Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation

Yuanyuan Zhou, Liviu Iftode, Kai Li and Jaswinder Pal Singh

Computer Science Department
Princeton University
Princeton, NJ 08544
{yzhou,liv,li,jps}@cs.princeton.edu

Brian R. Toonen, Ioannis Schoinas, Mark D. Hill, and David A. Wood

Computer Sciences Department
University of Wisconsin, Madison
Madison, WI 53705
{toonen,schoinas,markhill,david}@cs.wisc.edu

Abstract

During the past few years, two main approaches have been taken to improve the performance of software shared memory implementations: relaxing consistency models and providing fine-grained access control. Their performance tradeoffs, however, are not well understood. This paper compares the performance of three consistency models across four coherence granularity sizes on a 16-node cluster of workstations. The main result is that no single combination of protocol, granularity, and mechanism to handle message arrival performs best for all applications. The experiments with five classes of applications show that the combination of the sequential consistency (SC) protocol, a polling mechanism, and very fine (64-byte) coherence granularity works quite well for most applications. As the coherence granularity increases, single-writer lazy release consistency (SW-LRC) and home-based lazy release consistency (HLRC) perform better than the SC protocol for applications with false sharing and coarse-grain synchronization. However, the absolute performance with a coarse coherence granularity depends upon whether the application causes significant internal fragmentation. The HLRC protocol works better than SW-LRC, when the size of coherence granularity increases to a typical page size (4,096 bytes). The polling mechanism is very crucial for the SC protocol and not crucial for SW-LRC and HLRC protocols for certain applications.

1 Introduction

There are two important issues in providing a coherent shared address space abstraction on a network of computers, consistency models and coherence granularity. Consistency models define how applications use the shared address space, whereas the degree of the relaxation of a consistency protocol and the granularity of coherence determine the efficiency of an implementation. This paper evaluates the performance tradeoffs of the combinations of three consistency models with four sizes of coherence granularity for software shared memory implementations on a real hardware platform.

The original shared virtual memory (SVM) proposal and prototype [19] uses the traditional virtual memory access protection mechanism to detect access misses and implements a sequential consistency model [16]. The main advantage of the approach is that it implements shared memory entirely in software on a network of commodity workstations [18] to run applications developed for hardware shared-memory multiprocessors. A disadvantage is that it restricts the coherence granularity to be a virtual memory page size. For systems with large page sizes, false sharing and fragmentation will occur in applications with multiple writer, fine-grained access patterns.

During the past few years, two main approaches have been taken to address this problem: relaxing consistency models and providing access control at a fine granularity. Relaxed consistency models introduce additional programmer restrictions in exchange for (hopefully) better performance. Examples of relaxed consistency models include release consistency [9], lazy release consistency [14], entry consistency [1], automatic update release consistency [10], scope consistency [12], home-based lazy release consistency [27], and single-writer lazy release consistency [15]. These consistency models employ sophisticated protocols to reduce false sharing and fragmentation.

An alternative approach is to preserve the simplicity of sequential consistency, but find some approach to reduce the coherence granularity. Examples of providing fine-grained access control include taking advantage architectural features such as the ECC bits to trap access faults [26], using software instrumentation for shared reads and writes [26, 25], and building special access control hardware for commodity workstations [22]. The finer the granularity, the less false sharing and fragmentation occur, and hence the less need to use relaxed models. A disadvantage of fine-grain coherence is that the smaller granularity may result in excessive misses and poor remote bandwidth.

To date, the performance tradeoffs between relaxed consistency models and coherence granularities have not been well studied. This paper attempts to understand these tradeoffs by conducting experiments on a real system. Our study focuses on coherent shared memory systems with a fixed coherence granularity. We focus on the following two questions:

- What is the best combination of granularity and consistency protocol for different classes of applications, and how much difference does it make?

- For applications that suffer performance losses in moving to coarser granularities under sequential consistency, can the performance be regained using sophisticated consistency protocols?

We conducted our experiments on a on 16 dual-processor Sun SPARCstation 20s interconnected by Myrinet and augmented with fine-grained access control hardware that supports multiple sizes of coherence granularity. We studied the combinations of three consistency protocols (sequential consistency (SC) [16], single-writer lazy release consistency (SW-LRC) [15] and home-based lazy release consistency (HLRC) [27]) with four sizes of coherence granularity. We also studied two mechanisms (polling and interrupt) to handle message arrivals for each case. Our experiments used five real applications developed for hardware shared memory systems: LU decomposition, Ocean, Water-Nsquared, Water-Spacial and Barnes. Each represents a particular combination of shared data access patterns and different synchronization methods.

The main result is that no single combination of protocol, granularity, and mechanism to handle message arrival performs best for all applications. Our results show that the combination of the SC protocol, the polling mechanism, and very fine (64-byte) coherence granularity works quite well for most applications. As the coherence granularity increases, the SW-LRC and HLRC protocols perform better than the SC protocol for applications with false sharing and coarse-grain synchronization. However, the absolute performance with a coarse coherence granularity depends upon whether the application causes significant internal fragmentation. The HLRC protocol tolerates internal fragmentation better than SW-LRC because it sends *diffs* rather than entire pages. The polling mechanism works better with fine coherence granularity and the interrupt mechanism works better with coarse coherence granularity. The polling mechanism is very crucial for the SC protocol and is not important for the SW-LRC and HLRC protocols.

2 Consistency Protocols

We studied three consistency protocols including sequential consistency (SC), single-writer lazy release consistency (SW-LRC), and multiple-writer, home-based lazy release consistency (HRLC). The main rationale of using these three models is twofold. First, the sequential consistency model is supported in many hardware shared memory multiprocessors. Second, the two relaxed consistency models are the latest proposed models that perform well for page-size coherence granularity.

All protocols examined in the paper support coherence granularity at block level (64, 256, 1,024, 4,096 bytes) and they all use virtual memory mapping mechanisms to allow data from a shared virtual address space to be transparently cached at a local physical address. Specifically, when a node touches a page for the first time a page fault is generated, which causes the page to be `mmap`-ed to local memory. Depending on how coherence in managed, additional actions may be necessary to initialize the page.

Each page has a *home*. Initially, pages are assigned home nodes round robin. After the beginning of an application's parallel phase, page homes migrate to the first node that "touches" them (sometimes called *first touch*). A "touch" is a load or a store for SC and store for HLRC. Regular application LU has "touch arrays" that explicitly touch data structures to manage data layout. When subsequent nodes touch an already-touched page, they go to the original home, find out about the new home, and thereafter remember the new home. Specifically, a page's home node ID is found in the distributed table and cached in a local table.

2.1 SC Protocol

The sequential consistency model allows each coherence unit to have either a single writer or one or more readers; readers and writers never co-exist at the same time. This model is generally considered the simplest for programmers, because, informally, a read always returns the result of the most recent write.

Our sequential consistency implementation is based on the Stache protocol [23] and is similar to many directory-based hardware implementations [17]. On a miss, a request message is sent to the designated *home* node. If invalidations are required, the home node collects the acknowledgments before forwarding the data to the requesting node. When an invalidation arrives at a node, the message is processed immediately (modulo the polling/interrupt issue); read-only copies are invalidated and read-write copies are written-back to the home node and invalidated.

2.2 SW-LRC Protocol

The single-writer lazy release consistency (SW-LRC) model allows a single-writer to co-exist with multiple readers and delays the propagations of updates to shared memory to the executions of acquire operations [15]. This model is more relaxed than the sequential consistency, but less relaxed than the multiple-writer lazy release consistency protocols.

Our SW-LRC protocol uses the same timestamp-based coherence control as proposed for Lazy Release Consistency [14] but it allows only a single writable copy to co-exist with multiple read-only copies. In this protocol, a write fault causes ownership to migrate but, unlike a sequential consistency protocol, read-only copies are not invalidated. Instead, SW-LRC blocks are invalidated at the acquire execution following the coherence information (**write notices**) sent with the lock. Shared blocks are version-ed each time the ownership changes. By including the versions in the write-notices and storing them away, the SW-LRC protocol can service a read fault in an one-hop roundtrip and can also avoid unnecessary invalidations.

2.3 HLRC Protocol

The Home-based Lazy Release Consistency (HLRC) protocol implements the well-known lazy release consistency model but has several performance and implementation advantages [27]. Both HLRC and traditional LRC protocols use the same multiple-writer solution based on using "twin" and "diff" but with different update schemes. Each writer is allowed to write into its copy once a clean version of the block (twin) has been created. Changes are detected by comparing the current (dirty) copy with the clean copy (twin) and recorded in a structure called a *diff*. Updates from one copy are transferred into another copy by diff-ing the first copy, sending the diff and applying it on the second copy.

The traditional LRC implementation uses a distributed diff scheme [13]. Diffs are created and sent on demand at the faulting block. To bring a copy up-to-date, diffs must be applied in the proper causal order determined using vector timestamps [14].

The HLRC multiple-writer scheme differs from LRC by having the diffs sent and applied eagerly a designated home of the block. With such a scheme the home's copy of the block is kept up-to-date and its whole content will be fetched on demand to update the other copies. Our implementation extends earlier work [27] by supporting various coherence granularities.

3 Testbed

This section describes the platform used for these experiments. The testbed consists of 16 dual-processor Sun SPARCStation 20s[1]. Each contains two 66 MHz Ross HyperSPARC processors [24]; however, our study only uses one processor on each node. Each processor has a 256 KB L2 cache and each node contains 64 MB of main memory. The cache-coherent 50 MHz MBus connects the processors and memory. I/O devices reside on the 25 MHz SBus, which connects to the MBus via a bridge. All nodes run Solaris 2.4.

Each node contains a Myrinet network interface [2], which consists of a 7-MIPS custom processor (LANai) and 128KB of memory. The LANai performs limited protocol processing and schedules DMA transfers between the network and LANai memory or LANai memory and SPARC memory. The 16 nodes used in this paper are connected with three Myrinet 8-port crossbar switches. Two ports of each switch are used to connect to other switches.

Each node also contains a Typhoon-0 card that *logically* performs fine-grain access control checks on all loads and stores by *physically* snooping memory bus transactions and exploiting inclusion [21]. When the Typhoon-0 hardware detects an access control violation, it generates an exception to the shared-memory run-time system via a special fast-exception method supported by the device driver (approximately 5 μs). All protocol processing occurs on the faulting processor. The Typhoon-0 card does not directly support messaging, but does accelerate polling for messages on the Myrinet by providing a cacheable location that indicates whether a message has arrived.

With Myrinet hardware, the host (SPARC) processor and Myrinet LANai processor cooperate to send and receive data. Our communication library is based on the LANai Control Program (LCP) used in Berkeley's LAM library [5]. The host processor uses loads and stores to move small messages and headers for large messages to and from LANai memory. The LANai processor uses DMA to directly move larger messages through intermediate kernel/user buffers. This organization lowers the latency of small messages and increases the throughput and the latency for large messages.

Message latency depends upon whether *polling* or *interrupts* are used to detect message reception. With polling, the LANai communicates with the Typhoon-0 board via a dedicated signal, which sets a cacheable memory location. Applications are instrumented via executable editing to check this message reception flag on all backedges. With interrupts, the LANai's hardware interrupt is translated by Solaris into a UNIX signal, which takes a minimum of 67 μs. Polling results in lower latency because signals are so slow. However, polling introduces needlessly overhead when no message is present. With polling, a microbenchmark shows 4-, 32-, 512-, and 4K-byte messages see round-trip times of 40, 57, 160, and 869 usecs. With interrupts the corresponding numbers are 267, 284, 499, and 1289 usecs. Large messages achieve bandwidths of about 17 and 15 MB/sec for polling and interrupts respectively, which are close to the values obtained by others [6, 20].

[1] *Note to reviewers: We actually have 40 machines and hope to have 32-node runs for the final version.*

4 Applications

To evaluate the performance of the three protocols with different sizes of coherence units, we used five applications, each representing a class of applications that have an access pattern to shared data with either fine-grained or coarse-grained synchronization. We classify the applications by several criteria:

- **Single writer vs. Multiple writer** Based on the number of concurrent writers on the same coherence unit we can divide the applications in single writer and multiple writer. Write-write false sharing occurs only for multiple writer applications.
- **Coarse-grain vs. Fine-grain data access** Data access granularity affects how the communication to computation ratio changes with the coherence granularity [11]. Applications with coarse-grain access tend to access a whole contiguous page at a time. Fine-grain applications are likely to scatter reads and writes across multiple pages. Fine-grain reads can introduce fragmentation with coarse coherence granularity and/or false sharing.
- **Coarse-grain vs. Fine-grain synchronization** The frequency of synchronization events is an important performance factor for shared virtual memory protocols. Usually applications with fine-grain synchronization perform small tasks and associate synchronization with the basic data unit(cell, molecule) thus generating significantly more synchronization events. On the other hand, applications with coarse-grain synchronization tend to execute large tasks and associate synchronization with the number of processors (queues).

Application	Problem Size	Sequential Execution Time (secs)
LU	1024×1024	73.41
Ocean-Contiguous	514×514	37.43
Water-Nsquared	4096 molecules, 3 steps	575.283
Water-Spatial	4096 molecules, 5steps	898.454
Barnes	16384 particles	33.787

Table 1: Benchmark applications, problem sizes, and sequential execution times.

We used five applications from the SPLASH-2 suite, including LU decomposition, Ocean, Water-Nsquared, Water-Spatial, and Barnes. Table 1 shows the problem sizes

Application	Concurrent Writer per Block	Spatial Access Granularity	Temporal Synch. Granularity	Number of Lock Acquires	Number of Barriers	Performance Level	Effect of Increasing Coherence Granularity
LU	single	coarse	coarse	0	64	all medium	all improve
Ocean-Contiguous	single	fine	coarse	72	328	all poor	SC declines; LRCs improve
Water-Nsquared	multiple	coarse	fine	588	12	all good	all improve slightly
Water-Spatial	multiple	fine	coarse	21	18	all good	SC declines; SW-LRC declines at 4K; HLRC stays constant
Barnes	multiple	fine	fine	SC: 2086 LRCs: 17167	8	SC medium; LRCs poor	SC declines; LRCs constant

Table 2: Classification and performance of sharing patterns and synchronization granularity in the applications.

for five applications and their sequential execution times. Table 2 summarizes the classification of sharing patterns and synchronization granularity in the applications.

LU performs the blocked LU factorization of a dense matrix. In this paper, we used a version which allocates each block contiguously in virtual memory and assigns contiguous blocks to each processor.

Ocean simulates eddy currents in an ocean basin. We focus first on the "contiguous" version in SPLASH-2 in which the data in each subgrid are allocated contiguously in virtual memory using a 4-d array. Writes are local but reads of border elements are nonlocal and fine-grained for column-oriented borders. Thus, there is little false sharing but significant fragmentation in the induced patterns.

Water-Nsquared simulates a system of water molecules in liquid state, using an $O(n^2)$ brute force method with a cutoff radius. The water molecules are allocated contiguously in an array of n molecules, and partitioned among processors into contiguous pieces of n/p molecules each. The challenging phase for SVM happens when each processor updates its own n/p molecules and the following $(n/2 - n/p)$ molecules of other processors in the array, using per-partition locks for mutual exclusion.

Water-Spatial solves the same problem as Water-Nsquared, but with different data structures and different algorithms. The 3-d physical space is broken up into cells, and every processor is assigned a contiguous cubical partition of cells together with the linked lists of molecules in them. A processor reads data from those processors that own cells on the boundary of its partition. After several time-steps, the molecules in a node's partition belong to separate pages thus exhibiting a fine-grain access pattern.

Barnes is an irregular application which simulates the interactions among a system of particles over a number of time steps, using the Barnes-Hut hierarchical N-body method. We first focus on the "rebuild" version in SPLASH-2 which builds the tree from scratch after each computation phase. Like Water-Spatial, in this application each node accesses cells which belong to different pages. Therefore, both read and write access patterns to the global particle and cell arrays are fine-grained.

5 Performance

We experimented the five applications with the combinations of three protocols (SC, SW-LRC, HLRC) with four block sizes (64, 128, 1,024 and 4,096 bytes) and two mechanisms to handle message arrivals (polling and interrupt). We will first present the overall application performance results and then analyze the performance difference for each application in detail.

5.1 Overall Performance

Figure 1 shows the speedup of each application nodes for all combinations of the three protocols and four block sizes. There are two graphs for each application: one using a polling mechanism to handle message arrivals and the other using a fast interrupt mechanism.

In our experiments, no single combination performs best for all applications. The SC protocol with 64-byte block size and polling mechanism has best performance for Ocean, Water-Spatial and Barnes, whereas the HLRC protocol with 4,096-byte block size and the interrupt mechanism has the best performance for LU and Water-Nsquared. Except for Barnes, the differences between the best combinations of SC and HLRC protocols are small.

The best granularity for the HLRC protocol is 4,096 bytes for all applications, whereas the best for the SC protocol is 64 bytes except for the two applications with coarse-grain accesses (LU and Water Nsquared). The best granularity for the SW-LRC protocol varies among applications: 64 bytes for Barnes, 256 byte for Ocean-Contiguous, 1,024 byte for Water-Spatial, and 4,096 byte for LU and Water-Nsquared.

For applications that suffer performance losses in moving to coarser granularity under sequential consistency, the performance can be regained using both SW-LRC and HLRC protocols effectively. For Ocean-Contiguous, both SW-LRC and HLRC protocols outperform the SC protocol slightly with 4,096-byte block size. For Water-Spatial, the effect is dramatic. With 4,096-byte block size, the HLRC protocol improves the SC protocol performance by more than a factor of two, whereas the SW-LRC protocol improves by about 40%.

In general, the polling method is better than the interrupt

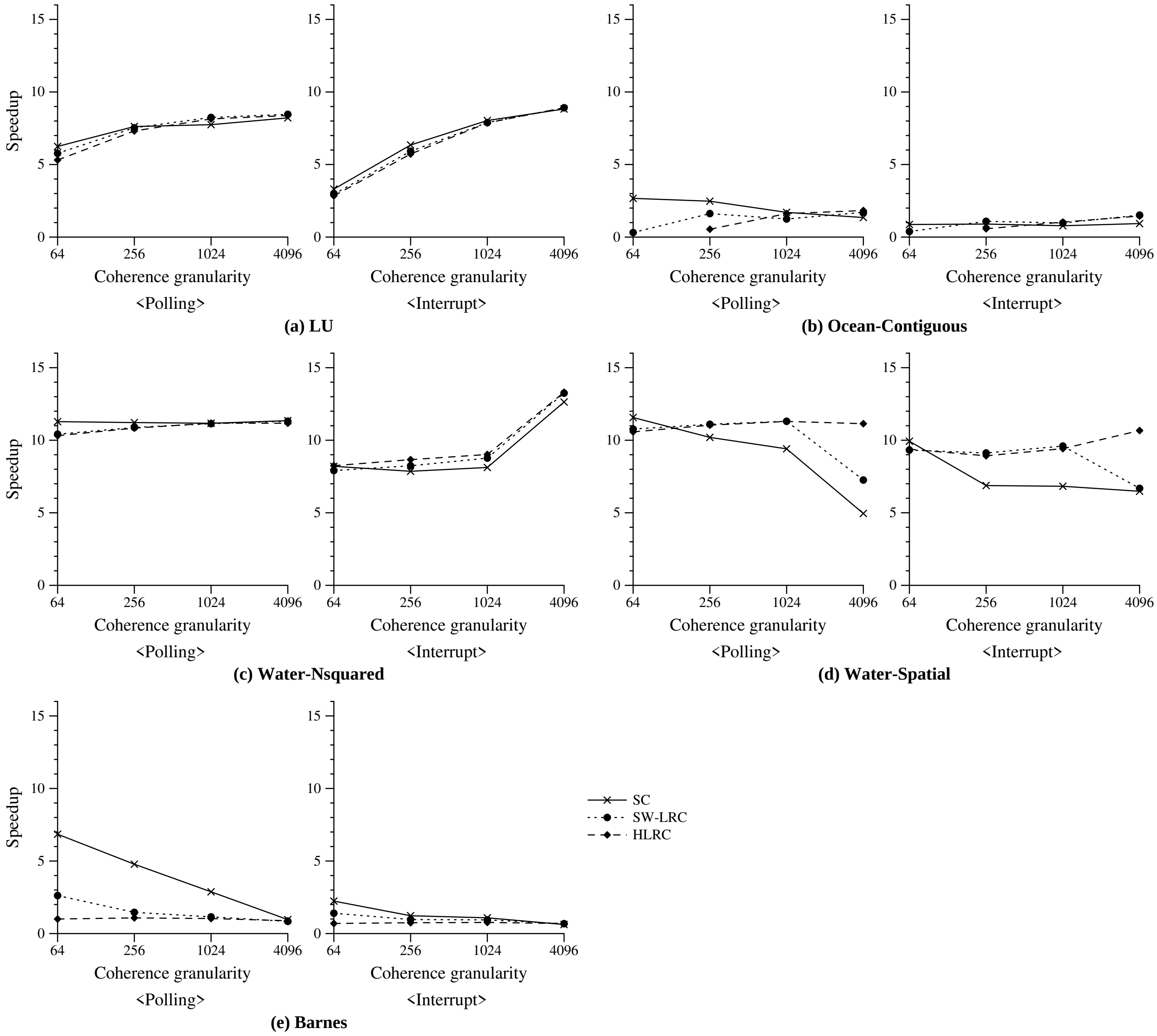


Figure 1: Speedups on T0 with 16 nodes.

method at a fine granularity and the interrupt method is better than the polling method at a coarse granularity but none of the three protocols performs consistently better with a single method for all granularities. Our results also show that the SC protocol is more sensitive to which method is used than the SW-LRC and HLRC protocols. For example, the speedup of the Water-Nsquared application using the SC protocol with the polling method at the 64-byte granularity is 11.3, a factor of 40% better than that with the interrupt method. The speedup of the same application using the HLRC protocol or the SW-LRC with the polling method at the same granularity is only about 25% better than that with the interrupt method.

5.2 Detailed Analysis

Since each of our five applications represents a particular combination of false sharing pattern, access granularity, and synchronization style, we will analyze the application performance individually.

SWCA: Single-writer with coarse-grain access

LU(Table 3) is a typical application in this category, with good spatial locality and only one writer for each shared page. It is expected that all the protocols performs better at coarse granularity because coarse coherence granularity has the positive effect of prefetching. The speedup curves and the frequency numbers validate this expectation. There is no fragmentation as long as the block sizes used to partition the work are larger than the coherence granularities. In the absence of false sharing and fragmentation, coarser coherence granularities can reduce the number of protocol operations, and thus reducing the protocol overheads. For all the protocols, the number of read misses decreased by around 4 times when the access granularity becomes 4 times larger.

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	24655	6297	1574	393
	SW-LRC	24655	6297	1574	393
	HLRC	24655	6297	1574	393
Write Fault	SC	0	0	0	0
	SW-LRC	0	0	0	0
	HLRC	0	0	0	0
Invalidation	SC	0	0	0	0
	SW-LRC	0	0	0	0
	HLRC	0	0	0	0
Data Message	SC	24655	6297	1574	393
	SW-LRC	24655	6297	1574	393
	HLRC	24655	6297	1574	393
Data Message Traffic (MByte)	SC	1.50	1.54	1.54	1.54
	SW-LRC	1.50	1.54	1.54	1.54
	HLRC	1.50	1.54	1.54	1.54
Control Message	SC	25912	7555	2831	1650
	SW-LRC	24715	6357	1634	453
	HLRC	24715	6357	1634	453
Control Message Traffic (MByte)	SC	0.19	0.05	0.02	0.01
	SW-LRC	0.38	0.10	0.03	0.01
	HLRC	3.02	0.78	0.20	0.06

Table 3: LU

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	31791	28121	26875	11069
	SW-LRC	30288	27310	25557	8886
	HLRC		27309	25608	8780
Write fault	SC	26631	22385	21439	8441
	SW-LRC	25244	19169	16815	5417
	HLRC		19425	16824	5438
Invalidation	SC	31703	28028	26752	10921
	SW-LRC	30323	27365	25551	8879
	HLRC		27341	25603	8770
Data Message	SC	32879	28498	27081	11254
	SW-LRC	55533	46479	42372	14304
	HLRC		27330	25631	8803
Data Message Traffic (MByte)	SC	2.01	6.96	26.45	43.96
	SW-LRC	3.39	11.32	41.25	55.31
	HLRC		6.65	24.91	33.88
Control Message	SC	97080	85585	81759	34744
	SW-LRC	57240	47977	43826	15312
	HLRC		28453	26574	9708
Control Message Traffic (MByte)	SC	0.49	0.43	0.41	0.17
	SW-LRC	3.89	3.06	2.73	0.98
	HLRC		4.57	3.67	1.27

Table 4: Ocean-Contiguous

As the result, the number of data messages and data traffic reduce with the same factor as the access granularity.

All the protocols perform almost the same at all granularities because all the protocols have the same number of protocol operations, and almost same amount of communication traffic. No write fault occurs in all protocols because there is only one writer for each page. For the same reason, the HLRC protocol does not perform any diff operation. Furthermore, in LU, a page is never read by any nodes before it is written. This means that the SC protocol performs no invalidation at any write times and neither the LRC protocols at any synchronization points.

SWFA: Single-writer with fine-grain access

Ocean-Contiguous (Table 4) is an example application in this class. Ocean is a grid-based application with fine-grain accesses resulting from the nearest-neighbor computation. A contiguous block allocation eliminates false-sharing ensuring a single writer for all pages which belong to the same block.

All the protocols performs poorly. The best speedup is 2.7 which is produced by the combination of the SC protocol with 64-byte block size and the polling mechanism. The main reason for this poor performance is the fragmentation due to the mismatch between the access granularity and coherence granularity. The mismatch is more pronounced at a coarse coherence granularity. An antagonist, positive effect, namely prefetching can reduce the negative effect of fragmentation at very large granularity. For example, with 4,096-byte block size, the additional data traffic introduced by fragmentation is around 99% of the total data traffic. With 64-byte block size, the fragmentation is still around 88%.

For the SC protocol, fine granularity can improve the performance of the protocols by reducing the extra data traffic caused by fragmentation. For the SW-LRC and HLRC protocols, fine granularities increase the protocol overhead, thus the overall performance decreases. With coarse granularities, the HLRC and SW-LRC protocols perform better than the SC protocol because they can hide the latency of writes. The SC protocol needs to wait for invalidation acknowledgments before the computation can resume. The LRC protocols have also 20% less access faults and 15% less invalidations due to the alleviation of the read-write false sharing caused by the relaxation of the consistency model.

The SW-LRC protocol performs best with 256-byte block size. This is because the communication fragmentation is more significant than write latency hiding. With larger block sizes, the SW-LRC and HLRC behave similarly because there is no write-write false sharing in this applications.

Due to limited physical memory on each node of our testbed, we were not able to execute the HLRC for 64-byte granularity and the execution time for 256-byte is affected by disk swapping.

MWCA: Multiple-writer with coarse-grain access

Water-Nsquared (Table 5) is an application with multiple writers per molecule and migratory access pattern. Since each process updates successively a large number of molecules the migratory pattern at molecule level is preserved at page level which leads to a coarse-grain access pattern and large prefetching effects.

With 4,096-byte block size, the HLRC protocol has fewer read misses and invalidations due to its multiple-writer scheme but the overhead of the diff operations and transfers

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	22184	15158	8644	2158
	SW-LRC	22125	13636	7073	1783
	HLRC	20490	12551	6538	1663
Write Fault	SC	9224	5610	2872	732
	SW-LRC	10856	6589	3370	851
	HLRC	11497	6926	3372	948
Invalidation	SC	16391	11634	6790	1685
	SW-LRC	18029	11077	5730	1447
	HLRC	16394	10020	5195	1327
Data Message	SC	28687	19324	10835	2711
	SW-LRC	32851	20215	10445	2634
	HLRC	28817	18015	9380	2775
Data Message Traffic (MByte)	SC	1.75	4.72	10.58	10.59
	SW-LRC	2.01	4.93	10.18	10.19
	HLRC	1.37	3.26	6.50	6.60
Control Message	SC	63282	44407	26239	7904
	SW-LRC	34462	21683	11834	3927
	HLRC	21987	13993	7864	2995
Control Message Traffic (MByte)	SC	0.35	0.24	0.14	0.04
	SW-LRC	2.13	1.46	0.93	0.52
	HLRC	3.51	2.31	1.25	0.65

Table 5: Water-Nsquared

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	34776	56462	43399	53758
	SW-LRC	34243	20277	11485	4447
	HLRC	34238	20157	11350	4321
Write Fault	SC	3310	31408	29453	34276
	SW-LRC	3189	2349	1686	35832
	HLRC	3189	2280	1486	919
Invalidation	SC	34359	51496	41699	39456
	SW-LRC	34243	20280	11490	4448
	HLRC	34238	20160	11351	4322
Data Message	SC	35481	57101	43892	54517
	SW-LRC	37433	22627	13171	40279
	HLRC	34258	20495	11692	4759
Data Message Traffic (MByte)	SC	2.17	13.94	42.86	212.96
	SW-LRC	2.28	5.52	11.52	152.05
	HLRC	2.09	4.94	9.98	12.50
Control Message	SC	105571	160627	127997	133880
	SW-LRC	37704	22806	13313	40392
	HLRC	34412	20289	11463	4424
Control message Traffic (MByte)	SC	0.53	0.83	0.66	0.72
	SW-LRC	0.97	0.64	0.40	0.73
	HLRC	4.39	2.62	1.49	0.60

Table 6: Water-Spatial

offsets some of the gains due to the relaxed consistency. Thus, the HLRC protocol performs 5-10% better than the SC protocol with coarse granularity and the interrupt mechanism. With the polling mechanism, the HLRC protocol performs similarly to the SW-LRC protocol for all block sizes, whereas the HLRC protocol performs 5% better than SW-LRC with the interrupt mechanism.

MWFACS: Multiple-writer with fine-grain access and coarse-grain synchronization

Water-Spatial (Table 6) exposes both write-write and read-write false sharing at coarse granularity. Its synchronization is coarse grained (only 21 lock acquires, and 18 barriers). This application illustrates the case for which relaxed consistency protocols at coarse granularity can amortize the cost of the extra protocol overhead. Both the HLRC protocol with 4,096-byte block size and the SC protocol with 64-byte block size perform well. The former combination performs well because relaxed consistency model reduces false sharing. The latter combination performs well false sharing is eliminated.

Both SW-LRC and HLRC protocols performs better than the SC protocol with large block sizes because they can alleviate the read-write false sharing by delaying the invalidations. The number of the SW-LRC protocol's read misses, for example, is only about 1/10 of that of the SC protocol's, while the number of invalidations is only about 1/9. On the other hand, the protocol overhead of either LRC protocol is higher than that of the SC protocol. The high protocol overhead offsets the the advantage of relaxed consistency for fine granularity such as 64 bytes. Since HLRC is a multiple-writer protocol, it hides write latency and reduces write misses. For example, it reduces the the number of write misses of the SW-LRC and SC protocols by a factor of 30 at coarse granularities. As a result, it performs 50% better than the SW-LRC protocol with 4,096-byte block size. When the block size is equal or less than 1,024 bytes, the number of write misses for both LRC protocols becomes the same, so SW-LRC's performance is very close to HLRC's. At fine granularity, all the protocols have the same number of access misses and invalidations. Since the relaxed protocols have more overhead, their performance is worse than the SC's performs by about 7 to 10%.

MWFAFS: Multiple-writer with fine-grain access and fine-grain synchronization

Barnes (Table 7) is an irregular application whose read-write accesses cause false sharing and fragmentation for almost all block sizes. For Barnes, using relaxed protocols like HLRC or SW-LRC requires adding more synchronization to the application in order to make it to comply with the release consistency model. These changes will increase the frequency of synchronizations significantly.

The question is whether it is worth to pay the additional protocol overhead to relax the consistency model and reduce false-sharing. It turns out that both protocols perform poorly compared to SC mainly due to the high frequency of fine-grain synchronization events. The cost of the 17,167 additional lock operations counts for more than 50% of the total execution time. Although the effect of the relaxation is significant: 4 times fewer read misses and 30% fewer write misses and 4 times fewer invalidations, the potential overhead reduction becomes negligible comparing with the cost of the synchronization events.

Neither SW-LRC protocol nor HLRC protocol can eliminate the false sharing and fragmentation in this application.

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	7024	10008	11709	14923
	SW-LRC	5280	5586	4451	3377
	HLRC	5184	5767	4388	3396
Write Fault	SC	2792	5631	5772	5949
	SW-LRC	2685	5484	5715	5853
	HLRC	6971	5565	4792	3997
Invalidation	SC	6749	9784	11529	14775
	SW-LRC	5255	5528	4436	3413
	HLRC	5114	5693	4372	3453
Data Message	SC	8603	15154	17362	21091
	SW-LRC	7965	11071	10167	9231
	HLRC	10996	10920	9003	7252
Data Message Traffic (MByte)	SC	0.53	3.70	16.96	82.39
	SW-LRC	0.49	2.70	9.93	36.04
	HLRC	0.46	1.63	4.52	13.76
Control Message	SC	28729	41288	46781	57760
	SW-LRC	22496	25776	24851	23868
	HLRC	19864	20427	19138	18241
Control Message Traffic (MByte)	SC	0.15	0.22	0.25	0.31
	SW-LRC	2.20	3.08	3.12	2.82
	HLRC	3.49	3.48	3.27	3.10

Table 7: Barnes

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	26679	7570	3159	5306
	SW-LRC	18961	5128	1668	781
	HLRC		5176	1653	759
Write Fault	SC	21018	5744	2289	3291
	SW-LRC	19163	5381	1965	2418
	HLRC		5377	1850	984
Invalidation	SC	21774	6076	2576	4262
	SW-LRC	18976	5138	1675	789
	HLRC		5179	1659	766
Data Message	SC	26691	7590	3184	5340
	SW-LRC	38124	10510	3634	3200
	HLRC		5453	1929	1061
Data Message Traffic (MByte)	SC	1.63	1.85	3.11	20.86
	SW-LRC	2.33	2.56	3.51	12.16
	HLRC		1.28	1.69	3.21
Control Message	SC	69744	20087	8794	12972
	SW-LRC	39715	11521	4589	4148
	HLRC		6149	2611	1718
Control Message Traffic (MByte)	SC	0.37	0.10	0.04	0.07
	SW-LRC	2.94	0.92	0.41	0.31
	HLRC		0.92	0.43	0.29

Table 8: Ocean-Noncontiguous

The data traffic of the HLRC with 4,096 block size is 25 times more than that of the SC with 64-byte block size. With 4,096 block size, the SW-LRC's data traffic is almost twice of the HLRC's, so SW-LRC performs worse than HLRC. As the block size decreases, SW-LRC becomes better than HLRC because of its lower protocol overhead.

5.3 Changing the Applications

We also modified two applications to move them from one class to another class to see how much better the performance can improve. Figure 2 shows the speedups of the modified version of Ocean and Barnes for all three protocols.

Moving from SWFA to SWCA

As mentioned above, all protocols perform poorly for Ocean-contiguous because its fine grain access pattern causes large (88-99%) fragmentation. We modified the noncontiguous implementation of Ocean from SPLASH-2 to partition the grid row-wise. This change moves the application to the class with single writer and coarse-grain access pattern. Comparing with the original Ocean-Contiguous version, the best speedup after the change increases from 2.7 to 10.0.

Although this change increases the ratio of inherent communication to computation, it reduces the ratio of induced communication to computation. As a result, the change significantly reduces the frequency of all operations including faults, data traffic and control traffic (See Table 8).

Moving from MWFAFS to MWFACS

We used the "partree" version of Barnes (Table 9). This version uses a new tree-building algorithm that each processor first constructs a partial tree and then everything is merged into a single global tree thus reducing the lock by a factor of 10 compared with the rebuild version. So we can consider this change as moving from fine-grained synchronization style to coarse-grain synchronization.

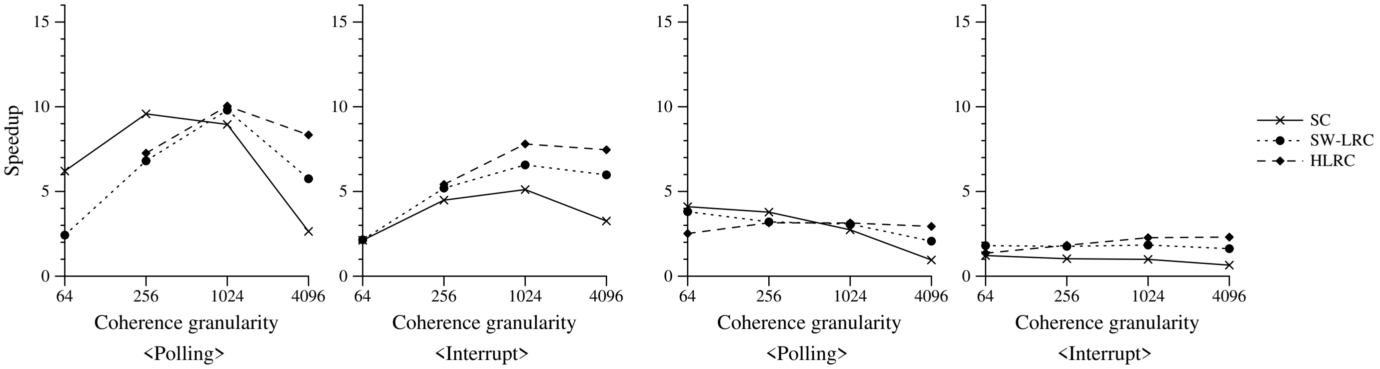
The HLRC protocol improves its speedup by a factor of more than 2 for all block sizes, though it has more access misses and invalidations with smaller block sizes. The SW-LRC protocol improves by about 15% with 64-byte coherence granularity, but a factor of more than 2 for 4,096-byte coherence granularity.

6 Related Work

There is a large body of literature in the area of distributed shared memory. The most related work to this paper includes research on relaxing consistency models and providing fine-grained coherence granularity for software coherent shared memory.

The original shared virtual memory (SVM) proposal and prototype [19] uses the traditional virtual memory access protection mechanism to detect access misses and implements the sequential consistency model [16] on a network of workstations. The coherence unit of the prototype is a 1,024-byte virtual memory page.

Since then, two main approaches have been taken to deal with the false-sharing and fragmentation problem in SVM systems: relaxing consistency models and providing fine-grained access control. Examples of relaxed consistency models include release consistency (RC) [9] and its SVM



(a) Ocean-Noncontiguous **(b) Barnes-Partree**

Figure 2: Speedups on T0 with 16 nodes for the two modified applications.

Operation or Message Traffic	Protocol	Coherence Granularity			
		64	256	1024	4096
Read Fault	SC	14532	13871	13511	16725
	SW-LRC	9498	7088	4566	1908
	HLRC	9773	8239	4690	1890
Write Fault	SC	5891	6804	6117	6269
	SW-LRC	5087	6947	6241	6189
	HLRC	8357	5143	3574	1844
Invalidation	SC	13873	13257	13000	16252
	SW-LRC	9568	7182	4597	1917
	HLRC	9792	8346	4716	1896
Data Message	SC	18695	19802	19097	22780
	SW-LRC	14586	14036	10808	8098
	HLRC	15874	12841	7954	3589
Data Message Traffic (MByte)	SC	1.14	4.83	18.65	88.98
	SW-LRC	0.89	3.43	10.55	31.60
	HLRC	0.75	2.22	4.79	7.56
Control Message	SC	52953	52564	51362	62037
	SW-LRC	15323	14842	11574	8729
	HLRC	10543	9043	5341	2510
Control Message Traffic (MByte)	SC	0.28	0.28	0.28	0.33
	SW-LRC	0.93	1.14	0.94	0.43
	HLRC	1.82	1.45	0.90	0.46

Table 9: Barnes-Partree

implementation [4], delayed consistency model [7] and its SVM protocols [3, 8], multiple-writer lazy release consistency (LRC) model [14] and implementation [13], entry consistency model and prototype [1], automatic update release consistency [10], scope consistency [12], home-based lazy release consistency and its implementations [27], and single-writer lazy release consistency [15]. The prototypes built based the relaxed consistency models all use virtual memory page sizes as their coherence units.

The other approach is to preserve the sequential consistency model and to find ways to reduce the coherence granularity. Examples of providing fine-grained access control include taking advantage architectural features such as the ECC bits to trap access faults [26], using software instrumentation for shared reads and writes [26, 25], and building special access control hardware for commodity workstations [22].

Although previous implementation or simulation studies have compared various protocols with various kinds of applications, none has studied the performance tradeoffs between relaxing consistency models and providing fine coherence granularity.

7 Conclusions

This paper investigates the performance tradeoffs of relaxing consistency models and coherence granularity. Our performance results with five kinds of applications show several results. The main result is that no single combination of protocol, granularity, and mechanism to handle message arrival performs best for all applications. Except for Barnes, the differences between the best combinations of SC and HLRC protocols are small.

The best granularity for the HLRC protocol is 4096 bytes for all applications, whereas for the SC protocol the best granularity is 64 bytes except for LU and Water N-squared. The best granularity for the SW-LRC protocol varies among applications.

For applications that suffer performance losses in moving to coarser granularity under sequential consistency, the performance can be regained with both SW-LRC and HLRC protocols significantly. For Water-Spatial, the improvements are 40% with SW-LRC and more than 100% with HLRC.

The performance advantage due to relaxing consistency model and supporting multiple-writer differs for different applications. Applications with false sharing but coarse-grain synchronization, like Water-Spatial, HLRC with 4K granularity can achieve the same performance as the SC does with 64 byte granularity. For fine-grain regular applications like Ocean-contiguous, the relaxed protocols (HLRC and SW-LRC) cannot achieve the same performance as SC can provide at 64 byte granularity because relaxation cannot alleviate fragmentation which is the dominant effect for these applications.

For applications with fine-grain synchronization like Barnes, HLRC and SW-LRC performs poorly at 4K granularity because more locks have to be added to the application to make it release consistent. As a result, the synchronization overhead dominates, thus reducing the impact of false sharing and write latency alleviation.

Our experimental results also show that the polling method works better for fine granularities and the interrupt method works better for coarse granularities and that none of the three protocols performs consistently better with a single method for all granularities.

Our study has several limitations. First, we have not studied block sizes greater than 4,096 bytes. It is not clear how the combinations in this paper behave for applications like LU, Water-Nsquared, Water-Spatial, since some combinations perform best with 4,096 block size. Second, although the applications used in the paper all have different access granularity, patterns and synchronization styles, we only experimented with an application in each class. Third, we have not compared memory utilization of different combinations.

As a final note to the reviewers, we could not complete the delayed consistency protocol before the submission deadline, but we expect to complete it and include our results in the final version of the paper.

References

[1] B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon. The Midway Distributed Shared Memory System. In *Proceedings of the IEEE COMPCON '93 Conference*, February 1993.

[2] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[3] L. Borrmann and M. Herdieckerhoff. A Coherency Model for Virtual Shared Memory. In *Proceedings of the 10th International Parallel Processing Symposium*, June 1990.

[4] J.B. Carter, J.K. Bennett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proceedings of the Thirteenth Symposium on Operating Systems Principles*, pages 152–164, October 1991.

[5] David Culler, Lok Tin Liu, Richard Martin, and Chad Yoshikawa. LogP Performance Assessment of Fast Network Interfaces. *IEEE Micro*, pages 35–43, February 1996.

[6] David Culler, Lok Tin Liu, Richard Martin, and Chad Yoshikawa. LogP Performance Assessment of Fast Network Interfaces. *IEEE Micro*, pages 35–43, February 1996.

[7] M. Dubois, J.C. Wang, L.A. Barroso, K. Lee, and Y-S Chen. Delayed Consistency and Its Effects on the Miss Rate of Parallel Programs. In *Supercomputing '91*, pages 197–206, 1991.

[8] A. Erlichson, N. Nuckolls, G. Chesson, and J. Hennessy. SoftFLASH: Analyzing the Performance of Clustered Distributed Virtual Shared Memory. In *The 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[9] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 15–26, May 1990.

[10] L. Iftode, C. Dubnicki, E. W. Felten, and Kai Li. Improving Release-Consistent Shared Virtual Memory using Automatic Update. In *The 2nd IEEE Symposium on High-Performance Computer Architecture*, February 1996.

[11] L. Iftode, J. P. Singh, and Kai Li. Understanding Application Performance on Shared Virtual Memory. In *Proceedings of the 23rd Annual Symposium on Computer Architecture*, May 1996.

[12] L. Iftode, J.P. Singh, and K. Li. Scope Consistency: a Bridge Between Release Consistency and Entry Consistency. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1996.

[13] P. Keleher, A.L. 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, January 1994.

[14] P. Keleher, A.L. 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.

[15] P.J. Keleher. The Relative Importance of Concurrent Writers and Weak Consistency Models. In *Proceedings of the IEEE COMPCON '96 Conference*, February 1996.

[16] L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocessor Programs. *IEEE Transactions on Computers*, C-28(9):690–691, 1979.

[17] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica Lam. The Stanford DASH Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[18] K. Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume II Software, pages 94–101, August 1988.

[19] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pages 229–239, August 1986.

[20] Scott Pakin, Mario Laura, and Andrew Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing '95*, 1995.

[21] Robert W. Pfile. Typhoon-Zero Implementation: The Vortex Module. Technical report, Wisconsin University, CS department, 1995.

[22] S. K. Reinhard, R. W. Pfile, and D. A. Wood. Decoupled Hardware Support for Distributed Shared Memory. In *Proceedings of the 23rd Annual Symposium on Computer Architecture*, May 1996.

[23] S.K. Reinhardt, J.R. Larus, and D.A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21st Annual Symposium on Computer Architecture*, pages 325–336, April 1994.

[24] ROSS Technology, Inc. *SPARC RISC User's Guide: hyperSPARC Edition*, September 1993.

[25] D.J. Scales, K. Gharachorloo, and C.A. 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*, October 1996.

[26] I. Schoinas, B. Falsafi, A.R. Lebeck, S.K. Reinhardt, J.R. Larus, and D.A. Wood. Fine-grain Access for Distributed Shared Memory. In *The 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 297–306, October 1994.

[27] 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*, October 1996.