Limits to the Performance of Software Shared Memory: A Layered Approach

Author names omitted for blind review process

Abstract

Much research has been done in fast communication on clusters and in protocols for supporting software shared memory across them. However, the end performance of applications that were written for the more proven hardware-coherent shared memory is still not very good on these systems. Three major layers of software (and hardware) stand between the end user and parallel performance, each with its own functionality and performance characteristics. They include the communication layer, the software protocol layer that supports the programming model, and the application. These layers provide a useful framework to identify the key remaining limitations and bottlenecks in software shared memory systems across clusters, as well as the key areas where efforts might yield major performance improvements. This paper performs such an integrated study, using this framework of layers, for two types of software shared memory systems: page-based shared virtual memory (SVM) and fine-grained software systems.

For the two system layers (communication and protocol), the focus of this study is on the performance costs of basic operations in the layers rather than on their functionalities (which bears further research). This is possible because their functionalities are now fairly mature. The less mature applications layer is treated through application restructuring. We examine the layers individually and in combination, understanding their implications for the two types of protocols and exposing the synergies among layers.

1 Introduction and Related Work

As clusters of workstations, PCs or symmetric multiprocessors (SMPs) become important platforms for parallel computing, there is increasing research interest in supporting the attractive, shared address space programming model across them in software. The traditional reason is that it may provide successful low-cost alternatives to tightly-coupled, hardware-coherent distributed shared memory (DSM) machines. A more important reason, however, is that clusters and hardware DSMs are emerging as the two major types of platforms available to users of multiprocessing, who would like to write parallel programs once and run them on both types of platforms: If the shared address space model cannot be supported efficiently on clusters, then message passing programming may dominate regardless of the success of hardware coherence in tightly coupled DSMs. Thus, despite (and, in fact, because of) the success of hardware-coherent DSM, software shared memory on clusters remains an important topic of research.

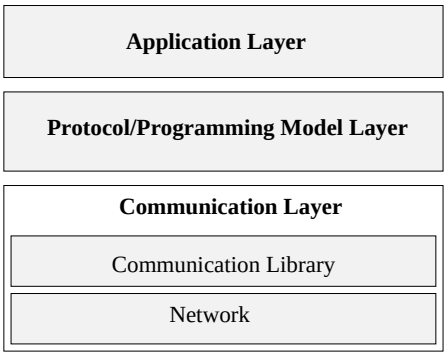

Figure 1: The layers that affect the end application performance in software shared memory.

Supporting a programming model gives rise to a layered communication architecture that is shown in Figure 1. The lowest layer is the *communication layer*: the communication hardware and the low level software libraries that provide basic messaging facilities. Next is the *protocol* layer that provides the programming model to the parallel application programmer. We assume all-software protocols in this paper, and focus on two fairly mature approaches: page-based shared virtual memory (SVM) and fine-grained or variable-grained access control. Finally, above the programming model or protocol layer runs the *application* itself.

The last decade has seen a lot of excellent research in the individual layers, especially the lower two, system layers. Commercial system area networks (SANs) [6, 11, 13] and network interfaces [5, 1] have reduced hardware latency and increased bandwidth, and are the key enabling technology for clusters. Fast user-level communication libraries [9, 23, 7, 33, 1, 25] have brought the latencies and especially bandwidths quite close to those afforded by the underlying hardware. Much excellent research has been done in the design and implementation of software shared memory protocols as well [20, 17, 19, 35, 10, 30]. Some research has also been done in restructuring applications to interact better with both the granularity and performance characteristics of software shared memory [16]. Still, software shared memory systems currently yield performance that is far behind that of efficient hardware-coherent systems, even at quite small scale.

This research examines the limitations and key sources of potential benefits to end application performance on software shared memory systems through a layered framework. The high level goals are to examine where the major gains (or losses) in the parallel performance of end application can or cannot come from in the future, both in the individual layers and through combinations of them, and to help cluster software and hardware designers determine where best to spend their energy for the goal of supporting software shared memory.

In the application layer, the chief variable is how an application is structured or orchestrated for parallelism. The two system layers (communication and protocol), however, have both functionality and performance characteristics, both of which contribute to end application performance. It is this space that we need to navigate. This paper takes one important cut through this research. It treats the functionality of the system layers as fixed (since their functionality is quite mature), and varies only their basic performance costs. The less mature, application layer is treated by examining the impact of application restructuring, as performed for SVM in [16]. We examine the impact of the layers individually and in combination, and also isolate the impact of different costs in each layer. (Research in providing different functionality in the communication layer and adapting the protocol to use this functionality can be seen as complementary with this work. It can benefit from this research by focusing in on functionality aimed at reducing the costs that we find here to be critical.)

Solving the problems at each layer has its own advantages, disadvantages, and potential for being realizable. It is under the control of different forces: technology; architects or protocol designers; and application/algorithm designers. Although application layer improvements are not under the control of system designers, they are still important to examine. Layer improvements that are not very beneficial for the original applications, as written for hardware-coherent systems, may become valuable when applications are restructured in more *performance portable* ways. Knowledge of actual trends in the layers (application or system) can be used to draw implications for the future of different approaches. For example, we are not so concerned at this stage with the realism of achieving the cost improvements that we examine (especially relative to growing processor speeds), but just with the limitations and potential benefits *if* those improvements could somehow achieve. Since degradations in layer performance relative to processor speed are not unlikely, we also examine these where relevant.

Some individual aspects of this work have been studied in the past. The impact of individual communication architecture parameters on performance has been studied for SVM [4] and for other programming models: hardware cache coherence [12] and a non-coherent shared address space [22]. However, it has not been studied for fine-grained or variable grained software shared memory, or by varying parameters simultaneously as we do here. Hardware support to accelerate protocols has been examined for SVM in the AURC [14] and Cashmere [19] protocols (as well as to perform diffs in hardware [3]), and for fine-grained software shared memory in the T-zero prototype [27]. However, a uniform approach based on protocol costs has not been changed.

A previous study [8] compared the performance of several protocols from a different class of SVM protocols than the one we use; it included the impact of switching network bandwidths between Ethernet and ATM and varying a unified software overhead cost between zero and a realistic value. The study included only three small kernels and one application. Fine-grained and page-grained approaches were compared on a particular, somewhat dated hardware platform with a fixed set of cost parameters in [36], with a focus on the impact of consistency models and coherence granularity. Analytical modeling of SVM performance is done in [24]. (Many other studies have examined the performance impact of protocol enhancements or modifications, but these issues are not the subject of this paper.)

This is the first study we are aware of that examines the benefits and limitations of all these protocol and communication cost issues in a common framework in which they can be varied individually and together, and for a wide range of applications. The contributions of this paper are: (i) the layered framework it provides for investigating performance issues, (ii) the integrated approach in which different layers are varied at the same time, (iii) studying these issues for the fine-grained protocol as well, (iv) examining several previously unexplored scenarios of varying one layer while keeping others fixed, (v) studying the synergies among layers, and (vi) providing detailed results and analysis for a wide range of applications. In addition to the main conclusions of the paper, we believe that this layered framework is a generally useful one for examining cluster performance.

Our highest level conclusions are: (i) For currently achievable system parameters the fine-grained and SVM approaches are competitive in performance, at least at the scale we examine. (ii) For the fine-grained approach, the most important layer to improve is the communication layer. (iii) For SVM, all three layers are important and exhibit a lot of synergy, and no one or even two of them will suffice for a wide range of applications; in general, the order of importance is application, communication, protocol. (iv) Overall, if only one system layer can be improved, it should be the communication layer (though different components of it matter for different protocols). More concrete conclusions are in Section 5. For SVM, the synergy among layers, and the dependence on all three rather than just the communication layer may appear intimidating, but it is in some ways a promising sign, especially since the communication layer may be difficult to improve relative to processor speed.

Section 2 describes the protocols, and Section 3 the methodology we use. Section 4 presents the main results of this work. It briefly examines some new results for the application layer and compares our protocols for a base system that is realistic today. Then it addresses the issue of modifying the protocol costs by keeping the communication layer fixed (relative to processor speed). In the next subsection we do the same for varying communication costs. These two subsections detail the individual impact of different parameters of their layers, and examine the combined impact with the layers before them. The last subsection of Section 4 examines the synergy that emerges between the different layers through the entire exploration. Finally, Section 5 concludes the paper.[1]

2 Protocols

The protocols we use for page-based SVM and fine-grained access control are well known, and among the state-of-the-art protocols, so we will describe them only briefly. Page-based coherence protocols use the virtual memory mecha-

[1] For a note on length, please see Section 4.4.

nism of uniprocessors for access control at page granularity [20]. Fine-grained access control can be provided by either hardware support or by code instrumentation in software [10]. In both cases, we assume the coherence protocol runs in software handlers rather than in hardware, and on the main processor rather than on co-processors.

For SVM, we use the Home-based Lazy Release Consistency (HLRC) protocol [35], which implements the well-known lazy release consistency (LRC) model [18] to reduce the impact of false sharing. Both HLRC and older LRC protocols use software *twinning* and *diffing* to solve the multiple-writer problem, but with different schemes for propagating and merging diffs (updated data). Traditional LRC schemes maintain distributed diffs at the writers, from where they must be fetched on a page fault [17]. In HLRC, the writer sends the diffs eagerly to a designated home node for the page, and the diffs are applied there to the home copy which is always kept up to date according to the consistency model. On a page fault, instead of fetching diffs from previous writers, the whole page is fetched from the home.

Fine-grained protocols are able to reduce the *occurrence* of false sharing by virtue of their fine granularity of coherence, so they do not rely heavily on relaxed consistency models. However, they require support for fine-grained access control. Our fine-grained (or variable-grained) implementation uses sequential consistency (SC), is based on the Stache protocol [26], and is similar in protocol structure to many directory-based hardware implementations. In fact, we call it the SC protocol from here on. Access control is assumed to be provided by hardware, at any fixed power of two granularity for a given application, as in the Typhoon-zero prototype [27]. We optimistically assume in our simulator that there is no performance cost for the hardware access control. The main reason for this assumption is described next.

Recently, the performance of fine-grained access control through software code instrumentation has been greatly improved [30]. However, good technology for instrumenting memory accesses with low overhead is not yet available for the x86 architecture we assume. The tradeoff between instrumentation-based and hardware access control for all-software protocols on commodity clusters is also not clear. For our purposes, simply adding a fixed number of instructions to each load and store does not appear to be an appropriate strategy to model instrumentation overhead. For these reasons, we provide parallel performance results assuming no instrumentation or access control overhead. Thus, while the results we obtain can be used to compare SVM with software SC assuming very efficient hardware access control, they cannot be used to directly compare SVM with a system like Shasta (however, readers may use them to extrapolate or obtain a rough idea).

The fact that we use the best-performing granularity for each application in the SC case requires some programmer intervention. The granularities used are 64 bytes in all other cases than the regular applications: FFT (4 KBytes), LU (4 KBytes) and Ocean (1 KByte).

3 Methodology

Although real SVM systems are available, and we have our own locally, a study that varies (and especially reduces) costs in the manner done here can only be done through simulation. We therefore use a detailed execution-driven simulator which we validate carefully against real systems. While computational clusters are moving toward using SMP rather than uniprocessor nodes, we perform this study assuming uniprocessor nodes. We do this for two reasons: First, protocols for SMP nodes are not so mature, and second there are many more interactions in SMP nodes that can affect performance in subtle ways [2].

Table 1 shows the applications and the problem sizes we use in this work. These applications are written for hardware shared memory systems and they are known to deliver excellent parallel speedups for hardware cache coherent systems at the 16-processor scale we assume in this paper. They are taken from the SPLASH-2 suite and from the restructured versions in [16], and will not be described further in this paper.

3.1 Simulation Environment

The simulation environment we use is built on top of `augmint` [31], an execution driven simulator using the *x86* instruction set, and runs on *x86* systems. It models cluster of 16 uniprocessor nodes connected with a commodity Myrinet-like interconnect [6] (Figure 2). Contention is modeled in great detail at all levels, including the network end-points, except in the network links and switches themselves. Thus, when we change protocol or communication layer costs, the impact on contention is included as well. Cache pollution due to protocol processing is also included. The processor has a P6-like instruction set, and is assumed to be a 1 IPC processor. The memory hierarchy within a node is modeled on that of the PentiumPro systems we use in our real implementation; it is kept constant in our experiments, and is described in Appendix.

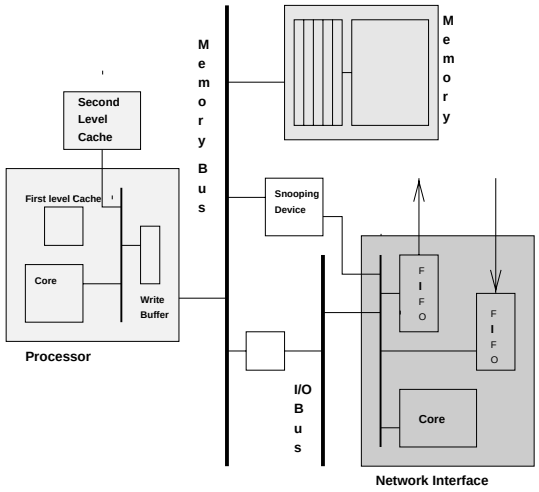

Figure 2: Simulated node architecture.

[2] Note to Referees: If it is helpful, we can present results for SMP nodes (instead of, or as well as) in the final version of this paper, if accepted. In short, SMP nodes help both protocols [29, 32], and while some protocol costs are higher we do not expect the results of the paper to change substantially. Another possibility we considered for inclusion is a fine-grained protocol that uses delayed consistency or single-writer, eager release consistency instead of sequential consistency. We have not included results for these cases to prevent overcrowding, but can include them if deemed desirable. Results show that these protocols are a little better than SC for most granularities smaller than a page (as in [36]) since they alleviate the effects of read-write false sharing, but not by a large amount.

Each network interface (NI) has two incoming and outgoing queues to hold incoming and outgoing packets. These queues are modeled as being in the NI's own SRAM memory (as in Myrinet). Enough NI memory is devoted to the queues that they do not fill in our experiments. Network links operate at processor speed and are 16 bits wide. We assume a fast messaging system [7] as the basic communication library.

Application	Problem Size	Instrum. cost
Barnes-original	8K particles	8%
FFT	1M points	29%
LU-Contiguous	512x512 matrix	29%
Ocean	514x514 grid	12%
Radix	1M keys	33%
Raytrace	car	29%
Volrend	256x256x256 CT head	5%
Water-Nsquared	512 mols	14%
Water-Spatial	512 mols	18%

Table 1: Applications, problem sizes and instrumentation costs. The last column is the instrumentation cost in Shasta.

Message handling may be done through interrupts or polling. Fine-grained protocols usually use polling—because messages are frequent and interrupts are expensive—typically by instrumenting the application code to poll at back-edges. Table 1 shows the instrumentation costs for the DEC Alpha instruction set and architecture in Shasta [30]. With the less frequent messages in SVM, it is less clear which method is better. However, it is known that when interrupts are used their cost is the most significant cost in the communication architecture [4]. For these reasons, and to make message handling uniform across protocols, we choose to assume polling instead of interrupts. Unfortunately, our lack of instrumentation tools for x86 means that we have to approximate the effects of polling. We must address both the instrumentation overhead for polling and the asynchronous message handling cost from when an incoming message is at the head of the polled queue to the time its hander begins executing instructions. Here too, we ignore the instrumentation cost. For polling, unlike for access control, this is fair even for comparing protocols since the mechanism would be the same for both types of protocols. The second, the asynchronous message handling, is dealt with by taking advantage of the fact that the number of messages that are *actually* handled is the same regardless of whether interrupts or polling are used. We therefore use an interrupt-like asynchronous mechanism in the simulator, but set the interrupt and message handling cost to what would be the message handling cost under polling. This cost is incurred once per message, as desired, and the only difference from true polling is in when exactly the cost is incurred: on a successful poll versus on an incoming request. Especially since the cost is small, we do not expect the approximation to be a major issue for this study.

We performed extensive validation of the simulator against real systems, the main results of which are discussed in Appendix 6.

3.2 Communication Layer Parameters

The communication layer parameters that are of most interest to us are the following. *Host overhead* is the time the host processor itself is busy sending a message, i.e. placing it in a buffer for the NI. We assume asynchronous send operations. *NI occupancy* is the time spent in the NI processor or state machine to prepare each packet of the message and place it in an output queue for transmission. Packets are of variable size, depending on how much of the message data is available to the NI at the time, and are up to 4 KBytes long. *I/O bus bandwidth* determines the host to network bandwidth (network links and memory buses tend to be much faster). Finally, *message handling cost* models the time from the moment a message reaches the head of the NI incoming queue to the moment the handler for the message starts to execute (time and contention before reaching the head is simulated in detail and is not included in this parameterized cost). Incoming data messages (as opposed to requests) do not invoke a handler, and are deposited directly in host memory by the NI without causing an interrupt or requiring a receive operation on the processor [5, 7]. Hardware link latency is kept fixed at 20 processor cycles since it is usually very small compared to the other costs.

Parameter	Achievable	Best	Worse	Real system
Host Overhead	600	0	1200	600-800
I/O Bus Bandwidth	0.5	2	0.25	0.45
NI Occupancy	1000	0	2000	1000
Message Handling Cost	200	0	400	N/A

Table 2: Communication parameter values. The first three columns describe the values we use in our study as the Achievable, Best and Worse sets of values. Units are in cycles (and bytes/cycle for bandwidth). If we assume a 1 IPC, 200 MHz processor the achievable values are from top to bottom 3μs, 100 MBytes/s, 5μs and 1μs. The numbers in parentheses are the same values in different units if we assume a 1IPC, 200 MHz processor. The fourth column for each application contains the values measured using a real communication library (VMMC [7] on a network of Intel Pentium nodes connected with Myrinet). The message handling cost is obtained from the polling-based Blizzard-S and Shasta systems (Personal communications.).

All costs we discuss for the communication architecture and the protocol are normalized to processor cycles. Table 2 shows the parameter values we use in the base system. We call this set of values *achievable* since they are modeled on a cluster of Intel PentiumPro based nodes connected by a Myrinet interconnect. Occupancy per packet is high because the processor in the Myrinet and most other commercial network interfaces is slow. However, since packets can be as large as 4 KBytes, occupancy can be amortized when applicable. Networks like the DEC Memory Channel can achieve lower one-way latencies per word, but they have lower bandwidths. Recall that all contention in the system is modeled in detail, so when costs are changed the effects on contention are simulated as well.

The other set of values we consider for the communication layer are: **A:** the original *achievable* set; **B:** the *best* set, where all protocol costs are set to zero; **H:** all values halfway between the two (more realistic than B); **W:** all values double of the achievable set (i.e. worse, relative to processor speed).

3.3 Protocol Cost Parameters

The protocol costs we vary and their *original* values in the base system are shown in Table 3. The page protection

Parameter	Original	Best	Halfway	Units
Page Protection	50	0	25	cycles/page
Diff creation	10,10	0,0	5,5	cycles/word
Diff Application	10	0	5	cycles/word
Twin Creation	10000	0	5000	cycles/word
Handler Cost	100+x	0+x	50+x	cycles

Table 3: Protocol cost parameter values. The diff creation cost incurs a cost per word that is compared (first number) and an additional cost per word that is put in the diff (second number). The handler cost is composed of a basic cost and any additional cost for operations (diffs, twins, etc) that may be executed by the handler.

cost is the cost of `mprotect`. A single `mprotect` call may be made to the kernel for a range of contiguous pages; it incurs a startup processing overhead, that is simulated, plus this per-page cost. This cost is aggressive for current operating systems but ought to be doable, and we verify that using a somewhat larger cost does not change the results much. The effects of twinning and diffing on the cache (and the misses incurred) are simulated. Protocol handlers themselves cost a variable number of cycles. The basic handler cost parameter is set to 100 cycles for both HLRC and SC; additional costs for diffing or traversing lists such as write notice lists (20 cycles per list element) are added to this cost. All these costs closely approximate those of our real implementation. The other set of values we consider for the protocol layer costs are: **O:** the *original* set; **B:** the *best* (idealized) set, where all costs are set to zero; **H:** all costs are *halfway* between original and best.

3.4 Presentation of Results

To save space, all the main speedup results in this paper—for different combinations of layer parameters—are contained in Figure 3. Breakdowns of execution time are presented in Figure 4. As we go through the next few sections, we shall focus on portions of the data in these figures. Let us therefore take a moment to describe the structure of how the results are presented.

For every application, there is a graph in Figure 3 with bars depicting speedups for different combinations of settings for the three layers. Each application graph has two major sets of bars, on the left for the HLRC protocol and on the right for the SC protocol. When an application has two versions, there are two sets of bars for each protocol, separated by a space: original and restructured. For ease of comparison, speedups are always measured with respect to the same best sequential version, and the order of arrangement of the bars for the restructured version is a mirror image of the order for the original version.

The combinations of system layers for which we show results are **AO** in black to focus attention and with an arrow pointing to it; then on one (the better) side of it **AB**, **BO**, **BH**, **BB**, **BB+** and **PRAM**; and on the worse side of it **WB** and **WO**. **PRAM** is the algorithmic speedup for each application assuming a PRAM model. This is usually the highest bar (except when super-linear cache effects dominate, as in Ocean and Volrend)[3] **BB+** stands for "better than best" communication costs; in this case, in addition to setting the above communication parameters to zero, the link cycle time is also set to zero cycles and I/O bus bandwidth is set to four bytes per cycle or twice the memory bus bandwidth. **BO**, **BH** and **BO** are shown in a different shading to more easily identify this set of bars that differs only in protocol costs (all with idealized communication costs). Since we do not run protocol cost variants for SC (as discussed later), the SC protocol has fewer bars. We also examine a number of other combinations with halfway (**H** cost parameters for a layer, since halfway improvements are more realistic than improvements to zero costs; to prevent overcrowding, we do not show these results but simply discuss them.

The breakdown graphs divide the time into components, such as busy time, local cache stall time, data wait time or time spent waiting for communicated data, lock wait time, barrier wait time, and protocol overhead. Breakdowns are shown averaged over all processors for a given execution, but this is only due to space reasons. To analyze the results we always refer to per-processor breakdowns and more detailed per-processor statistics gathered in the simulator to understand imbalances as well. The averaging may sometimes lead to discrepancies between the heights of the breakdown bars and the speedup bars; however, the alternative of showing the *worst* or last-to-finish processor instead is often unrepresentative in terms of the breakdowns.

In the following sections we present our results starting from the topmost layer, the application layer, and then continuing with the system layers, first the protocol and then the communication layer.

4 Results

Let us first look at some results on the base, AO, system, to make some basic comparisons and place the results in context with previous work. This includes comparing HLRC with SC with our system assumptions (Section 4.1), validating the application restructuring results for HLRC from [16] and examining application restructuring results for SC (which is new, in Section 4.2). Then, we will examine the impact of the system layers and the synergy.

4.1 HLRC and SC on Base Architecture

The results in Figure 3 can be used to compare the performance of the HLRC and SC protocols by looking at only the AO bars (the black bars with arrows). For both the original and the restructured applications, SC either equals or outperforms HLRC (of course, with the former always at its best granularity and with zero-cost access control). SC is especially better in applications that use locks frequently, like Barnes-original and to a lesser extent Water-Spatial and Volrend, and those like Radix in which coarse granularity causes a lot of false sharing. The only case in which HLRC does better is the restructured version of Barnes (Barnes-Spatial). Note, however, that this comparison does not account for access control costs in SC. With software instrumentation costs (see Table 1 with all caveats of Section 2), performance would be much closer, with each protocol outperforming the other in certain cases. This would also be similar to the results obtained on the Typhoon-zero platform with hardware access control and commodity protocol processing [36]. The communication parameters here are different than on Typhoon-zero: the AO platform has somewhat more bandwidth relative to processor speed, helping HLRC, but much more efficient access control, helping SC.

[3] We currently do not show PRAM results but 16-fold speedup. This will be rectified. PRAM speedups are close to 16 in all cases.

A true comparison with all-software SC awaits further research.

Qualitatively, then, it appears that the protocols are similar at this scale with realistic access control, differing based on application. For both protocols, however, the speedups on AO are clearly far from ideal, so there is a lot that can be gained from optimizing the layers.

4.2 Impact of the Application Layer

The original versions of the applications we use (Table 1) are described in [34]. Many characteristics relevant to SVM, including sharing patterns, message frequencies and message sizes, are described in [21, 36, 4]. These will not be repeated here. The restructured versions for SVM are described in [16].

If evolutionary communication trends hold relative to processor speed, and hardware support to reduce protocol costs is not forthcoming, application restructuring may be the only way to improve performance. For HLRC, the result of comparing the two black AO bars for an application (where restructured versions are used) is essentially that of [16]. Only the sources of the often large and sometimes dramatic improvements are important for this paper. They include (i) making remote access granularities larger (e.g. writing to a local buffer first in Radix), (ii) reducing locking and fine-grained synchronization at perhaps some cost in load balance (thus reducing PRAM speedups, e.g. in the tree building phase of Barnes), and (iii) in some cases improving the initial assignments of tasks so there is less need for task stealing (which is now very expensive due to synchronization and protocol activity, e.g. in Volrend; note that in Volrend restructuring also greatly improves false sharing and fragmentation in the image at page granularity, and hence data wait time).

For SC, examined for the first time here, restructuring helps significantly in cases where access granularity is made larger (e.g. Ocean), since it allows a larger granularity to be used successfully and overheads to be amortized, and when the need for task stealing is reduced (Volrend). However, restructuring that helps HLRC can sometimes hurt SC, such as in cases where load balance is compromised to reduce lock wait time (e.g. Barnes). Locks are not so problematic in SC as in SVM protocols: protocol activity is not postponed till synchronization points, and satisfying finer-grained block faults within critical sections is less costly than satisfying page faults, so there is less serialization.[4]. Instead, load imbalance matters more. Overall, restructuring is not beneficial in as many situations as in HLRC because the SC protocol looks a lot more like the hardware-coherent protocols for which the original applications are optimized.

4.3 Impact of the Protocol Layer

Even with restructuring, performance is not where we want it to be and improvements in the lower layers are clearly needed. This section first examines the impact of idealizing protocol costs (Table 3), i.e. moving from AO to AB. Protocol costs can be reduced by the node architect independently, by providing hardware support for protocol actions. Then, we examine the combinations of protocol cost improvements and application restructuring, i.e. looking at the AO and AB bars in the restructured case as well.

Impact for HLRC: Focusing only on the black AO bar and the AB bar just next to it for the original applications in Figures 3, we see that even the complete elimination of protocol costs usually does not lead to dramatic improvements in performance without improving the underlying communication architecture (with the exception of Raytrace). For coarse-grained-access, single-writer applications like FFT and LU, there is very little expensive protocol activity to begin with. Several of the other applications that use locks or have irregular access and invalidation patterns benefit to varying extents: Barnes-original, Water-Spatial and Volrend-original about 10%, and Radix 20% (though performance for Radix remains very poor). Greater improvements are seen in Ocean-Contiguous, Water-Nsquared and especially Raytrace.

We instrumented the simulator to understand which protocol costs take up most of the execution time. Table 4 shows the results for diff computation and protocol handler execution. The other components are very small. The applications in which handler execution time dominates protocol costs contain Ocean-Contiguous and Raytrace. In Ocean-Contiguous, for example, there is little diffing due to the single-writer nature, but remote access is fine-grained at column-oriented boundaries so there is a message per word of useful data. Raytrace has a very large number of fine-grained messages due to irregular access, so protocol handler cost is a large fraction of data wait time and processors spend a lot of time in the handlers themselves. There is relatively little synchronization and hence diff activity. For the other irregular applications, diff-related computation at synchronization points (such as their frequent locks) usually dominates. Water-Nsquared, a regular application, also computes many diffs for a lot of migratory data when it is updating forces.[5] When diff cost is a problem, hardware support for automatic write propagation [11, 5] can eliminate diffs [19, 14, 15], at the potential cost of contention and/or code instrumentation.

Improving protocol costs halfway (AH, not shown in the figures) usually provides about half or less of the benefit of eliminating them (more on this in Section 4.5). Finally, even idealized protocol costs together with application restructuring is insufficient to approach the desired (say, hardware-coherent) levels of performance. The large page granularity usually does not compensate for a less aggressive communication architecture even if protocol costs are magically made zero.

Impact for SC: SC does not have diffs or complicated protocol operations. There are only two protocol-related costs for SC: access control and protocol handlers. A rough idea of the impact of instrumentation costs for software access control (which we do not simulate, as discussed earlier) can be gleaned from Table 1. Since protocol handlers are very simple in SC, the cost of running the handlers is very small compared to the communication layer costs associated with handling the messages and invoking the handlers. Instru-

[4] this serialization turns out to be a major problem in the restructured Ocean-rowwise in HLRC, preventing it from improving performance as much as in SC

[5] This coarse-grained, regular case can be addressed by adaptively using a single-writer protocol instead for these data [2], which we do not explore here. Also, coarser-grained locking can be used in the update phase, but this does not help performance since it leads to more serialization at locks

Application	Total	Handler	Diff Compute
FFT	0.5	100	0
LU-Contiguous	0.6	80	20
Ocean-Contiguous	3.0	90	10
Barnes-Spatial	3.4	12.5	87.5
Radix	7.0	15	85
Radix-Local	8.3	16	84
Volrend	13.2	15	85
Water-Nsquared	10.7	20	80
Water-Spatial	3.2	6	94

Table 4: Percentage of Time Spent by Processors in Protocol Activity, and its Breakdown into Diff Computation and Protocol Handler Execution.

mentation of the original runs for SC shows that changing the cost of handlers (within a reasonable range) will not really affect performance, so we do not simulate different costs for SC protocol handlers.

4.4 Impact of the Communication Layer

Unlike protocol costs, communication layer parameters are largely constrained by technology trends and commodity products that designers may not be able to do much about. The exception is the processor overhead for sending and receiving messages, which can potentially be improved by hardware support. In recent times, overheads and bandwidths have scaled just a little bit slower than effective processor speed (about 50% per year versus about 70%) and NI occupancy can be assumed to scale with processor speed. It is useful to examine both what might be gained if thresholds or breakthroughs occur in communication performance and what might be lost in parallel speedups if it degrades relative to increasing processor speeds. (We did not examine a *worse* configuration for protocol costs since they are quite closely linked to processor speed.) Note that contention is still modeled in all parts of the system.

Let us first look at the impact of improving the communication architecture only; i.e., compare just the black, AO bar with the BO bar that is one bar away from it. The difference between protocols is interesting.

Impact for SC: For SC, Figure 3 shows that the best communication layer makes a dramatic difference to speedups, bringing them quite close to the ideal for the original applications. Except for the (small) protocol handler cost, the SC system becomes like hardware cache coherence with a very fast communication architecture. Of course, adding in instrumentation costs for software access control would make the speedup significantly worse, but the impact of the communication architecture is clear. The cases where BO speedups are substantially less than ideal (the Barnes cases) are due to load imbalance and the still significant synchronization cost.

Impact for HLRC: For HLRC, the effects of the best communication architecture alone (BO versus AO) are substantial but less dramatic than for SC. Applications that have little protocol activity, coarse-grained remote access and few or no locks, like FFT and LU, now achieve close to ideal performance just as with SC (for FFT, communication bandwidth is still a problem, so the "better than best" or BB+ configuration improves performance still). The irregular applications are helped substantially, but speedups with BO are still as low as 5-7 for several of them. The breakdowns shows that the data wait time is mostly eliminated, but time waiting at locks is often still high. Imbalances in this time cause time waiting at barriers to be high too. Comparatively, starting from AO, communication costs generally seem to have greater impact than protocol costs even for HLRC (with some exceptions like Water-spatial, Raytrace, and some restructured versions); but as we shall see in Section 4.5, once communication costs are improved even to halfway (or restructuring occurs), protocol costs begin to gain in importance.

Best-Best: Let us now consider moving both protocol and communication costs in the *best* configuration (BB) for HLRC. Two major problems cause critical sections to be dilated and hence serialization at locks in HLRC: protocol activity at synchronization points and page faults/fetches within the critical section. Moving to the best protocol cost essentially eliminates the first component. Improving the communication layer greatly reduces the second. For many applications, these enhancements together make a large difference, bringing speedups to the 12-15 range for BB. For example, in Volrend task stealing is now cheap enough to work well for load balancing (Per-processor breakdowns for the different configurations show the increased balance in compute times very well, but we do not show them for space reasons. We may do it in the final version.) However, even BB is not nearly enough in some cases. In Barnes-original, for example, the many critical sections in its tree-building phase each incur not one but several page faults due to communication. The *best* communication architecture still does not have infinite communication bandwidth, so fetching large pages is costly and serialization occurs at locks. The BB+ configuration, with much larger bandwidth and hence shorter critical sections, reduces lock wait time dramatically (Figure 4). The other irregular applications have fewer page faults in each critical section, so protocol activity affects critical section dilation and hence the performance improvement more than communication costs.

Radix is a case where communication bandwidth requirements and hence contention are extremely large. Protocol cost improvements don't help much. The data wait time is not only large but also very imbalanced among processors due to contention effects (again, this effect and its improvement is revealed very well in per-processor breakdowns, not shown). The BB speedup is only 4.6, and it takes the very unrealistic bandwidth of BB+ to improve the situation and raise speedup to over 11. (When contention is very high, we observe that sometimes a "worse" communication architecture performs better than a "better" one, due to throttling effects.)

Halfway and Worse: Improving the communication costs to the halfway point (HO, not shown) usually improves performance about halfway between AO and BO, for both HLRC and SC. Sometimes the improvement up to HO is smaller, since improvement in contention is nonlinear in improvement in occupancies. This effect is more pronounced for HB (between AB and BB), since protocol costs don't interfere (see Figure 3 for magnitudes). Finally, the results for a 2x worse communication architecture generally mirror those of moving to a better one, for both SC and HLRC.

This shows both that no strange threshold affects our results and that not improving communication performance as processor speed increases will indeed have a substantial impact on end performance. In the cases where improving only communication costs was more successful than improving only protocol costs, WB performs worse than the original AO, and WB to WO has less impact than AB to AO.

Effects of Individual Parameters: Figure 5 shows the impact of varying only one communication parameter at a time for some applications.[6] These data for HLRC were discussed in detail in [4], but have not been gathered for SC or compared with each other. We see that fine-grained SC depends mostly on overhead and occupancy, whereas HLRC depends mostly on bandwidth (overhead and occupancy don't matter much, and interrupts which dominated in [4] are not used). More importantly, these data show the points where crossovers in protocol performance might happen. Together with access control costs, they can be used to draw conclusions about choices among the protocols for specific communication architectures and for the future.

Overall, we conclude that dramatic improvements in communication relative to processor performance will dramatically improve data wait time and SC performance; for HLRC, performance improves substantially (usually more than for just improving protocol costs, at least starting from AO) but is not enough since a lot of the problem in irregular applications happens due to fine-grain synchronizations like locks. More work needs to be done in improving protocol costs and application structure as well. [7] Protocol costs are often more effective at reducing serialization at locks. However, in some applications even the BB case is not good enough to approach ideal performance, so either application restructuring or an even better bandwidth architecture like BB+ would be needed.

4.5 Synergy between Layers

We already know that application restructuring is very beneficial for HLRC but is not enough in itself [16]. Let us now examine both how improvements in the system layers affect the benefits of application restructuring and are affected by them, as well as (more generally) the synergy between pairs of layers.

Synergy in SC: For SC, since restructuring helps primarily due to the ability to use coarser granularity, the communication architecture is most important in reducing (or increasing, when made worse) the impact of restructuring. When instrumentation is used for access control, its cost may improve with restructuring (due to coarser access granularity and spatial locality), but this effect is much smaller. Even in Barnes, where the restructured version hurts rather than helps due to increased load imbalance, per-process time breakdowns reveal that the key imbalance is that among data wait times. A faster communication architecture reduces this imbalance, leading to less damage from restructuring. As for synergy between the system layers themselves, the costs that matter are communication and instrumentation overhead, and they are mostly quite orthogonal.

Synergy in HLRC: In HLRC we find a lot of interesting interactions and a lot more synergy among layers. First, how restructuring affects, and is affected by, the two system layers (relative to one another) depends on the application. In Barnes-original, restructuring reduces the amount of locking and serialization, and while its importance diminishes as system layers improve (as expected), it remains important all the way to BB+. Because reducing communication costs had a dramatic impact on lock serialization in the original, these costs affect the impact of restructuring more than protocol costs. While protocol costs take on greater relative significance after restructuring than before it, even in the restructured version communication costs are more important than protocol costs. For Ocean, the Ocean-rowwise version greatly reduces the number of messages, so the impact of message handling (protocol) cost is reduced. Unlike in Barnes, protocol costs impact the benefits of restructuring more than communication costs; also, the relative importance of the two system layers is reversed in the restructured version. Volrend and Radix behave similarly to Barnes in this respect: restructuring remains important as lower layers improve (just less so), and the impact is reduced more by communication costs than by protocol costs (increasing the importance of protocol costs in the restructured version). While Radix continues to have low performance, restructuring makes a substantial difference (66%) even at BB, and it is only at BB+ that the difference is small (about 12%).

In HLRC, the two system layers themselves show substantial synergy as well. For example, while protocol costs are limited in the improvement they can provide with the original communication architecture in many applications, once communication costs are dramatically reduced (even halfway) the impact of reducing protocol costs becomes much greater, even as a percentage improvement. For example in Volrend the performance improvement in going from AO to AB is 10% while BO to BB is 53%. The improvement in going from BO to BH is much smaller than BH to BB, indicating that protocol costs need to be reduced very aggressively to have a large effect. The same is found to be true in other cases (e.g. Water-nsquared and Raytrace), and for AO to AH to AB (not shown). In Barnes-original, AO to AB improves by 10% while BO to even BH improves by 63%, and even in a regular application like Ocean-Contiguous AO to AB improves 30% while BO to BB improves 60% (BO to BH only 20%). Similarly, the impact of the communication layer improvement is greater once the protocol costs are already reduced; for example, Ocean-Contiguous improves 43% from AO to BO but 71% from AB to BB, and Volrend improves 38% from AO to BO but 87% from AB to BB. And the difference in going from WB to WO is often smaller than that from AB to AO.

Overall, in SVM no two layers are enough to improve performance to rival hardware coherence for all applications, and all three layers are important to improve. The application layer and either of the lower layers combine to give a much greater improvement than any system layer does individually. In most cases, the order of impact is application followed by communication followed by protocol. For both HLRC and SC, the communication layer appears to be more important to making the programmer's job easier (i.e.,

[6] Note to Reviewers: Especially with the pages of figures, the paper is currently a little long. This is one possible section (including 1-page figure) to cut, that we'd appreciate feedback for a final version, if accepted.

[7] With an extremely aggressive communication architecture one could choose to implement a much simpler SVM protocol, such as SC, even at this coarse grain, but such architectures are presented as a limit and are not expected or argued here to be realistic.

making application restructuring less important). Restructuring changes the balance between the relative importance of protocol and communication costs, usually making protocol costs more important than they were before. Protocol costs also become more important once communication costs are improved (to say halfway), and vice versa. Table 5 summarizes for HLRC, for each application, whether communication cost or protocol cost is initially more important, whether or HB is better than BH, and what combinations of parameter values does it take to achieve about 10-fold speedup on 16 processors, which tells us both what is more important and what is needed. The need for improvement in all layers is apparent from the last category. More detailed trends can be obtained from the graphs.

5 Discussion and Conclusions

We have presented a framework for studying the limitations on the performance of software shared memory systems, in terms of the layers of software and hardware that can be improved and how they impact the end performance of a wide range of applications. We have studied the limitations and performance effects through detailed simulation, treating the system layers (protocol and communication layer) through only their cost parameters. The main limitations of the present work include the fact that it does not simulate true instrumentation-based fine-grained software shared memory, but rather a system with very efficient hardware access control but software protocols. Our main conclusions can be summarized as follows.

First, for currently achievable architectural values the variable-grained SC and page-grained HLRC protocols appear to perform quite similarly overall, especially if we factor in aggressive but realistic access control costs. We also assume that SC is allowed to choose the best granularity for each application; for cases like FFT where SC benefits from coarse granularity, we have found using a finer granularity to perform substantially worse.

Second, for SC the communication layer is key; protocol layer costs are not very significant, except for instrumentation cost when software access control is used. Application restructuring is also not so widely important as for HLRC, at least when starting from applications that are well tuned for hardware-coherent DSMs. Exceptions are when restructuring enables a coarse granularity of coherence to be used effectively (as in Ocean), and when it reduces the frequency of task stealing (Volrend). As might be expected, the communication parameters that matter most are overhead and NI occupancy, while bandwidth matters only when a coarser granularity is used.

Third, for HLRC, contrary to our results for SC, we find a much richer set of interactions. (i) Improvements in all three layers are often necessary to achieve performance comparable to efficient hardware-coherent systems. No one layer or even pair of layers is enough across the range of applications. (ii) There is synergy among the layers, in that improving one system layer (even halfway) allows the other to have greater impact, so that realistic improvements to both system layers relative to processor speed may go a good way, and improving the applications often does not substantially diminish the further impact of system layers. (iii) Application restructuring is used both to make access patterns to communicated data more coarse grained and to reduce the frequency of synchronization; when applicable, it usually outperforms the gains from idealizing any one other layer. Thus, if useful guidelines can be developed for programming SVM systems, they can be extremely helpful. (iv) Among the communication architecture parameters, the greatest dependence of HLRC starting from the achievable architecture is on bandwidth, and it is quite insensitive to improvements in the other parameters (when interrupts are used instead of polling, interrupt cost is a key bottleneck [4]). (v) Among protocol costs, the sensitivity is usually greatest to the costs associated with diffs, primarily diff creation. Handler cost can matter substantially in applications that are more constrained by data message count than by synchronization. Table 5 summarizes the key results for HLRC in terms of relative importance and what is needed for good performance.

For HLRC, the fact that it needs synergistic improvements in multiple layers to compete with hardware coherence is on one bad news for SVM compared to SC since work is required on many fronts. On the other hand, it is good news because there is less reliance on the layer that a cluster designer has the least control over (the communication layer), and the one for which it is unclear whether parameters can or will ever be improved relative to processor speed. Also, the parameter it relies on most, bandwidth, is the most likely to improve.

Fourth, for both protocols if only one system layer could be chosen for improvement, the communication architecture would be more significant. However, the costs in this layer may be less controllable by the system designer than the costs in the protocol layer, which can be improved by adding the appropriate hardware support to the node. Improving communication or protocol costs halfway usually achieves half, but sometimes quite a bit less, of the benefit of idealizing them (especially for protocol handler costs).

While this paper has not explored improving the system layers by altering their functionality or behavior instead of, or in addition to their costs, that is an important complementary area to pursue within the same framework of the layers. In fact, for SVM the synergy among the layers in terms of costs suggests that enhancing the functionality of system layers in an integrated way may lead to significant improvements.

6 Acknowledgments

Omitted for blind review process.

References

[1] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain, Colorado*, December 1995.

[2] J. Bennett, J. Carter, and W. Zwaenepoel. Adaptive software cache management for distributed shared memory architectures. In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 125–134, May 1990.

[3] R. Bianchini, L. I. Kontothanassis, R. Pinto, M. De Maria, M. Abud, and C. L. Amorim. Hiding communication latency and coherence overhead in software DSMs. In *Proc. of the 7th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOSVII)*, pages 198–209, Oct. 1996.

[4] A. Bilas and J. P. Singh. The effects of communication parameters on end performance of shared virtual memory clusters. In *In Proceedings of Supercomputing 97, San Jose, CA*, November 1997.

Application	Original					Restructured				
	BO > AB?	BH > HB?	Min Levels?			BO > AB?	BH > HB?	Min Levels?		
			Com.	Prot.	Comb.			Com.	Prot.	Comb.
FFT	**Yes**	**Yes**	H	–	HO	—	—	–	–	–
LU	Yes	Yes	A	O	AO	—	—	–	–	–
Ocean	Yes	No	–	–	BH/HB	**Yes**	Yes	H	–	HB/BH
Barnes	**Yes**	**Yes**	–	–	BB+	**Yes**	Yes	A	O	AO
Radix	**Yes**	**Yes**	–	–	BB+	**Yes**	**Yes**	–	–	BB+
Volrend	Yes	Tied	–	–	BH/HB	Yes	Tied	A	O	AO
Raytrace	**No**	**No**	–	B	AB	—	—	–	–	–
Water-Nsquared	No	No	–	B	AB/BH	—	—	–	–	–
Water-Spatial	**Yes**	Yes	B	–	BO/HB	—	—	–	–	–

Table 5: Summary of system layer impact for HLRC SVM. The columns for each version of each application are: (i) is communication more important than protocol to begin with (i.e. is BO better than AB); (ii) is BH better than HB, and (iii) what minimum level of each (communication or protocol), with original level of the other, and what combination of improved levels does it take to get about 10-fold or greater speedup on 16 processors? For example, an H under the entry called "Com." says that the halfway or better communication costs, together with any protocol costs starting from the original, is enough to obtain 10-fold speedup. (B+ for communication is excluded since it is too unrealistic). For (i) and (ii), bold font is used when the difference is large. A dash either means not applicable or "none".

[5] 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.

[6] 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.

[7] 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*, pages 388–396, April 1997.

[8] S. Dwarkadas, P. Keleher, A. Cox, and W. Zwaenepoel. An evaluation of software distributed shared memory for next-generation processors and networks. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993. To Get.

[9] 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.

[10] I. S. et al. Fine-grain access control for distributed shared memory. In *Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 297–307, October 1994.

[11] R. Gillett, M. Collins, and D. Pimm. Overview of network memory channel for PCI. In *Proceedings of the IEEE Spring COMPCON '96*, Feb. 1996.

[12] C. Holt, M. Heinrich, J. P. Singh, , and J. L. Hennessy. The effects of latency and occupancy on the performance of dsm multiprocessors. Technical Report CSL-TR-95-xxx, Stanford University, 1995.

[13] R. W. Horst and D. Garcia. ServerNet SAN I/O Architecture. In *Proceedings of Hot Interconnects V Symposium*, Stanford, Aug. 1997.

[14] 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.

[15] 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.

[16] D. Jiang, H. Shan, and J. P. Singh. Application restructuring and performance portability across shared virtual memory and hardware-coherent multiprocessors. In *Proceedings of the 6th ACM Symposium on Principles and Practice of Parallel Programming*, June 1997.

[17] 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.

[18] P. Keleher, A. Cox, and W. Zwaenepoel. Lazy consistency for software distributed shared memory. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, pages 13–21, May 1992.

[19] L. I. Kontothanassis and M. L. Scott. Using memory-mapped network interfaces to improve t he performance of distributed shared memory. In *The 2nd IEEE Symposium on High-Performance Computer Architecture*, Feb. 1996.

[20] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Trans. Comput. Syst.*, 7(4):321–359, Nov. 1989.

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

[22] 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.

[23] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for myrinet. In *Supercomputing '95*, 1995.

[24] E. W. Parsons, M. Brorsson, and K. C. Sevcik. Predicting the performance of distributed virtual shared memory applications. In *IBM Systems Journal, Vol 36, No. 4*, 1997.

[25] L. Prylli. Draft: BIP messages user manual for BIP 0.92. http://lhpca.univ-lyon1.fr/bip.html, July 1997.

[26] 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.

[27] S. K. Reinhardt, R. W. Pfile, and D. A. Wood. Decoupled hardware support for distributed shared memory. In *Proceedings of the 23rd Annual International Symposium on Computer Architecure*, pages 34–43, New York, May22–24 1006. ACM Press.

[28] R. Samanta, A. Bilas, L. Iftode, and J. P. Singh. Home-based svm protocols for smp clusters: Design, simulations, implementation and performance. In *To appear in Proceedings of the 4th International Symposium on High Performance Computer Architecture, Las Vegas*, February 1998.

[29] D. Scales and K. Gharachorloo. Towards transparent and efficient software distributed shared memory. In *Proceedings of the Sixteenth Symposium on Operating Systems Principles*, Oct. 1997.

[30] D. Scales, K. Gharachorloo, and C. Thekkath. Shasta: A low overhead, software-only approach for supporting fine-grain shared memory. In *The 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[31] 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.

[32] 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.

[33] H. Tezuka, A. Hori, and Y. Ishikawa. PM: a high-performance communication library for multi-user parallel environments. Technical Report TR-96015, Real World Computing Partnership, 1996.

[34] 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.

[35] 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.

[36] Y. Zhou, L. Iftode, J. Singh, K. Li, B. Toonen, I. Schoinas, M. Hill, and D. Wood. Relaxed consistency and coherence granularity in dsm systems: A performance evaluation. In *Proceedings of the 6th ACM Symposium on Principles and Practice of Parallel Programming*, June 1997.

Appendix: Parameters of a Simulated Node

The data cache hierarchy is an 8 KByte first-level direct mapped write-through cache and a 512 KByte second-level two-way set associative cache (line size 32B each). The L2 cache size matches that on the PentiumPro systems we use in our real implementation, and are large enough to not let capacity misses be a big problem for these applications and problem sizes. Write buffer stalls are simulated. The read hit cost is one cycle if satisfied in the write buffer or first level cache, and 10 cycles if satisfied in the second-level cache. The latency for a memory operation in the memory module is 70 cycles. The memory subsystem is fully pipelined, and the page size is 4 KBytes.

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, in decreasing order, are: outgoing network path of the NI, second level cache, write buffer, memory, incoming path of the NI (this order was generally the best in our experiments). The *relative* bus bandwidth and processor speed match those on modern P6-based systems. If we assume that the processor has a 200 MHz clock, the memory bus is 400 MBytes/s.

Appendix: Simulator Validation

We validated the simulator in two ways. First, we set the communication architecture parameters (relative to processor speed) close to those on the Typhoon-zero platform [27] and compared the performance of the two protocols. In this case, matching absolute performance exactly is unrealistic given the different processors and instruction set, but we found surprisingly close results (not shown for space reasons) for the differences in the protocols.

Second, we compared detailed breakdowns of execution time with those obtained on a real HLRC-SMP implementation on P6 SMP PCs connected by Myrinet [28]. The results are shown in figure 6. The speedups and breakdowns obtained from the simulator are very close to the implementation, which gives us good confidence in the simulator. The exception is Radix, in which there is a lot of the `mprotect` activity, for which our simulator is more aggressive than the real implementation, and a lot of contention that exaggerates all differences.

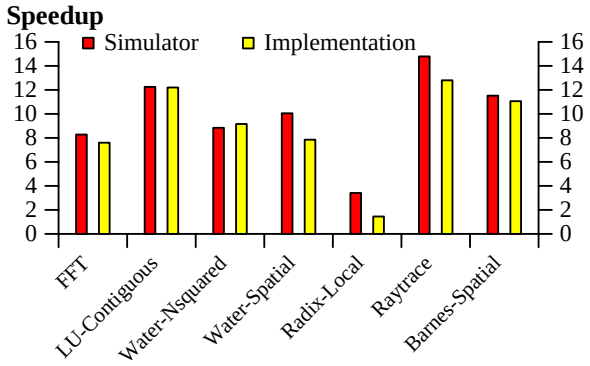

Figure 6: Simulator validation results. The implementation results are from HLRC-SMP on a network of Intel SMPs connected with Myrinet. The simulator results are for a similar configuration for a total of 16 processors.

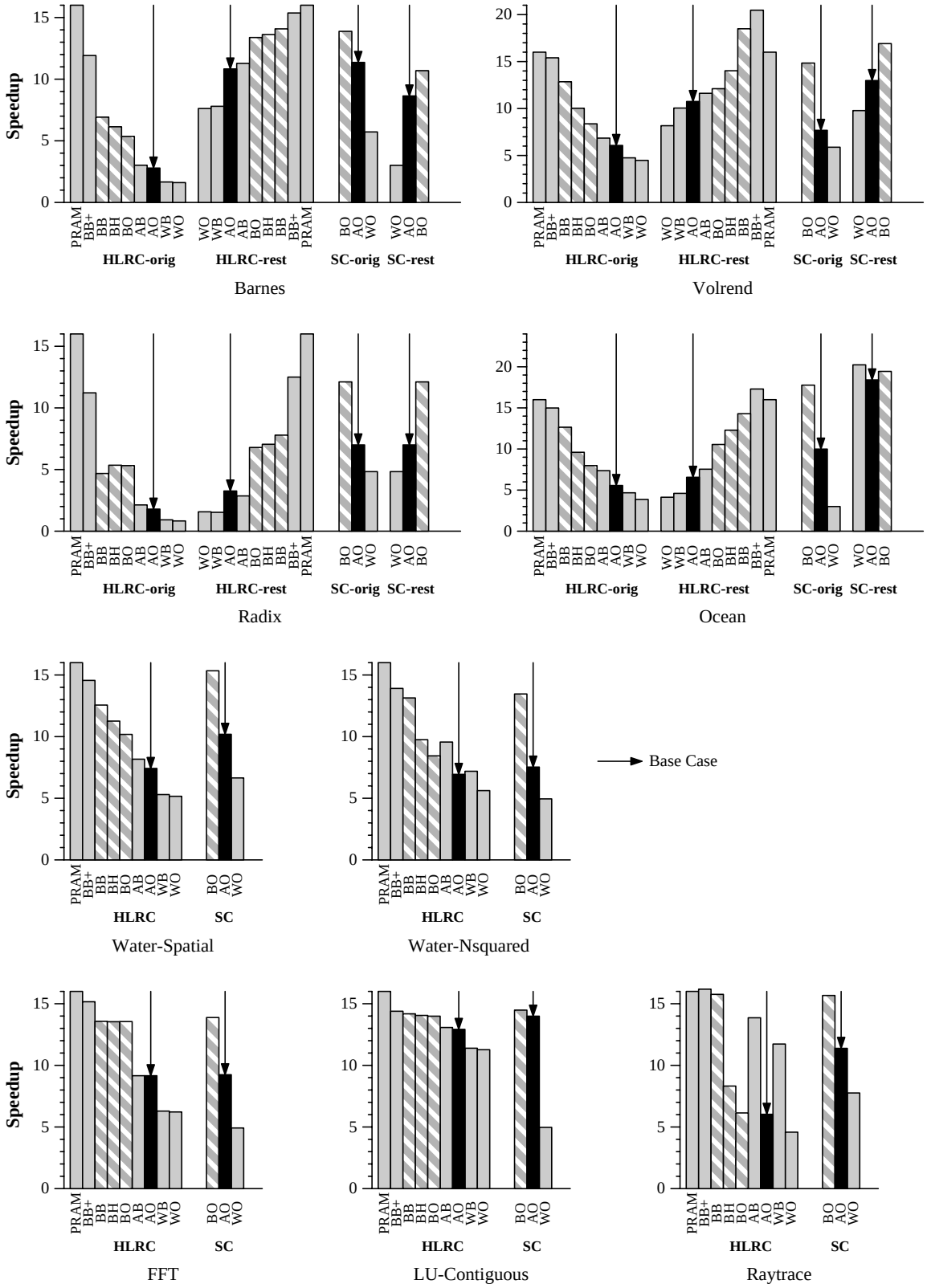

Figure 3: Application speedups. Please see text for how to interpret the bars and focus attention on some of them.

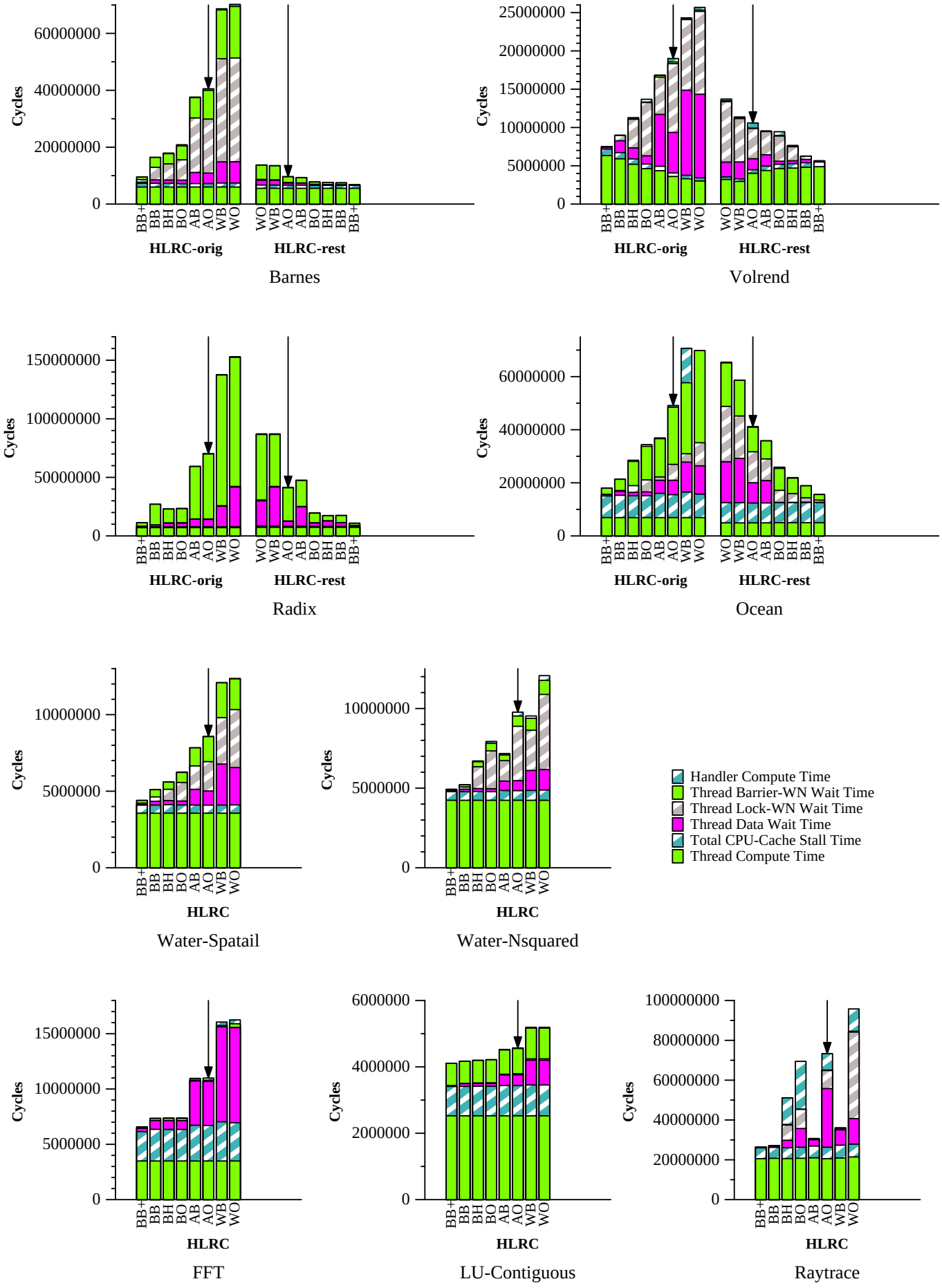

Figure 4: Application execution time breakdowns.

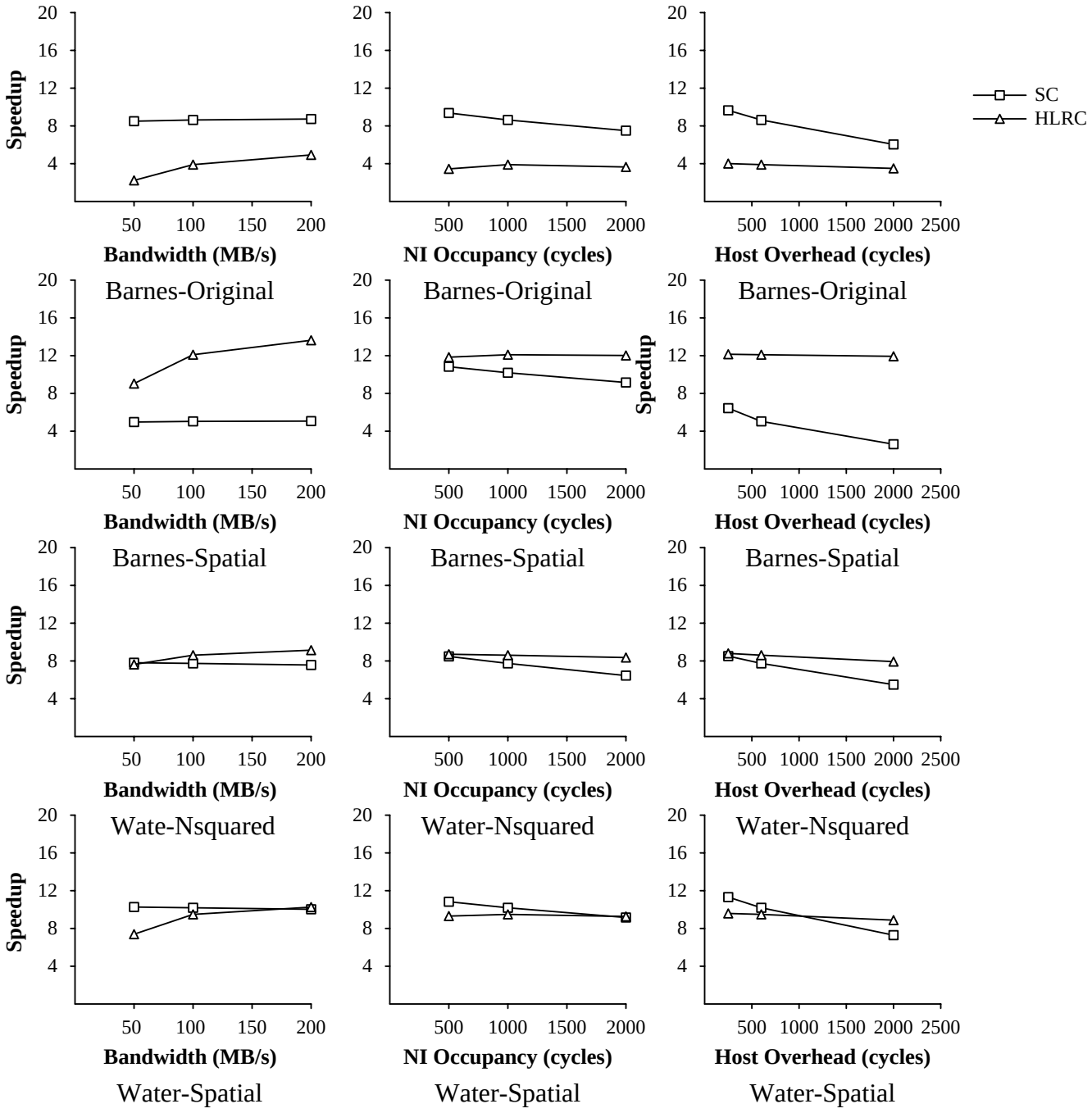

Figure 5: Speedups with communication architecture parameters. Each row is a different application, and each column is a different parameter. Only four applications are shown; for some of the more regular ones SC uses a coarse granularity too and the behavior of the two protocols is very similar. The curves with triangles are for HLRC, and with squares are for SC.