support. We later examine a variant, called automatic update release consistency (AURC) that uses automatic (hardware) propagation of writes to remote nodes to perform updates to shared data, and also extend our analysis to the use of uniprocessor nodes where this is useful. The major performance parameters we consider are the host processor **overhead** to send a message, the network interface **occupancy** to prepare and transfer a packet, the node-to-network **bandwidth** (often limited by I/O bus bandwidth), and the **interrupt cost.** We do not consider network link latency, since it is a small and usually constant part of the end-to-end latency, in system area networks (SAN). After dealing with performance parameters, we also briefly examine the impact of key granularity parameters of the communication architecture. These are the page size, which is the granularity of coherence, and the number of processors per node.

Assuming a realistic system that can be quite easily implemented today, and a range of applications that are well optimized for SVM systems [10], we see (Figure 1) that, for most applications, protocol and communication overheads are substantial. The speedups obtained in the realistic implementation are much lower than in the ideal case, where all communication costs are zero. This motivates the current research, whose goal is twofold. First, we want to understand how performance changes as

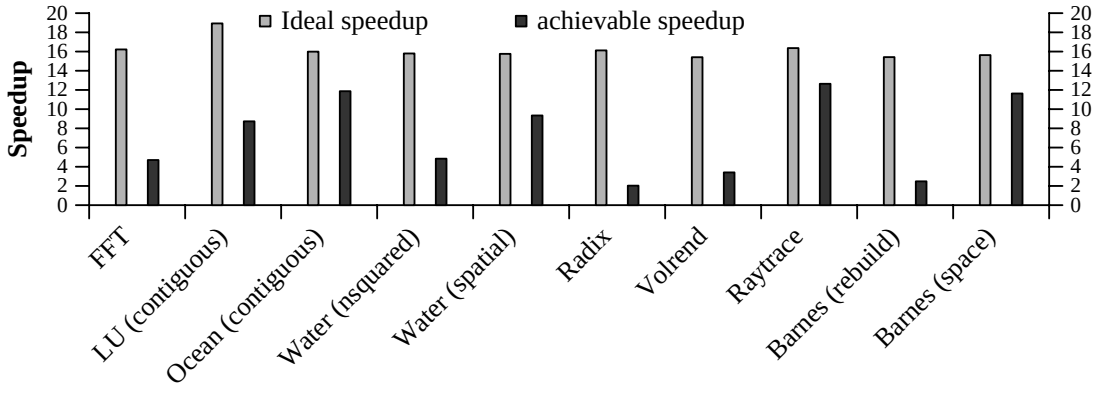

Figure 1. Ideal and realistic speedups for each application. The ideal speedup is computed as the ratio of the uniprocessor execution time divided by the sum of the compute and the local cache stall time in the parallel execution, i.e. ignoring all communication and synchronization costs. The realistic speedup corresponds to a realistic set of values (see Section 3) for communication architecture parameters today, in a configuration with four processors per node.

the parameters of the communication architecture are varied relative to processor speed, both to see where we should invest systems energy and to understand the likely evolution of system performance as technology evolves. For this, we use a wide range of values for each parameter. Second, we focus on three specific points in the parameter space. The first is the point for which the system generally achieves its best performance within the ranges of parameter values we examine. The performance on an application at this point is called its **best** performance. The second point is an aggressive set of values that the communication parameters can have in current or near-future systems, especially if certain operating system features are well optimized. The performance at this point in the space is called the **achievable** performance. The third point is the **ideal** point, which represents a hypothetical system that incurs no communication or synchronization overheads, taking into consideration only compute time and stall time on local data accesses. Our goal is to understand the gaps between the achievable, best and ideal performance by identifying which parameters contribute most to the performance differences. This leads us to the primary communication parameters that need to be improved to close the gaps.

We find, somewhat surprisingly, that host overhead to send messages and per-packet network interface occupancy are not critical to application performance. In most cases, interrupt cost is by far the dominant performance bottleneck, even though our protocol is designed to be very aggressive in reducing the occurrence of interrupts. Node-to-network bandwidth, typically limited by the I/O bus, is also significant for a few applications, but interrupt cost is important for all the applications we study. These results suggest that system designers should focus on reducing interrupt costs to support SVM well, and SVM protocol designers should try to avoid interrupts as possible, perhaps by using polling or by using a programmable communication assist to run part of the protocol avoiding the need to interrupt the main processor.

Our results show the relative effect of each of the parameters, i.e. relative to the processor speed and

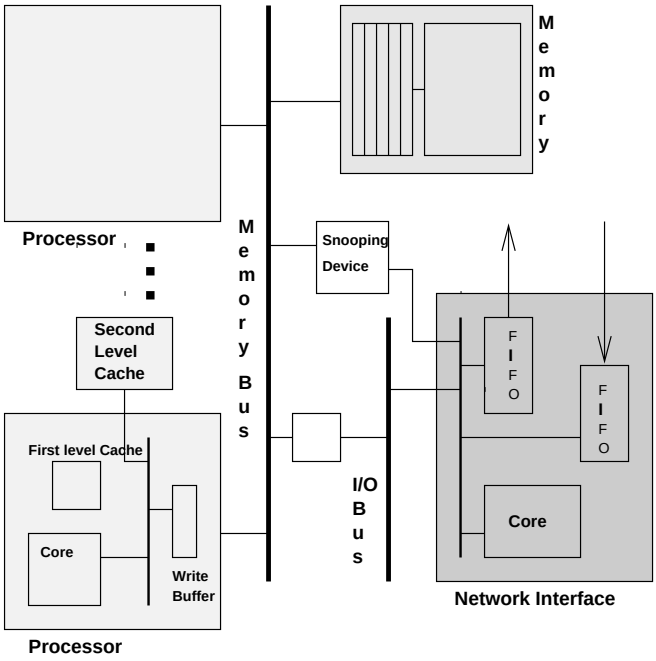

Figure 2. Simulated node architecture.

to one another. While the absolute parameter values that are used for the achievable set match what we consider achievable with aggressive current or near-future systems, viewing the parameters relative to processor speed allows us to understand what the behavior will be as technology trends evolve. For instance, if the ratio of bandwidth to processor speed changes, we can use these results to reason about system performance.

Section 2 presents the architectural simulator that we use in this work. In Section 3 we discuss the parameters we use and the methodology of the study. Section 4 presents the applications suite. Sections 5 and 6 present our results for the communication performance parameters. In Section 5 we examine the effects of each parameter on system performance and in Section 6 we discuss for each application the parameters that limit its performance. Section 7 presents the effects of page size and degree of clustering on system performance. We discuss related work in Section 8. Finally we discuss future work directions and conclusions in Sections 9 and 10 respectively.

2 Simulation Environment

The simulation environment we use is built on top of augmint [18], an execution driven simulator using the **x86** instruction set, and runs on **x86** systems. In this section we present the architectural parameters that we do not vary.

The simulated architecture (Figure 2) assumes a cluster of c–processor SMPs connected with a commodity interconnect like Myrinet [3]. Contention is modeled at all levels except in the network links and switches themselves. The processor has a P6-like instruction set, and is assumed to be a 1 IPC processor. The data cache hierarchy consists of a 8 KBytes first-level direct mapped write-through cache and a 512 KBytes second-level two-way set associative cache, each with a line size of 32 Bytes. The write buffer [19] has 26 entries, 1 cache line wide each, and a retire-at-4 policy. Write buffer stalls are

simulated. The read hit cost is one cycle if satisfied in the write buffer and first level cache, and 10 cycles if satisfied in the second-level cache. The memory subsystem is fully pipelined.

Each network interface (NI) has two 1 MByte memory queues, to hold incoming and outgoing packets. The size of the queues is such that they do not constitute a bottleneck in the communication subsystem. If the network queues fill, the NI interrupts the main processor and delays it to allow queues to drain. Network links operate at processor speed and are 16 bits wide. We assume a fast messaging system [5, 16, 4] as the basic communication library.

The memory bus is split-transaction, 64 bits wide, with a clock cycle four times slower than the processor clock. Arbitration takes one bus cycle, and the priorities are, in decreasing order: outgoing network path of the NI, second level cache, write buffer, memory, incoming path of the NI. The I/O bus is 32 bits wide. The **relative** bus bandwidth and processor speed match those on modern systems. If we assume that the processor has a 200 MHz clock, the memory bus is 400 MBytes/s.

Protocol handlers themselves cost a variable number of cycles. While the code for the protocol handlers can not be simulated since the simulator itself is not multi-threaded, we use for each handler an estimate of the cost of its code sequence. The cost to access the TLB from a handler running in the kernel is 50 processor cycles. The cost of creating and applying a diff is 10 cycles for every word that needs to be compared and 10 additional cycles for each word actually included in the diff.

The protocols we use are two versions of a home–based protocol, HLRC and AURC [8, 25]. These protocols either use hardware support for automatic write propagation (AURC) or traditional software diffs (HLRC) to propagate updates to the home node of each page at a release point. The necessary pages are invalidated only at acquire points according to lazy release consistency (LRC). At a subsequent page fault, the whole page is fetched from the home, where it is guaranteed to be up to date according to the lazy release consistency [8]. The protocol for SMP nodes attempts to utilize the hardware sharing and synchronization within an SMP as much as possible, reducing software involvement [1]. The optimizations used include the use of hierarchical barriers and the avoidance of interrupts as much as possible. Interrupts are used only when remote requests for pages and locks arrive at a node. Requests are synchronous (RPC like), to avoid interrupts when replies arrive at the requesting node. Barriers are implemented with synchronous messages and no interrupts. Interrupts are delivered to processor 0 in each node. More complicated schemes (i.e. round robin, random assignment) that result in better load balance in interrupt handling can be used if the operating system provides the necessary support. These schemes however, may increase the cost of delivering interrupts. In this paper we also examine a round robin scheme.

3 Methodology

As mentioned earlier, we focus on the following performance parameters of the communication architecture: host overhead, I/O bus bandwidth, network interface occupancy, and interrupt cost. We do not examine network link latency, since it is a small and usually constant part of the end-to-end latency, in system area networks (SAN). These parameters describe the basic features of the communication subsystem. The rest of the parameters in the system, for example cache and memory configuration, total number of processors, etc. remain constant.

When a message is exchanged between two hosts, it is put in a post queue at the network interface. In an asynchronous send operation, which we assume, the sender is free to continue with useful work. The network interface processes the request, prepares packets, and queues them in an outgoing network

queue, incurring an occupancy per packet. After transmission, each packet enters an incoming network queue at the receiver, where it is processed by the network interface and then deposited directly in host memory without causing an interrupt [2, 4]. Thus, the interrupt cost is an overhead related not so much to data transfer but to processing requests.

While we examine a range of values for each parameter, in varying a parameter we usually keep the others fixed at the set of **achievable** values. Recall that these are the values we might consider achievable currently, on systems that provide optimized operating system support for interrupts. We choose relatively aggressive fixed values so that the effects of the parameter being varied are observed.

In more detail:

- Host Overhead is the time the host processor itself is busy sending a message. The range of this parameter is from a few cycles to post a send in systems that support asynchronous sends, up to the time needed to transfer the message data from the host memory to the network interface when synchronous sends are used. If asynchronous sends are available, an achievable value for the host overhead is a few hundred processor cycles. Recall that there is no processor overhead for a data transfer at the destination end. The range of values we consider is between 0 (or almost 0) processor cycles and 10000 processor cycles (about 50μs with a 5ns processor clock). Systems that support asynchronous sends will probably be closer to the smaller values and systems with synchronous sends will be closer to the higher values depending on the message size. The achievable value we use is an overhead of 600 processor cycles per message.

- The I/O Bus Bandwidth determines the host to network bandwidth (relative to processor speed). In contemporary systems this is the limiting hardware component for the available node-to-network bandwidth; network links and memory buses tend to be much faster. The range of values for the I/O bus bandwidth is from 0.25 MBytes per processor clock MHz up to 2 MBytes per processor clock MHz (or 50 MBytes/s to 400 MBytes/s assuming a 200 MHz processor clock). The achievable value is 0.5 MBytes/MHz, or 100 MBytes/s assuming a 200 MHz processor clock.

- Network Interface Occupancy is the time spent on the network interface preparing each packet. Network interfaces employ either custom state machines or network processors (general purpose or custom designs) to perform this processing. Thus, processing costs on the network interface vary widely. We vary the occupancy of the network interface from almost 0 to 10000 processor cycles (about 50μs with a 5ns processor clock) per packet. The achievable value we use is 1000 main processor cycles, or about 5μs assuming a 200 MHz processor clock. This value is realistic for the currently available programmable NIs, given that the programmable communication assist on the NI is usually much slower than the main processor.

- Interrupt cost is the cost to issue an interrupt between two processors in the same SMP node, or the cost to interrupt a processor from the network interface. It includes the cost of context switches and operating system processing. Although the interrupt cost is not a parameter of the communication subsystem, it is an important aspect of SVM systems. Interrupt cost depends on the operating system used; it can vary greatly from system to system, affecting the performance portability of SVM across different platforms. We therefore vary the interrupt cost from free interrupts (0 processor cycles) to 50000 processor cycles for both issuing and delivering an interrupt (total 100000 processor cycles or 500 μs with a 5ns processor clock). The achievable value we use is 500 processor cycles, which results in a cost of 1000 cycles for a null interrupt. This choice is significantly

more aggressive than what current operating systems provide. However it is achievable with fast interrupt technology [21]. We use it as the achievable value when varying other parameters to ensure that interrupt cost does not swamp out the effects of varying those parameters.

To capture the effects of each parameter separately, we keep the other parameters fixed at their achievable values. Where necessary, we also perform additional guided simulations to further clarify the results.

In addition to the results obtained by varying parameters and the results obtained for the achievable parameter values, an interesting result is the speedup obtained by using the best value in our range for each parameter. This limits the performance that can be obtained by improving the communication architecture within our range of parameters. The parameter values for the best configuration are: host overhead 0 processor cycles, I/O bus bandwidth equal to the memory bus bandwidth, network interface occupancy per packet 200 processor cycles and total interrupt cost 0 processor cycles. In this best configuration, contention is still modeled since the values for the other system parameters are still nonzero. Table 1 summarizes the values of each parameter. With a 200 MHz processor, the achievable set of values discussed above assumes the parameter values: host overhead 600 processor cycles, memory bus bandwidth 400 MBytes/s, I/O bus bandwidth 100 MBytes/s, network interface occupancy per packet 1000 processor cycles and total interrupt cost 1000 processor cycles.

Parameter	Range	Achievable	Best
Host Overhead (cycles)	0-10000	600	$\sim$0
I/O Bus Bandwidth (Mbytes/MHz)	0.25-2	0.5	2
NI Occupancy (cycles)	0-10000	1000	200
Interrupt Cost(cycles)	0-50000	500	$\sim$0

Table 1. Ranges and achievable and best values of the communication parameters under consideration.

4 Applications

We use the SPLASH-2 [22] application suite. This section briefly describes the basic characteristics of each application relevant to this study. A more detailed classification and description of the application behavior for SVM systems with uniprocessor nodes is provided in the context of AURC and LRC in [9]. The applications can be divided in two groups, regular and irregular.

4.1 Regular Applications

The applications in this category are FFT, LU and Ocean. Their common characteristic is that they are optimized to be single-writer applications; a given word of data is written only by the processor to which it is assigned. Given appropriate data structures they are single-writer at page granularity as well, and pages can be allocated among nodes such that writes to shared data are almost all local. In HLRC we do not need to compute diffs, and in AURC we do not need to use a write through cache policy. Protocol action is required only to fetch pages. The applications have different inherent and induced communication patterns [22, 9], which affect their performance and the impact on SMP nodes.

Application	Page Faults			Page Fetches			Local Lock Acquires			Remote Lock Acquires			Barriers
	1	**4**	8	1	**4**	8	1	**4**	8	1	**4**	8	1,**4**,8
FFT (20)	397.12	**251.89**	270.32	393.31	**167.17**	91.59	0.00	**0.00**	0.00	0.00	**0.00**	0.00	**1.14**
LU(contiguous) (512)	81.36	**56.61**	48.07	71.78	**34.94**	11.86	0.02	**0.22**	0.25	0.27	**0.07**	0.04	**19.24**
Ocean(contiguous) (514)	647.61	**117.34**	103.17	646.97	**24.92**	7.20	0.00	**0.76**	1.31	2.17	**1.41**	0.86	**13.05**
Water(nsquared) (512)	69.19	**22.06**	8.04	68.26	**19.01**	7.29	0.01	**120.36**	158.14	203.20	**82.85**	45.06	**3.30**
Water(spatial) (512)	97.86	**21.42**	9.23	93.81	**17.73**	6.04	0.01	**1.83**	2.60	3.94	**2.16**	1.39	**4.19**
Radix (1K)	208.82	**82.73**	98.40	203.69	**44.92**	13.41	0.10	**0.44**	3.30	4.52	**4.11**	1.33	**1.04**
Volrend (head)	105.09	**44.06**	34.49	104.78	**29.35**	6.53	0.00	**29.34**	43.80	44.34	**17.64**	3.97	**1.61**
Raytrace (car)	89.80	**25.64**	6.83	89.79	**25.57**	6.76	0.03	**2.21**	3.96	4.89	**3.26**	1.34	**0.10**
Barnes(rebuild) (8K)	211.22	**103.02**	55.47	207.72	**90.90**	40.31	0.07	**33.92**	71.76	127.74	**93.81**	55.18	**1.44**
Barnes(space) (8K)	48.06	**10.43**	7.67	46.20	**9.92**	3.48	0.00	**0.16**	0.21	0.24	**0.07**	0.03	**1.79**

Table 2. Number of page faults, page fetches, local and remote lock acquires and barriers per processor per 10^7 cycles for each application for 1,4 and 8 processors per node.

FFT: The all-to-all, read-based communication in FFT is essentially a transposition of a matrix of complex numbers. We use two problem sizes, 256K(512x512) and 1M(1024x1024) elements. FFT has a high inherent communication to computation ratio.

LU: We use the contiguous version of LU, which allocates on each page data assigned to only one processor. LU exhibits a very small communication to computation ratio but is inherently imbalanced. We used a 512x512 matrix.

Ocean: The communication pattern in the Ocean application is largely nearest-neighbor and iterative on a regular grid. We run the contiguous (4-d array) version of Ocean on a 514x514 grid with an error tolerance of 0.001.

4.2 Irregular Applications

The irregular applications in our suite are Barnes, Radix, Raytrace, Volrend and Water.

Barnes: We ran experiments for different data set sizes, but present results for 8K particles. Access patterns in Barnes are irregular and fine-grained. We use two versions of Barnes, which differ in the manner they build the shared tree at each time step. In the first version (Barnes-rebuild, which is the one in SPLASH-2) processors load the particles that were assigned to them for force calculation directly into the shared tree, locking (frequently) as necessary. The second version, Barnes-space [10], is optimized for SVM, and it avoids locking as much as possible. It uses a different tree-building algorithm, in which disjoint subspaces that match tree cells are assigned to different processors. These subspaces include particles which are not the same as the particles that are assigned to the processors for force calculation. Each processor builds each own partial tree, and all partial trees are merged to the global tree without locking.

Radix: Radix sorts a series of integer keys. It is a very irregular application with highly scattered writes to remotely allocated data and a high inherent communication to computation ratio. We use the unmodified SPLASH-2 version.

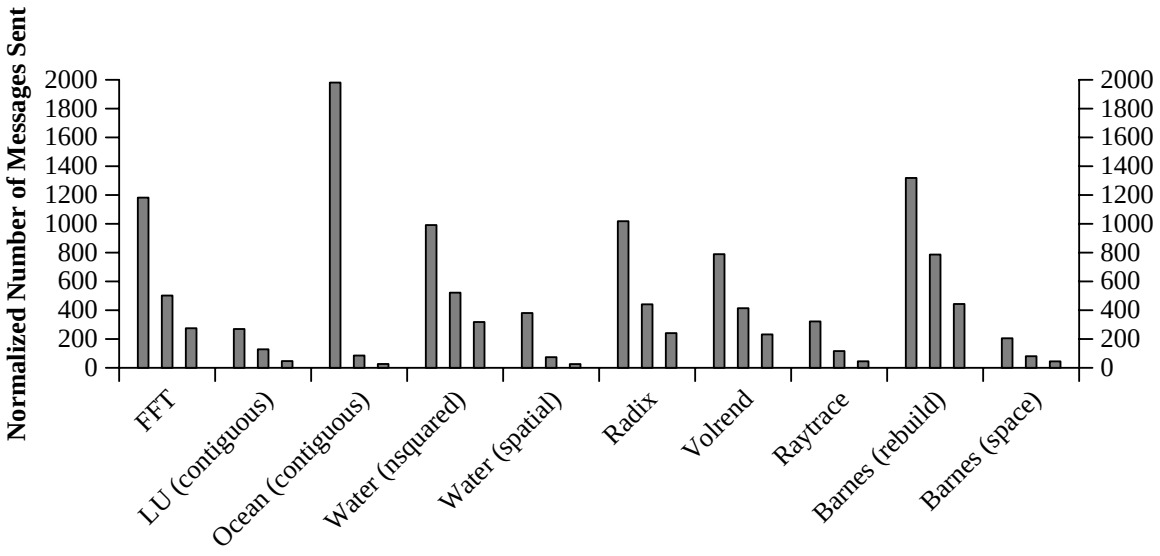

Figure 3. Number of messages sent per processor per 10^7 compute cycles for each application for 1,4 and 8 processors per node.

Raytrace: Raytrace renders complex scenes in computer graphics. The version we use is modified from the SPLASH-2 version to run more efficiently on SVM systems. A global lock that was not necessary was removed, and task queues are implemented better for SVM and SMP [10]. Inherent communication is small.

Volrend: The version we use is slightly modified from the SPLASH-2 version, to provide a better initial assignment of tasks to processes before stealing [10]. This improves SVM performance greatly. Inherent communication volume is small.

Water: We use both versions of Water from SPLASH-2, Water-nsquared and Water-spatial. Water-nsquared can be categorized as a regular application, but we put it here to ease the comparison with Water-spatial. In both versions, updates to water molecules positions and velocities are first accumulated locally by processors and then performed to the shared data once at the end of each iteration. The inherent communication to computation ratio is small. We use a data set size of 512 molecules.

Table 2 and Figures 3 and 4 can be used to characterize the applications. Table 2 presents counts of protocol events for each application, for 1, 4 and 8 processors per node (16 processors total in all cases). Figures 3 and 4 show the numbers of messages and MBytes of data (both application and protocol) that are sent by each processor in the system. These characteristics are measured per 10^7 cycles of application compute time per processor, and are averaged over all processors in the system. We can use them to categorize the applications in terms of the communication they exhibit. Both the number of messages and MBytes of data exchanged are important to performance; if we use the geometric mean of these properties, which captures their multiplicative effect, as a metric then we can divide the applications in

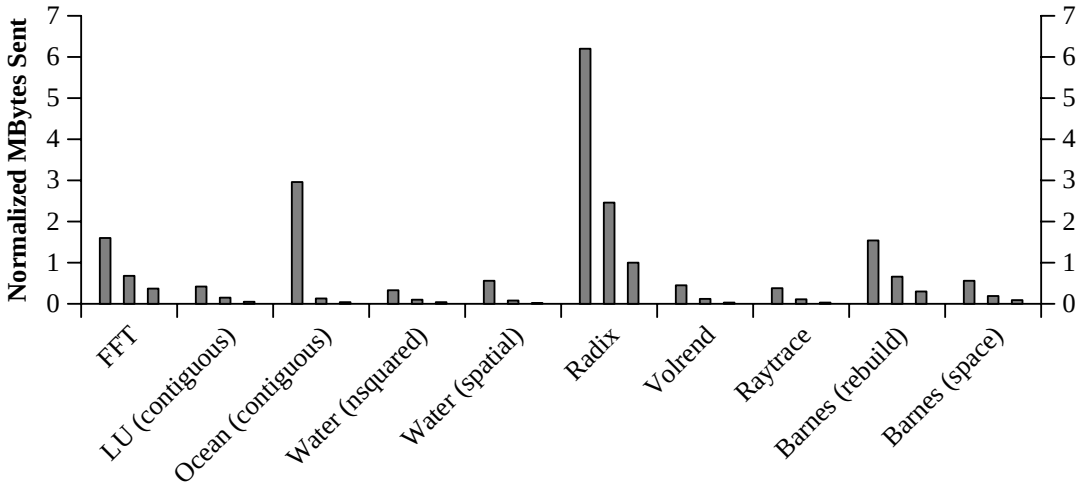

Figure 4. Number of MBytes sent per processor per 10^7 compute cycles for each application for 1,4 and 8 processors per node.

three groups. In the first group are Barnes-rebuild, FFT and Radix that exhibit a lot of communication. In the second group belong Water-nsquared and Volrend that exhibit less communication and in the third group the rest of the applications, LU, Ocean, Water-spatial, Raytrace and Barnes-space that exhibit very little communication. It is important to note that this categorization holds for the 4-processor per node configuration. Changing the number of processors in the node, can dramatically change the behavior of some applications, and the picture can be very different. For instance Ocean exhibits very high communication with 1 processor per node.

5 Effects of Communication Parameters

In this section we present the effects of each parameter on the performance of an all-software HLRC protocol for a range of values. Table 3 presents the maximum slowdowns for each application for the parameters under consideration. The maximum slowdown is computed from the speedups for the smallest and biggest values considered for each parameter, keeping all other parameters at their achievable values. Negative numbers indicate speedups. The rest of this section discusses the parameters one by one. For each parameter we also identify the application characteristics that most closely predict the effect of that parameter. The next section will take a different cut, looking at the bottlenecks on a per-application rather than per-parameter basis. At the end of this section we also present results for AURC.

Host Overhead: Figure 5 shows that the slowdown due to the host overhead is generally low, especially for realistic values of asynchronous message overheads. However, it varies among applications from less than 10% for Barnes-space, Ocean-contiguous and Raytrace to more than 35% for Volrend, Radix and Barnes-rebuild across the entire range of values. In general, applications that send more messages exhibit a higher dependency on the host overhead. This can be seen in Figure 6, which shows two

Application	Host Overhead	NI Occupancy	I/O Bus Bandwidth	Interrupt Cost	Page Size	Procs/Node
FFT	22.6%	11.9%	40.8%	86.6%	72.6%	13.8%
LU(contiguous)	17.9%	7.5%	15.9%	70.8%	34.4%	-35.3%
Ocean(contiguous)	4.5%	2.8%	6.5%	35.2%	19.6%	63.2%
Water(nsquared)	32.4%	16.6%	10.8%	83.2%	62.2%	-87.1%
Water(spatial)	23.7%	8.5%	8.9%	67.9%	51.0%	-87.5%
Radix	35.8%	-31.8%	77.6%	58.7%	-368.2%	-699.4%
Volrend	34.7%	12.8%	15.7%	91.3%	63.9%	-68.1%
Raytrace	8.2%	2.9%	8.9%	52.3%	9.1%	-16.1%
Barnes(rebuild)	40.7%	21.8%	44.8%	80.3%	71.5%	-383.4%
Barnes(space)	4.4%	-0.6%	27.5%	59.0%	-109.6%	-49.4%

Table 3. Maximum Slowdowns with respect to the various communication parameters for the range of values with which we experiment. Negative numbers indicate speedups.

curves. One is the slowdown of each application between the smallest and highest host overheads that we simulate, normalized to the biggest of these slowdowns. The second curve is the number of messages sent by each processor per 10^6 compute cycles, normalized to the biggest of these numbers of messages. Note that with asynchronous messages, host overheads will be on the low side of our range, so we can conclude that host overhead for sending messages is not a major performance factor for coarse grain SVM systems and is unlikely to become so in the near future.

Network Interface Occupancy: Figure 7 shows that network interface occupancy has even a smaller effect than host overhead on performance, for realistic occupancies. Most applications are insensitive to it, with the exception of a couple of applications that send a large number of messages. For these applications, slowdowns of up to 22% are observed at the highest occupancy values. The speedup observed for Radix is in reality caused by timing issues (contention is the bottleneck in Radix).

I/O Bus Bandwidth: Figure 8 shows the effect of I/O bandwidth on application performance. Reducing the bandwidth results in slowdowns of up to 82%, with 4 out of 11 applications exhibiting slowdowns of more than 40%. However, many other applications are not so dependent on bandwidth, and only FFT, Radix, and Barnes-rebuild benefit much from increasing the I/O bus bandwidth beyond the achievable relationships to processor speed today. Of course, this does not mean that it is not important to worry about improving bandwidth. As processor speed increases, if bandwidth trends do not keep up, we will quickly find ourselves at the relationship reflected by the lower bandwidth case we examine (or even worse). What it does mean is that if bandwidth keeps up with processor speed, it is not likely to be the major limitation on SVM systems for applications.

Figure 9 shows the dependency between bandwidth and the number of bytes sent per processor for each application. As before, units are normalized to the maximum of the numbers presented for each curve. Applications that exchange a lot of data, not necessarily a lot of messages, need higher bandwidth.

Interrupt Cost: Figure 10 shows that interrupt cost is a very important parameter in the system. Unlike bandwidth, it affects the performance of **all** applications dramatically, and in many cases a relatively

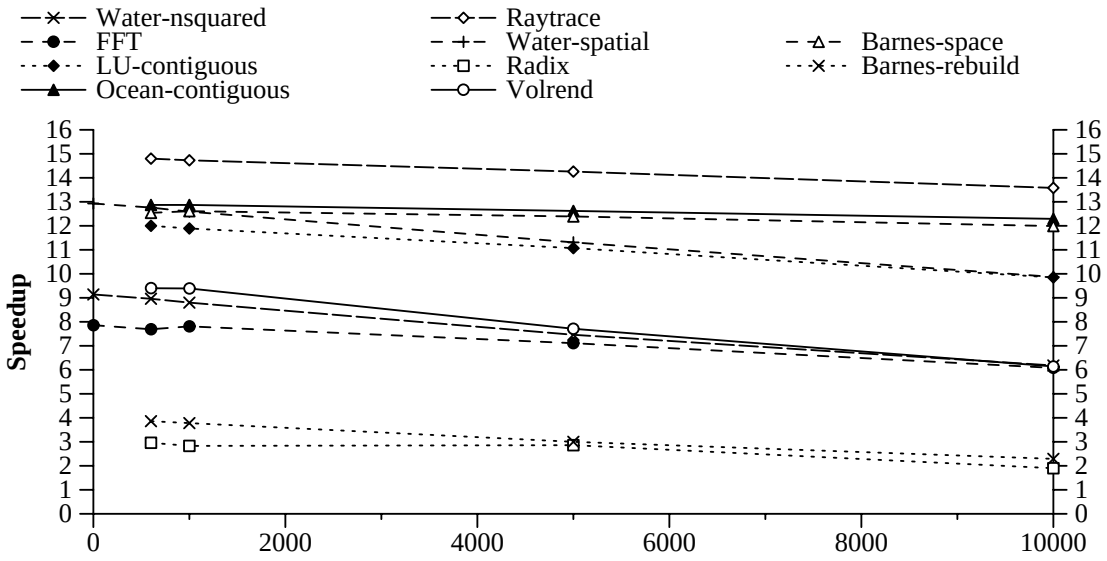

Figure 5. Effects of host overhead on application performance. The data points for each application correspond to a host overhead of 0, 600, 1000, 5000, and 10000 processor cycles.

small increase in interrupt cost leads to a big performance degradation. For most applications, interrupt costs of up to about 2000 processor cycles for each of initiation and delivery do not seem to hurt much. However, commercial systems typically have much higher interrupt costs. Increasing the interrupt cost beyond this point begins to hurt sharply. All applications have a slowdown of more than 50% when the interrupt cost varies from 0 to 50000 processor cycles (except Ocean-contiguous that exhibits an anomaly since the way pages are distributed among processors changes with interrupt cost). This suggests that architectures and operating systems should work harder to improving interrupt costs if they are to support SVM well, and SVM protocols should try to avoid interrupts as much as possible, Figure 11 shows that the slowdown due to the interrupt cost is closely related to the number of protocol events that cause interrupts—page fetches and remote lock acquires.

With SMP nodes there are many options for how interrupts may be handled within a node. Our protocol uses one particular method. Systems with uniprocessor nodes have less options, so we experimented with such configurations as well. We found that interrupt cost is important in that case as well. The only difference is that the system seems to be a little less sensitive to interrupt costs of between 2500 and 5000 cycles. After this range, performance degrades quickly as in the SMP configuration.

We also experimented with round robin interrupt delivery and the results look similar to the case where all interrupts are delivered to a fixed processor in each SMP. Overall performance seems to increase slightly, compared to the static interrupt scheme, but as in the static scheme it degrades quickly as interrupt cost increases. Moreover implementing such a scheme in a real system may be complicated and may incur additional costs.

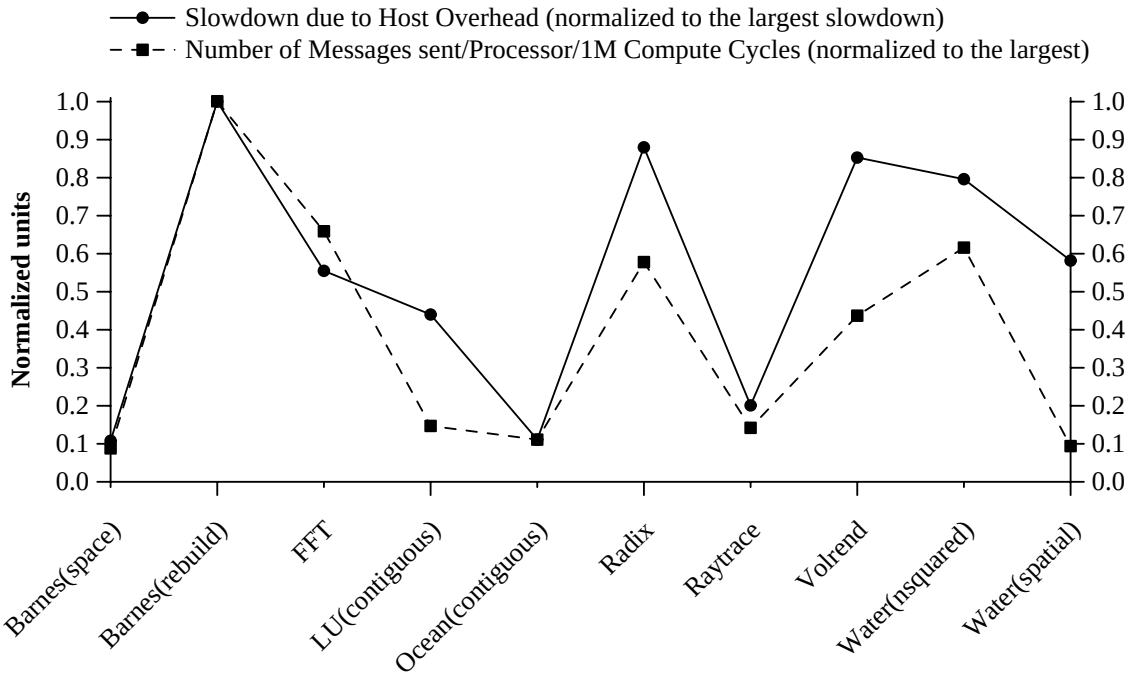

Figure 6. Relation between slowdown due to Host Overhead and Number of Messages sent.

AURC: As mentioned in the introduction, besides HLRC, we also used AURC to study the effect of the communication parameters when using hardware support for automatic write propagation instead of software diffs. The results look very similar to HLRC, with the exception that network interface occupancy is much more important in AURC. The automatic update mechanism may generate more traffic through the network interface because new values for the same data may be sent multiple times to the home node before a release. More importantly, the number of packets may increase significantly since updates are sent at a much finer granularity, so if they are apart in space or time they may not be coalesced well into packets. Figure 12 shows how performance changes as the NI overhead increases for both regular and irregular applications.

6 Limitations on Application Performance

In this section we examine the difference in performance between the **best** configuration and an ideal system (where the speedup is computed only from the compute and local stall times, ignoring communication and synchronization costs), and the difference in performance between the **achievable** and the **best** configuration on a per application basis. Recall that best stands for the configuration where all communication parameters assume their best value, and achievable stands for the configuration where the communication parameters assume their achievable values. The goal is to identify the application properties and architectural parameters that are responsible for the difference between the best and the

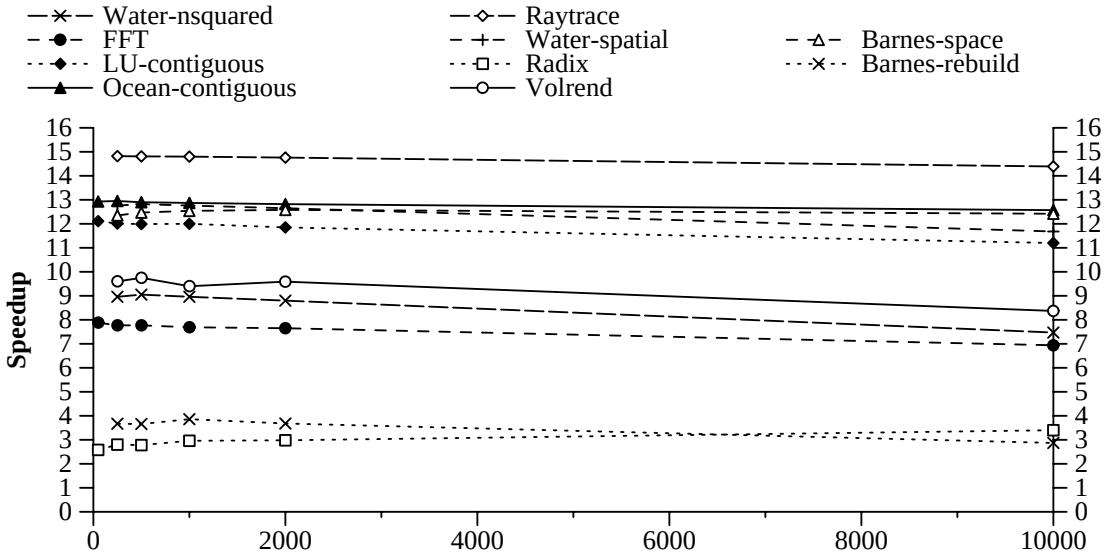

Figure 7. Effects of network interface occupancy on application performance. The data points for each application correspond to a network occupancy of 50, 250, 500, 1000, 2000, and 10000 processor cycles.

ideal performance, and the parameters that are responsible for the difference between the achievable and the best performance. The speedups for each configuration will be called **ideal, best** and **achievable** respectively. Table 4 shows these speedups for all applications. In many cases, the achievable speedup is close to the best speedup. However, in some cases (FFT, Radix, Barnes) there remains a gap. The performance with the best configuration is often quite far from the ideal speedup. To understand these effects, let us examine each application separately.

FFT: The best speedup for FFT is about 13.5. The difference from the ideal speedup of 16.2 comes from data wait time at page faults, which have a cost even for the best configuration, despite the very high bandwidth and the zero-cost interrupts. The achievable speedup is about 7.7. There are two major parameters responsible for this drop in performance: the cost of interrupts and the bandwidth of the I/O bus. Making the interrupt cost 0 results in a speedup of 11, while increasing the I/O bus bandwidth to the memory bus bandwidth gives a speedup of 10. Modifying both parameters at the same time gives a speedup almost the same as the best speedup.

LU: The best speedup is 13.7. The difference from the ideal speedup is due to load imbalances in communication and due to barrier cost. The achievable speedup for LU is about the same as the best speedup, since this application has very low communication to computation ratio, so communication is not the problem.

Ocean: The best speedup for Ocean is 10.55. The reason for this is that when the interrupt cost is 0 an anomaly is observed in first touch page allocation and the speedup is very low due to a large number of

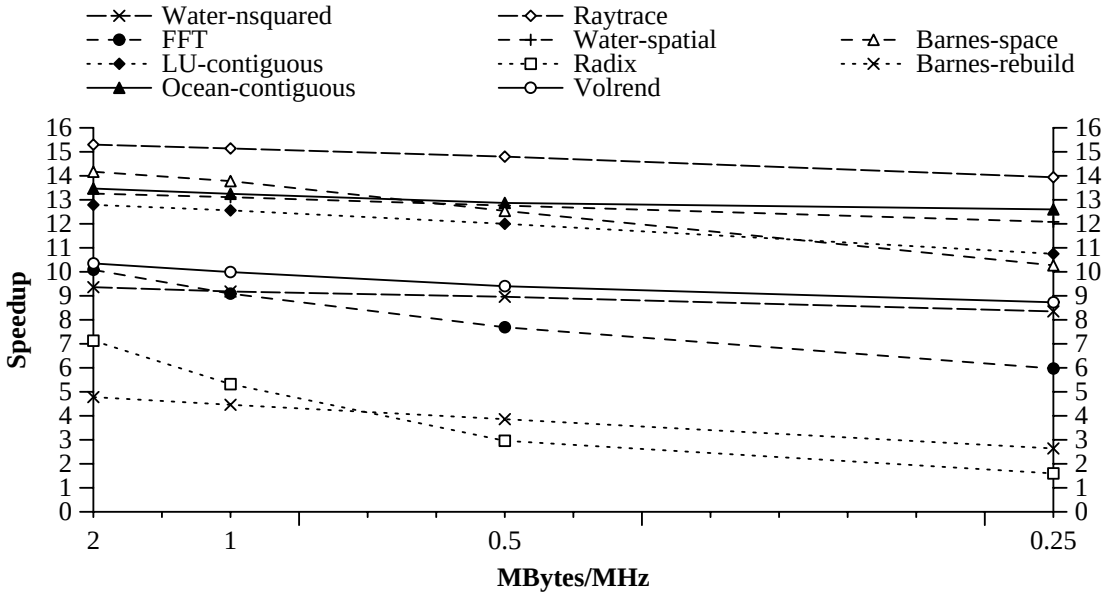

Figure 8. Effects of I/O bandwidth on application performance. The data points for each application correspond to an I/O bandwidth of 2, 1, 0.5 and 0.25 MBytes per processor clock MHz, or 400, 200, 100, and 50 MBytes/s assuming a 200 MHz processor.

page faults. The achievable speedup is 13.0, with the main cost being that of barrier synchronization. It is worth noting that speedups in Ocean are artificially high because of local cache effects: a processor's working set does not fit in cache on a uniprocessor, but does fit in the cache with 16 processors. Thus the sequential version performs poorly due to the high cache stall time.

Barnes-rebuild: The best speedup for Barnes-rebuild is 5.90. The difference from the ideal is because of page faults in the large number of critical sections (locks). The achievable speedup is 3.9. The difference between the best and achievable speedups in the presence of page faults is because synchronization wait time is even higher due to the increased protocol costs. These increased costs are mostly because of the host overhead (a loss of about 1 in the speedup) and the NI occupancy (about 0.8). To verify all these we disabled remote page fetches in the simulator so that all page faults appear to be local. The speedup becomes 14.64 in the best and 10.62 in the achievable cases respectively. The gap between the best and the achievable speedups is again due to host and NI overheads.

Barnes-space: The second version of Barnes we run is an improved version with minimal locking [10]. The best speedup is 14.5, close to the ideal. The achievable speedup is 12.5. The difference between these two is mainly because of the lower available I/O bandwidth in the achievable case. This increases the data wait time in an imbalanced way.

Water-Nsquared: The best speedup for Water-Nsquared is 9.9 and the achievable speedup is about 9. The reason for the not very high best speedup is page faults that occur in contended critical sections,

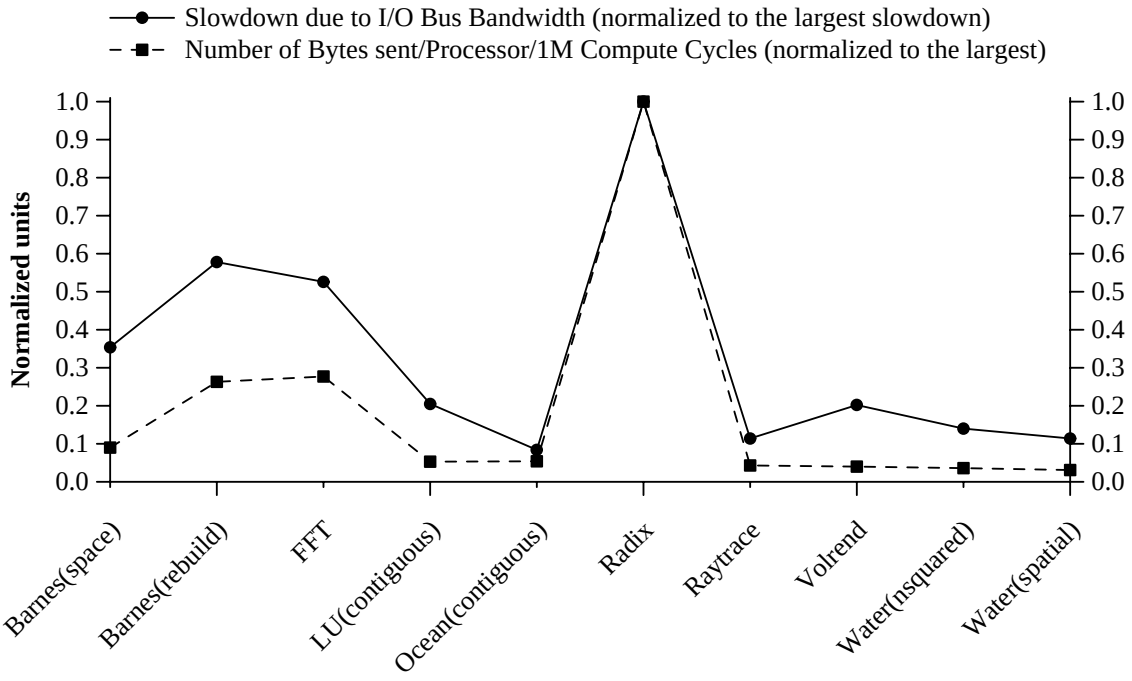

Figure 9. Relation between slowdown due to I/O Bus Bandwidth and Number of Bytes transferred.

greatly increasing serialization at locks. If we artificially disable remote page faults the best speedup increases from 9.9 to 14.1. The cost for locks in this artificial case is very small and the non-ideal speedup is due to imbalances in the computation itself.

Water-Spatial: The best speedup is 13.75. The difference from ideal is mainly due to small imbalances in the computation and lock wait time. Data wait time is very small. The achievable speedup is about 13.3.

Radix: The best speedup for Radix is 7. The difference from the ideal speedup of 16.1 is due to data wait time, which is exaggerated by contention even at the **best** parameter values, and the resulting imbalances among processors which lead to high synchronization time. The imbalances are observed to be due to contention in the network interface. The achievable speedup is only 3. The difference from the best speedup is due to the same factors: data wait time is much higher and much more imbalanced due to much greater contention effects. The main parameter responsible for this is I/O bus bandwidth. For instance, if we quadruple I/O bus bandwidth the achievable speedup for Radix becomes 7, just like the best speedup.

Raytrace: Raytrace performs very well. The best speedup is 15.64 and the achievable speedup 14.80.

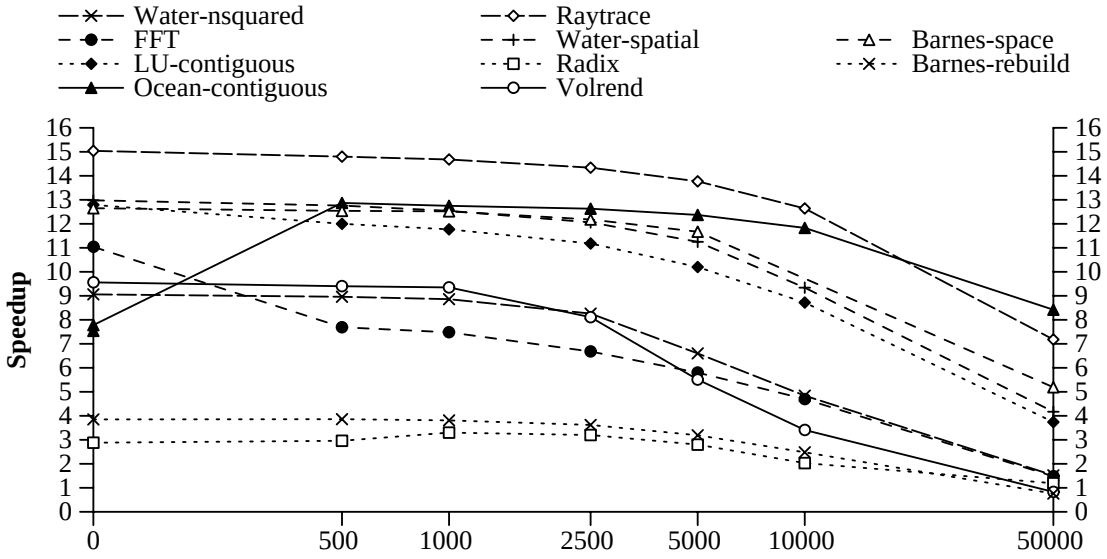

Figure 10. Effects of interrupt cost on application performance. The six bars for each application correspond to an interrupt cost of 0, 500, 1000, 2500, 5000, 10000, and 50000 processor cycles.

Volrend: The best speedup is 10.95. The reason for this low number is imbalances in the computation itself due to the cost of task stealing, and large lock wait times due to page faults in critical sections. If we artificially eliminate all remote page faults, then computation is perfectly balanced and synchronization costs are negligible (speedup is 14.9 in this fictional case). The achievable speedup is 9.40, close to the best speedup.

We see that the difference between ideal and best performance is due to page faults that occur in critical sections, I/O bandwidth limitations and imbalances in the communication and computation, and the difference between best and achievable performance is primarily due to the interrupt cost and I/O bandwidth limitations and less due to the host overhead. Overall, application performance on SVM systems today appears to be limited primarily by interrupt cost, and next by I/O bus bandwidth. Host overhead and NI occupancy per packet are substantially less significant, and in that order.

7 Page Size and Degree of Clustering

In addition to the performance parameters of the communication architecture discussed above, the granularities of coherence and data transfer–i.e. the page size–and the number of processors per node are two other important parameters that affect the behavior of the system. They play an important role in determining the amount of communication that takes place in the system, the cost of which is then determined by the performance parameters.

Page Size: The page size in the system is important for many reasons. It defines the size of the transfers, since in all software protocols data fetches are performed at page sizes. It also affects the

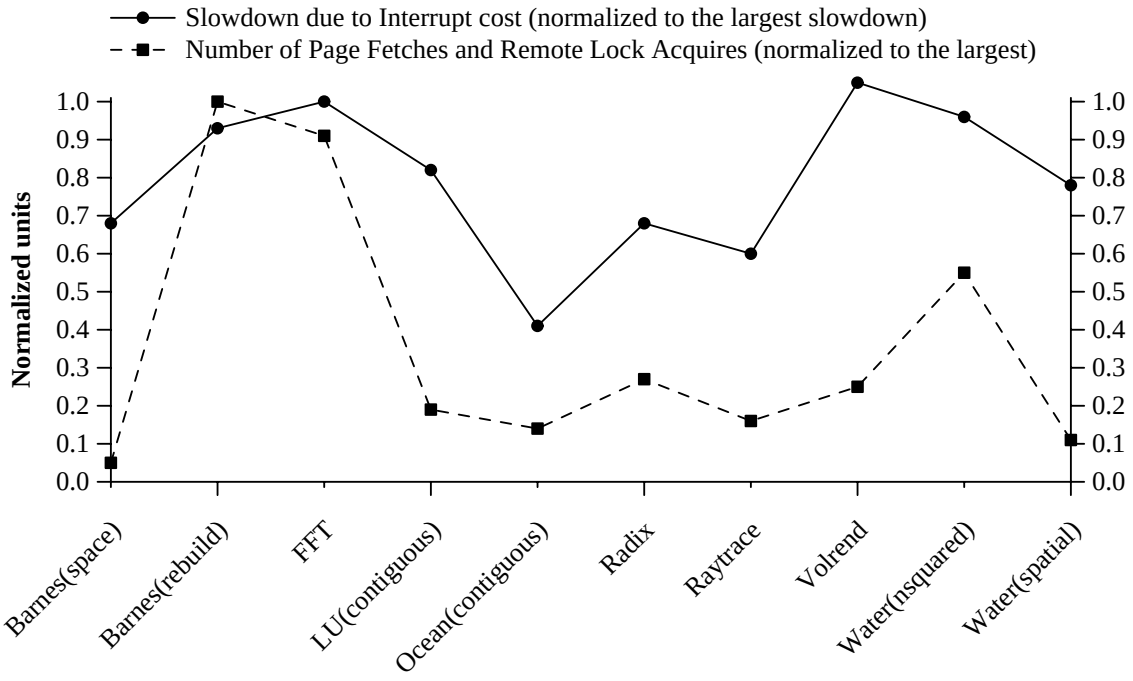

Figure 11. Relation between slowdown due to Interrupt cost and Number of Page Fetches and Remote Lock Acquires.

amount of false sharing in the system, which is very important for SVM. These two aspects of the page size conflict with each other: bigger pages reduce the number of messages in the system if spatial locality is well exploited in communication, but they increase the amount of false sharing, and vice versa. Moreover, different page sizes lead to different amounts of fragmentation in memory, which may result in wasted resources. Figure 13 shows that the effects of page size on applications vary a lot. Most applications seem to favor smaller page sizes, with the exception of Radix that benefits a lot from bigger pages. We vary the page size between 2 KBytes and 32 KBytes pages. Most systems today support either 4 KBytes or 8 KBytes pages. We should note two caveats in our study with respect to page size. First, we did not tune the applications specifically to the different page sizes. Second, the effects of the page size are often related to the problem sizes that are used. For applications in which the amount of false sharing and fragmentation (i.e. the granularity of access interleaving in memory from different processors) changes with problem size, larger problems that run on real systems may benefit from larger pages (i.e. FFT).

Cluster Size: The degree of clustering is the number of processors per node. Figure 14 shows that for most applications greater clustering helps even if the memory configuration and bandwidths are kept the same [1]. We use cluster sizes of 1, 4, 8 and 16 processors, always keeping the total number of processors

[1]This assumption, of keeping the memory subsystem the same and increasing the number of processors per node is not very realistic, since systems with higher degrees of clustering usually have a more aggressive memory subsystem as well,

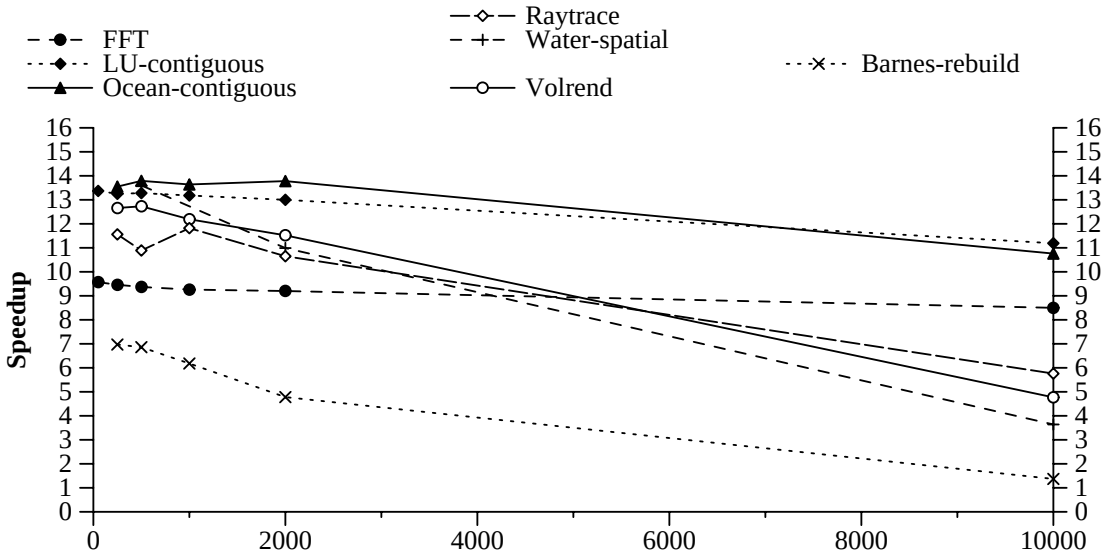

Figure 12. Effects of network interface occupancy on application performance for AURC. The data points for each application correspond to a network occupancy of 50, 250, 500, 1000, 2000, and 10000 processor cycles.

in the system at 16. These configurations cover the range from a uniprocessor node configuration to a cache-coherent, bus-based multiprocessor. Typical SVM systems today use either uniprocessor or 4-way SMP nodes. A couple of interesting points emerge. First, unlike most applications, for Ocean-contiguous the optimal clustering is four processors per node. The reason is that Ocean-contiguous generates a lot of local traffic on the memory bus due to capacity and conflict misses, and more processors on the bus exacerbate this problem. On the other hand, Ocean-contiguous benefits a lot from clustering because of the communication pattern. Thus when four processors per node are used, the performance improvement over one processor per node comes from sharing. When the system has more than four processors per node, the memory bus is saturated and although the system benefits from sharing, performance is degrading because of memory bus contention. Radix and FFT also put greatly increased pressure on the shared bus. The cross-node SVM communication however, is very high and the reduction in it via increased spatial locality at page grain due to clustering outweighs this problem. The second important point is that the applications that perform very poorly under SVM do very well on a shared bus system at this scale. The reason is that these applications either exhibit a lot of synchronization or make fine grain accesses, both of which are much cheaper on a hardware-coherent shared bus architecture. For example, applications where the problem in SVM is page faults within critical sections (i.e. Barnes-rebuild) perform much better on this architecture. These results show that bus bandwidth is not the most significant problem for these applications at this scale, and the use of hardware coherence and synchronization outweighs the problems of sharing a bus.

and are likely to provide greater node-to-network bandwidth.

Application	Best	Achievable	Ideal
FFT	13.5	7.7	16.2
LU	13.7	14.0	18.9
Ocean	10.5	13.0	16.0
Water(nsquared)	9.9	9.0	15.8
Water(spatial)	13.7	13.3	15.8
Radix	7.0	3.0	16.1
Volrend	10.9	9.40	15.4
Raytrace	15.6	14.8	16.4
Barnes(rebuild)	5.9	3.9	15.4
Barnes(space)	14.5	12.5	15.6

Table 4. Best and Achievable Speedups for each application

8 Related Work

Our work is similar in spirit to some earlier studies, conducted in [15, 7], but in different context. In [15], the authors examine the impact of communication parameters on end performance of a network of workstations with the applications being written in Split-C on top of Generic Active Messages. They find that application performance demonstrates a linear dependence on host overhead and on the gap between transmissions of fine grain messages. For SVM, we find these parameters to not be so important since their cost is usually amortized over page granularity. Applications were found to be quite tolerant to latency and bulk transfer bandwidth in the split-C study as well.

In [7], Holt et al. find that the occupancy of the communication controller is critical to good performance in DSM machines that provide communication and coherence at cache line granularity. Overhead is not so significant there (unlike in [15]) since it is very small.

In [11], Karlsson et al. find that the latency and bandwidth of an ATM switch is acceptable in a clustered SVM architecture. In [13] a Lazy Release Consistency protocol for hardware cache-coherence is presented. In a very different context, they find that applications are more sensitive to the bandwidth than the latency component of communication.

Several studies have also examined the performance of different SVM systems across multiprocessor nodes and compared it with the performance of configurations with uniprocessor nodes. Erlichson et al. [6] find that clustering helps shared memory applications. Yeung et al. in [23] find this to be true for SVM systems in which each node is a hardware coherent DSM machine. In [1], they find that the same is true in general for all software SVM systems, and for SVM systems with support for automatic write propagation.

9 Discussion and Future Work

This work shows that there is room for improving SVM cluster performance in various directions:

- Interrupts. Since reducing the cost of interrupts in the system can improve performance significantly, an important direction for future work is to design SVM systems that reduce the frequency

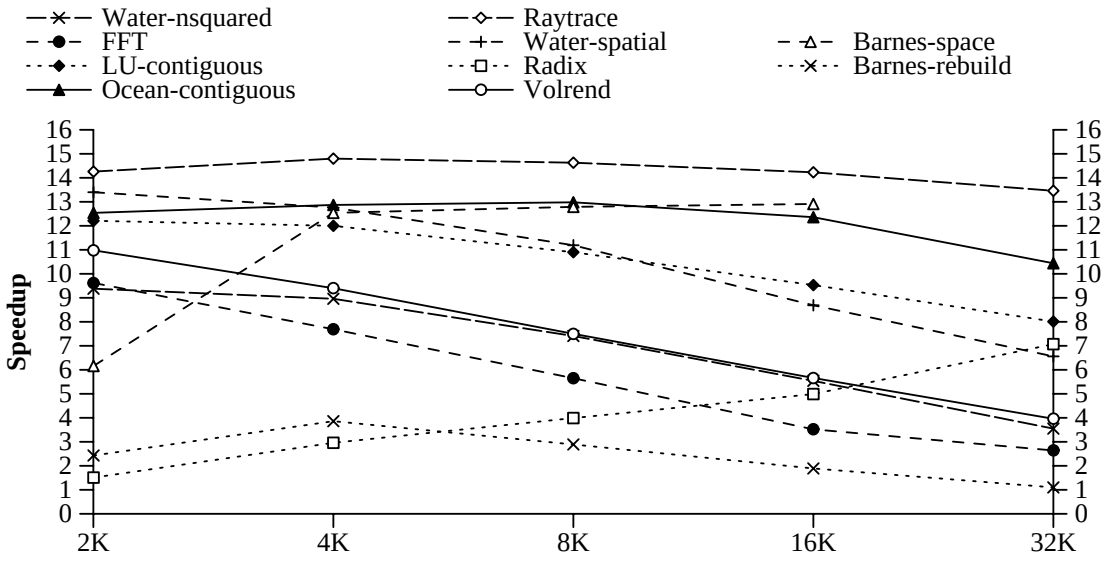

Figure 13. Effects of page size on application performance. The data points for each application correspond to a page size of 2 KBytes, 4 KBytes, 8 KBytes, 16 KBytes, and 32 KBytes.

and/or the cost of interrupts. Polling, better operating system support, or support for remote fetches that do not involve the remote processor are mechanisms that can help in this direction. Operating system and architectural support for inexpensive interrupts would improve system performance. Unfortunately this is not always achieved, especially in commercial systems. In these cases, protocol modifications (using non-interrupting remote fetch operations) or implementation optimizations (using polling instead of interrupts) can improve system performance and lead to more predictable and portable performance across different architectures and operating systems. Polling can be done either by instrumenting the applications or (in SMP systems) by reserving one processor for protocol processing. Recent results for interrupts versus polling in SVM systems vary. One study finds that polling may add a significant overhead, leading to inferior performance than interrupts for page grain SVM systems [24]. On the other hand, Stets et al. find that polling gives generally better results than interrupts [20]. We believe more research is needed on modern systems to understand the role of polling. Another interesting direction that we are exploring is moving some of the protocol processing itself to the network processor found in programmable network interfaces like such as Myrinet, thus reducing the need for interrupting the main processor.

- System bandwidth. Providing high bandwidth is also important, to keep up with increasing processor speeds. Although fast system interconnects are available, software performance is, in practice, rarely close to what the hardware provides. Low level communication libraries fail to deliver close to raw hardware performance in many cases. Further work on low level communication interfaces may also be helpful in providing low-cost, high-performance SVM systems. Multiple network interfaces per node is another approach that can increase the available bandwidth. In this case protocol changes may be necessary to ensure proper event ordering.

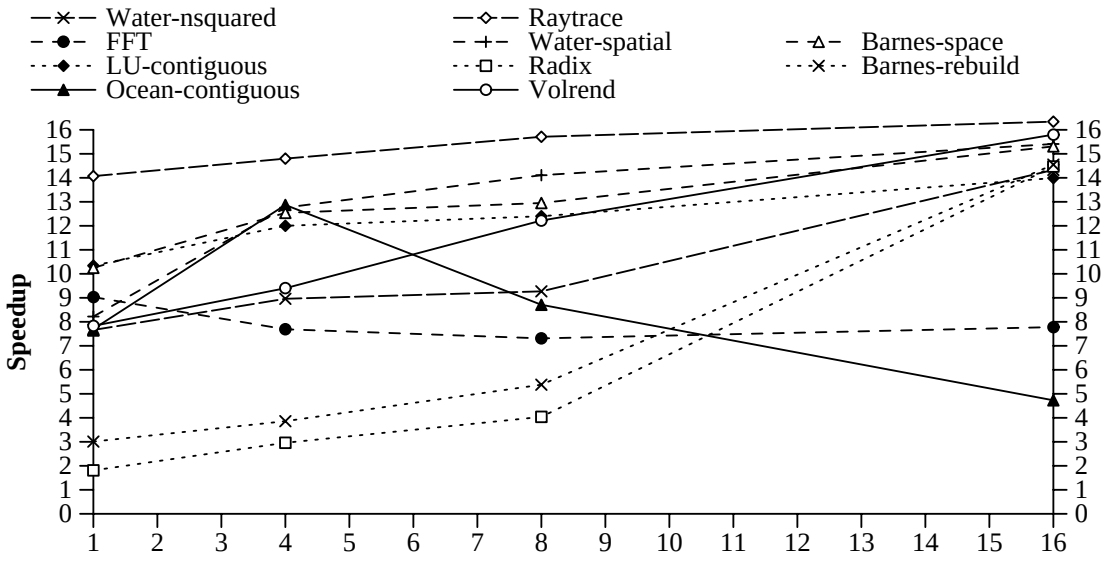

Figure 14. Effects of cluster size on application performance. The data points for each application correspond to a cluster size of 1, 4, 8, and 16 processors per node.

- Clustering. Up to the scale we examined, adding more processors per node helps in almost all cases. In applications where performance does not increase quickly with the cluster size, scaling of other system parameters, such as memory bus and I/O bandwidth, can have the desirable effects.

- Applications. In doing this work we found that restructuring applications is an area that can make a big difference. Understanding how an application behaves and restructuring it properly can dramatically improve performance far beyond the improvement in system parameters or protocols [10]. This however, is not always easy, and, unfortunately, not many tools are available in parallel systems to help easily discover the cause of bottlenecks and obtain insight about application restructuring needs, especially when contention is a major problem as it often is in commodity-based communication architectures. Architectural simulators are one of the few tools that can currently be used to understand how an application behaves in detail.

We should point out that this work is limited to a certain family of home-based SVM protocols. Other systems—for instance fine grain SVM systems—may exhibit different behavior and dependencies on communication parameters. Similar studies for other protocols and architectures can help us understand better the differences and similarities among SVM systems.

This work was based on a 16 processor system. To address the question of what happens in bigger systems we run some experiments with a 32 processor configuration and compared the number of protocol events between the two configurations. Table 5 shows the ratios of protocol events and communication traffic between a 32 and a 16 processor configuration. In most cases the event counts scale proportionally with the size of the system which leads us to believe that the results presented so far will hold for bigger configurations as well (at least up to 32 processors). Moreover, with larger problem sizes the problems related to the communication architecture are usually alleviated. However, more sophisticated

Application	Page Faults	Page Fetches	Remote Lock Acquires	Local Lock Acquires	Barriers	MBytes Sent	Messages Sent
FFT	1.46	1.70	-	-	2.00	1.75	1.70
LU	1.94	2.53	1.86	2.00	1.90	12.90 [2]	3.66
Ocean	0.75	0.53	2.77	1.57	1.99	2.50	1.95
Water-nsquared	2.89	2.63	1.40	2.50	1.99	2.80	2.37
Water-spatial	1.85	2.05	1.68	2.26	1.98	2.00	2.08
Radix	1.83	2.43	2.70	4.10	1.99	2.19	2.38
Volrend	1.45	1.79	-	-	1.98	1.90	1.35
Raytrace	2.08	2.08	1.33	2.40	2.00	2.08	1.83

Table 5. Ratios of protocol events for a 32 and a 16 processor configuration (4 processor per node).

scaling models, that take into account the problem size, may be necessary for more detailed and accurate predictions.

Another important question is how are these communication parameters going to scale with time. It seems that the parameters that closely follow hardware performance (host overhead, network interface occupancy, bandwidth) have more potential for getting better (relative to processor speeds) than interrupt cost which depends on the operating system and on special architectural support.

10 Conclusions

We have examined the effects of communication parameters to a family of SVM protocols. Through detailed architectural simulations of a cluster of SMPs and a variety of applications, we find that most applications are very sensitive to interrupt cost, and a few would benefit from improvements in bandwidth relative to processor speed as well. Unbalanced systems with relatively high interrupt costs and low I/O bandwidth can result in substantial losses in application performance. In these cases we observe slowdowns of more than 90% (a factor of 10 longer execution time). However, most applications are not sensitive to host overhead and network interface occupancy.

Most regular applications can achieve very good SVM performance under the **best** configuration of parameters. For irregular applications, though, even this best performance can be low. This is mainly due to serialization effects in critical sections, i.e. due to page faults incurred inside critical sections, which dilate the critical sections and increase serialization. For example by reducing the amount of locking by using a different algorithm for parallel tree building, the performance of Barnes improves by a factor of 2-3. Overall, the achievable application performance today is limited primarily by interrupt cost and then by node to network bandwidth. Host overhead and NI occupancy appear less important to improve relative to processor speed. If interrupts are free and bandwidth high relative to the processor speed, then the achievable performance approaches the best performance in most cases.

11 Acknowledgments

We thank Hongzhang Shan for making available to us the improved version of Barnes, and the anonymous reviewers for their comments and feedback.

References

[1] A. Bilas, L. Iftode, and J. P. Singh. Comparison of shared virtual memory across uniprocessor and SMP nodes. In **IMA Workshop on Parallel Algorithms and Parallel Systems**, Nov. 1996.

[2] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. A virtual memory mapped network interface for the shrimp multicomputer. In **Proceedings of the 21st Annual Symposium on Computer Architecture**, pages 142–153, Apr. 1994.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. **IEEE Micro**, 15(1):29–36, Feb. 1995.

[4] C. Dubnicki, A. Bilas, K. Li, and J. Philbin. Design and implementation of virtual memory-mapped communication on myrinet. In **Proceedings of the 1997 International Parallel Processing Symposium**, April 1997.

[5] T. Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrated communication and computation. In **Proceedings of the 19th Annual Symposium on Computer Architecture**, pages 256–266, May 1992.

[6] A. Erlichson, B. Nayfeh, J. Singh, and K. Olukotun. The benefits of clustering in shared address space multiprocessors: An applications-driven investigation. In **Supercomputing '95**, pages 176–186, 1995.

[7] C. Holt, M. Heinrich, J. Singh, E. Rothberg, and J. Hennessy. The effects of latency, occupancy and bandwidth in distributed shared memory multiprocessors. Technical Report CSL-TR-95-660, Stanford, Jan. 1995.

[8] L. Iftode, C. Dubnicki, E. W. Felten, and K. Li. Improving release-consistent shared virtual memory using automatic update. In **The 2nd IEEE Symposium on High-Performance Computer Architecture**, Feb. 1996.

[9] L. Iftode, J. P. Singh, and K. Li. Understanding application performance on shared virtual memory. In **Proceedings of the 23rd Annual Symposium on Computer Architecture**, May 1996.

[10] D. Jiang, H. Shan, and J. P. Singh. Application restructuring and performance portability on shared virtual memory and hardware-coherent multiprocessors. In **Sixth ACM Symposium on Principles and Practice of Parallel Programming**, June 1997.

[11] M. Karlsson and P. Stenstrom. Performance evaluation of cluster-based multiprocessor built from atm switches and bus-based multiprocessor servers. In **The 2nd IEEE Symposium on High-Performance Computer Architecture**, Feb. 1996.

[12] P. Keleher, A. Cox, S. Dwarkadas, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In **Proceedings of the Winter USENIX Conference**, pages 115–132, Jan. 1994.

[13] L. I. Kontothanasis, M. L. Scott, and R. Bianchini. Lazy release consistency for hardware-coherent multiprocessors. In **Supercomputing '95**, Nov. 1995.

[14] L. Kontothanassis, G. Hunt, R. Stets, N. Hardavellas, M. Cierniak, S. Parthasarathy, W. Meira, S. Dwarkadas, and M. Scott. VM-based shared memory on low-latency, remote-memory-access networks. **Proc., 24th Annual Int'l. Symp. on Computer Architecture**, June 1997.

[15] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effect of communication latency, overhead, and bandwidth on a cluster architecture. Technical Report CSD-96-925, Berkeley, Nov. 1996.

[16] S. Pakin, M. Buchanan, M. Lauria, and A. Chien. The Fast Messages (FM) 2.0 streaming interface. Submitted to Usenix'97, 1996.

[17] S. Reinhardt, J. Larus, and D. Wood. Tempest and typhoon: User-level shared memory. In **Proceedings of the 21st Annual Symposium on Computer Architecture**, pages 325–336, Apr. 1994.

[18] A. Sharma, A. T. Nguyen, J. Torellas, M. Michael, and J. Carbajal. Augmint: a multiprocessor simulation environment for intel x86 architectures. Technical report, University of Illinois at Urbana-Champaign, March 1996.

[19] K. Skadron and D. W. Clark. Design issues and tradeoffs for write buffers. In **The 3nd IEEE Symposium on High-Performance Computer Architecture**, Feb 1997.

[20] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. Scott. Cashmere-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network. In **Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16)**, Oct. 1997.

[21] D. Stodolsky, J. B. Chen, and B. Bershad. Fast interrupt priority management in operating system kernels. In USENIX Association, editor, **Proceedings of the USENIX Symposium on Microkernels and Other Kernel Architectures: September 20–21, 1993, San Diego, California, USA**, pages 105–110, Berkeley, CA, USA, Sept. 1993. USENIX.

[22] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. Methodological considerations and characterization of the SPLASH-2 parallel application suite. In **Proceedings of the 23rd Annual Symposium on Computer Architecture**, May 1995.

[23] D. Yeung, J. Kubiatowicz, and A. Agarwal. MGS: a multigrain shared memory system. In **Proceedings of the 23rd Annual Symposium on Computer Architecture**, May 1996.

[24] M. D. H. Y. Zhou, I. S. L. Iftode, B. R. T. K. Li, J. P. Singh, and D. A. Wood. Relaxed consistency and coherence granularity in DSM systems: A performance evaluation. Technical Report TR-535-96, Department of Computer Science, Princeton University, December 1996, 10 Pages.

[25] Y. Zhou, L. Iftode, and K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems. In **Proceedings of the Operating Systems Design and Implementation Symposium**, Oct. 1996.

Author Biography

Angelos Bilas received his B.S.E. degree in Computer Science from the Computer Engineering and Informatics Department, Patras University, Patras, Greece in 1993 and his M.A. degree in computer science from Princeton University in 1995. Currently, he is a Ph.D. student at Princeton University in the Department of Computer Science. His interests include distributed and parallel computing.

Jaswinder Pal Singh is an Assistant Professor in the Computer Science Department at Princeton University. He obtained his M.S. and Ph.D. degrees from Stanford University in 1989 and 1993, respectively, and his B.S.E. degree from Princeton in 1987. His research interests are at the boundary of parallel applications and multiprocessor systems. He has led the development and distribution of the SPLASH and SPLASH-2 suites of parallel programs, and is currently doing research in applications, systems and programming environments for supporting a shared address space programming model on various platforms.