A New Home-Based Software DSM Protocol for SMP Clusters*

Weiwu Hu, Fuxin Zhang, and Haiming Liu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing 100080
hww@water.chpc.ict.ac.cn

Abstract. This paper introduces an SMP protocol for the home-based software DSM system JIAJIA. In the protocol, intra-node processes in an SMP node share their home pages through hardware coherent sharing so as to take the full advantage of the *home effect* of home-based software DSMs. In contrast, cached remote pages of a process are not shared by its intra-node partners to avoid cache page conflict within an SMP. Besides, JIAJIA also implements the shared memory communication among processes within the same SMP node to accelerate intra-node communication. Performance evaluation with some well accepted benchmarks and real applications in a cluster of four two-processor nodes shows that the SMP protocol of JIAJIA reduces remote accesses, diffs, and consequently message amounts in all of the ten benchmarks and as a result obtains noticeable performance improvement in seven.

1 Introduction

With the wide spread of symmetric multiprocessor (SMP) systems, cluster of SMPs has been emerging as an attractive parallel processing platform to provide high performance with certain connectivity and affordable costs. Given the convenient shared address programming model of SMPs, it is natural to extend this convenience to the SMP clusters with shared virtual memory. The shared virtual memory has the obvious advantages over the message passing alternative on SMP clusters in that it can take the advantage of the efficient hardware-based shared memory within an SMP. In a software DSM on cluster of SMPs, the SMP hardware transparently provides shared memory at the cache line granularity for intra-node sharing, while the software protocol is responsible for providing shared memory at the page granularity for inter-node sharing.

Previous software DSMs on SMP clusters include the softFLASH system[3] which implements a single-writer protocol for sequential consistency, the Shasta system[12] which performs coherence through code instrumentation at cache-line granularity, the Cashmere-2L[13] which is based on the DEC Memory Channel

* The work of this paper is supported by National Natural Science Foundation of China (Grant No. 69703002) and National High Technology (863) Program (Grant No. 863-306-ZD01-02-2).

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 1132–1142, 2000.

network interface and network, the HLRC-SMP[11] which is an implementation of a lazy, home-based, multiple-writer protocol across SMP nodes and which uses the Virtual Memory Mapped Communication (VMMC-2) library for the Myrinet network, and the modified version of TreadMarks[7] which uses POSIX threads instead of processes to implement parallelism within a multiprocessor.

This paper introduces design and implementation of a new software DSM protocol for SMP clusters. The protocol is designed and implemented on the home-based software DSM system JIAJIA[4] and hence has a similar goal to the HLRC-SMP protocol. The main difference between HLRC-SMP and the SMP protocol of JIAJIA is that processes in an SMP share both home and cached pages in HLRC-SMP, while only home pages are shared by intra-node processes in JIAJIA. Sharing home pages among intra-node processes in SMP clusters helps to take the full advantage of the *home effect* of home-based software DSMs, i.e., the ability to dispense with page faults for references made by the home node of a given page. Though sharing cached pages is helpful for all processes within a node to benefit from a page fetch performed by one, it also causes cache page conflict such as when a process is writing a cached page, another process in the same node may covers this page due to a page fetch.

Another optimization JIAJIA makes for SMP clusters is the optimization of intra-node communication. With this optimization, processes within an SMP node communicate through shared memory.

The effect of the SMP protocol is evaluated on a Myrinet cluster of four two-processor Ultra-2 nodes with ten benchmarks, include Water, Barnes, Ocean, and LU from SPLASH2, MG and 3DFFT from NAS Parallel Benchmarks, SOR, and TSP from TreadMarks benchmarks, and two real applications EM3D for magnetic field computation and IAP18 for climate simulation. Evaluation results show that the SMP optimization achieves a speedup of $4\% - 5\%$ in LU, SOR, and EM3D, about 10% in MG and 3DFFT, and more than 20% in Ocean and IAP18.

The rest of this paper is organized as follows. The following Section 2 briefly introduces the JIAJIA software DSM system. Section 3 illustrates the SMP protocol of JIAJIA. Section 4 presents experiment results and analysis. The conclusion of this paper is drawn in Section 5.

2 The JIAJIA Software DSM System

In JIAJIA, each shared page has a home node and homes of shared pages are distributed across all nodes. References to home pages hit locally, references to non-home pages cause these pages to be fetched from their home and cached locally. A cached page may be in one of three states: Invalid (INV), Read-Only (RO), and Read-Write (RW). When the number of locally cached pages is larger than the maximum number allowed, some aged cache pages must be replaced to its home to make room for the new page. This allows JIAJIA to support shared memory that is larger than physical memory of one machine.

JIAJIA implements the scope memory consistency model. Multiple writer technique is employed to reduce false sharing. In JIAJIA, the coherence of cached pages is maintained through requiring the lock-releasing (or barrier arriving) processor to send to the lock write notices about modified pages in the associated critical section, and the lock-acquiring (or barrier leaving) processor to invalidate cached pages that are notified as obsolete by the associated write-notices in the lock. This protocol maintains coherence through write notices kept on the lock and consequently eliminates the requirement of directory.

The following optimizations[4,6] are made to the protocol.

Single-Writer Detection. In the optimization, if a page is modified only by its home processor during a synchronization interval, then it is unnecessary to send the associated write notice to the lock manager at the end of the interval. If a page is modified only by one remote processor during an interval, then the processor who makes the modification need not invalidate the page on next acquire or barrier.

Incarnation Number[8] Technique. With this optimization, each lock is associated with an incarnation number which is incremented when the lock is transferred. A processor records the current incarnation number of a lock on an acquire of the lock. When the processor acquires the lock again, it tells the lock manager its current incarnation number of the lock. With this knowledge, the lock manager knows which write notices have been sent to the acquiring processor on previous lock grants and excludes them from write notices sent back to the acquiring processor at this time.

Lazy Home Page Write Detection. Normally, home pages are write-protected at the beginning of a synchronization interval so that writes to home pages can be detected through page faults. The lazy home page write detection delays home page write-protecting until the page is first fetched in the interval so that home pages that are not cached by remote processors do not need to be write-protected.

Write Vector Technique. The write vector optimization is motivated by the idea of fetching diffs only on a page fault in homeless protocols. It avoids fetching the whole page on a page fault through dividing a page into blocks and fetches only those blocks that are dirty with respect to the faulting processor on a page fault. A write vector table is maintained for each shared page in its home to record for each processor which block(s) has been modified since the processor fetched the page last time.

Home Migration. The home migration optimization adaptively migrates home of a page to the processor most frequently writes to the page, to reduce diff overhead at the end of an interval because write to home pages does not produces twin and diff in home-based protocol. In the home migration scheme, pages that are written by only one processor between two barriers are recognized by the barrier manager and their homes are migrated to the single writing processor. Migration information is piggybacked on barrier messages and no additional communication is required for the migration.

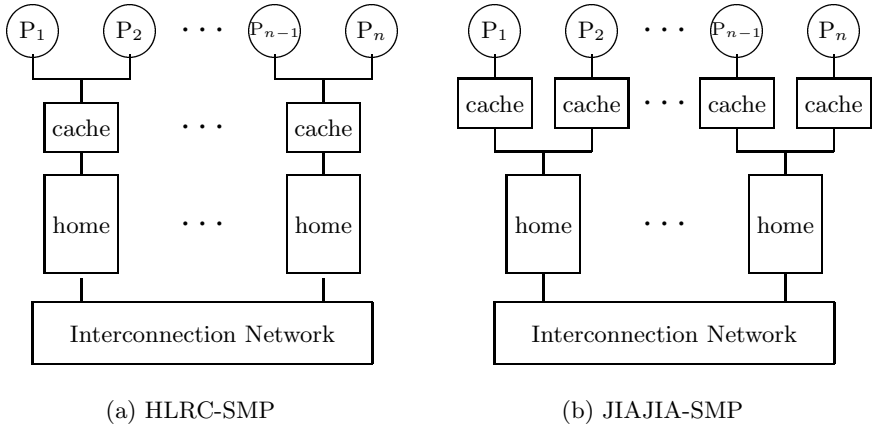

Fig. 1. Memory Organization of HLRC-SMP and JIAJIA-SMP

3 SMP Protocol for JIAJIA

3.1 Design Alternatives

Processors within an SMP can share data either through threads or through processes. The thread model provides an intuitive and efficient method of sharing data within an SMP. However, threads may cause some unexpected side effects. First, all threads within an SMP implicitly share all global variables, which makes it difficult to provide a uniform sharing memory between threads within a node and threads in different nodes. Second, all threads within an SMP have the same view of shared data, which means if a thread decides to invalidate a shared page, the page is invalidated for other threads as well.

The alternative is to use processes within an SMP node to share memory. One approach is to use `shmget()` and `shmat()` system calls. The major disadvantage of this approach is that both the number of shared segments and segment size are limited by the system, preventing it from being used by JIAJIA which can support a large shared memory. Besides, the system overhead of `shmget()` increases linearly with the number of segments[2].

Another and more efficient approach of sharing memory among processes with an SMP is to rely on virtual memory management and the `mmap()` system call (anonymous mapping with `MAP_SHARED` parameter). JIAJIA adopts this approach after comparing the portability, programmability, ability to support large memory, and implementation simplicity of all candidate alternatives. The disadvantage of this approach is that shared pages should be mapped before processes are forked in an SMP node. To meet this requirement, JIAJIA reserves a shared space before processes are forked in an SMP node at the initialization stage.

3.2 SMP Protocol

Figure 1 shows the memory organization of HLRC-SMP and SMP protocol of JIAJIA. As can be seen from Figure 1, the main difference between HLRC-SMP and the SMP protocol of JIAJIA is that processes in an SMP share both home and cached pages in HLRC-SMP, while only home pages are shared by processes in an SMP in JIAJIA.

Home-based software DSMs benefit from the *home effect*, the ability to dispense with page faults for references made by the home node of a given page. In SMP clusters, there are several processes running on an SMP node, each process is the home of part of the total shared pages. Sharing home pages among processes within a node combine their home pages together so that each process in the node "sees" a larger home.

Sharing cached pages within an SMP seems not so attractive to JIAJIA. Though with shared cache a processor may benefit from a remote page fetch carried out by another processor in the same SMP node, shared cache pages cause sharing violation among processors within the same SMP. For example, when a process fetches a page from its home, it has to ensure that no other processes in the same node are currently writing to the page, otherwise the incoming page may overwrite these writes. Besides, to support shared space larger than the physical memory of a machine, a cached page is dynamically mapped and unmapped to the faulting address when the page is cached and flushed by a process in JIAJIA. The dynamic mapping and unmapping of cached pages violates the requirement that pages physically shared by intra-node processes through `mmap()` should be mapped before intra-node processes are forked.

The decision of shared home and separated cache for intra-node processes greatly simplifies the SMP protocol of JIAJIA. Each processor in an SMP maintains cache coherence as if in cluster of single-processor nodes. Sharing home pages among intra-node processes is very simple and requires least modification to JIAJIA. Though all processes share their home pages, each process has its separate view (such as protect state) of a given home page and operates on the page as if it is the unique home host of the page. To service remote page fetch and diff request, one process is appointed as the *real home host* of a given page and a page fetch or diff request is always sent to the real home host of the faulting page. Real home hosts of shared pages in an SMP node are distributed across all processes within the node to avoid bottleneck problem. The process that regards a page as its home page, but does not service remote request about the page, are called *co-home host* of the page.

Though the SMP protocol of JIAJIA is simplified in the absence of some intra-node processes cache sharing related complexity, non-trivial issues still arise when coordinating with some optimization methods of JIAJIA.

In the lazy home page write detection, the home page write-protecting that are performed at the beginning of an interval is delayed until the page is first fetched in the interval so that home pages that are not cached by remote processors do not need to be write-protected. In the SMP protocol, a page fetch request is only sent to the real home host of the page, co-home hosts do not

know when the page is first fetched at an interval. Therefor, the lazy home page write detection technique is applied only to the real home host of a given page, and the co-home hosts of a page write-protect the page at the beginning of each interval as normal. With the observation that only the process that modifies a page needs to detect the write, the real home host of a given page can be dynamically shifted to the process that write the page most frequently.

In the write vector technique, a home page is divided into blocks and a write vector table is maintained for each shared page in its home to record for each processor which block(s) has been modified since the processor fetched the page last time, so that only modified blocks are sent back when the processor requires the page. In the normal protocol, updates by remote processors to a page are recorded in the write vector table when diffs are applied, and updates by the home node are recorded with the twin mechanism. In the SMP protocl, the modifications made by the co-home hosts of a page should also be known by the real home host. JIAJIA solves this problem through making the write vector table of any given page shared among all processes within a node, so that modifications to the write vector table made by co-home hosts are always visible to the real home host.

The home migration technique also causes problem to the SMP protocol of JIAJIA. As has been mentioned, the `mmap()` mechanism JIAJIA uses to share physical memory among intra-node processes requires that shared pages are mapped before intra-node processes are forked. In contrast, the home migration algorithm of JIAJIA maps and unmaps pages dynamically during runtime. Compromise has to be made here that if the home of a shared page is migrated from one node to another, then the advantage of sharing home page among intra-node processes is given up for this page. The new home host is the real home host of the page, no co-home host exists for this page.

3.3 Intra-node Communication

Another optimization JIAJIA makes for SMP clusters is the optimization of intra-node communication. In the old version of JIAJIA which regards processors of an SMP as separate nodes, UDP protocol is used for communication between two intra-node processes. With the SMP communication support, processes within an SMP node communicate through shared memory.

To support shared memory intra-node communication, JIAJIA allocates a shared communication buffer for each pair of intra-node processes. When a process wants to send a message to another process within the same node, the sending process first copies the message to the associated shared buffer. It then sends a SIGIO signal to the receiving process through `kill()` system call. On receiving the SIGIO signal, the receiving process directly reads the message from the shared communication buffer and then sets a tag in the buffer to indicate that the message has been read out. The sending process proceeds on observing this receiving tag.

To keep the simplicity of implementation, the current version of JIAJIA does not include some other SMP-related optimizations such as intra-node synchro-

Table 1. Characteristics and Sequential Run time of Benchmarks

Appl.	Size	Mem.	Barrs	Locks	Seq. Time(seconds)
Water	1728 mole.	0.5MB	70	1040	491.99
Barnes	16384 bodies	1.6MB	28	64	321.83
Ocean	$514 * 514$	60MB	858	1568	54.88
LU	$2048 * 2048$	32MB	128	0	133.45
SOR	$2048 * 2048$	16MB	200	0	89.44
TSP	`-f20 -r15`	0.8MB	0	1167	216.70
MG	$256 * 256 * 256$	443MB	592	0	398.56*
3DFFT	$128 * 128 * 128$	96MB	12	0	74.05
EM3D	$120 * 60 * 416$	160MB	20	0	101.46
IAP18	$144 * 91 * 18$	20MB	5400	0	1994.95

*: Estimated from $128 \times 128 \times 128$ MG sequential time (49.82 seconds). Sequential time of $256 \times 256 \times 256$ MG is unavailable due to memory size limitation.

Table 2. Runtime Statistics of Parallel Execution

Appl.	Msg. amt.(MB)		Get pages		Diffs	
	JIA	JIA_{smp}	JIA	JIA_{smp}	JIA	JIA_{smp}
Water	36	29	3420	2870	1897	1373
Barnes	81	65	9076	7408	510	353
Ocean	891	563	107073	67733	1294	1132
LU	25	18	12072	8292	0	0
SOR	12	5.3	2823	1216	0	0
TSP	7.4	6.3	4965	4237	3236	2780
MG	553	260	32508	18025	43192	17016
3DFFT	149	129	17976	15408	56	48
EM3D	2.4	1.1	1200	528	990	420
IAP18	2943	1355	276643	136474	131136	49920

nization and locating fault pages in the caches of intra-node processes on page faults.

4 Performance Evaluation

The evaluation is done in four Ultra-2 nodes connected by a Myrinet. Each node has two 200MHz Ultra-sparc processor and 256MB memory.

The benchmarks include Water, Barnes, Ocean and LU from SPLASH2[14], MG and 3DFFT from NAS Parallel Benchmarks[1], SOR and TSP from TreadMarks benchmarks[10], and two real applications EM3D for magnetic field computation and IAP18 for climate simulation[5]. Table 1 gives characteristics and sequential run time of these benchmarks.

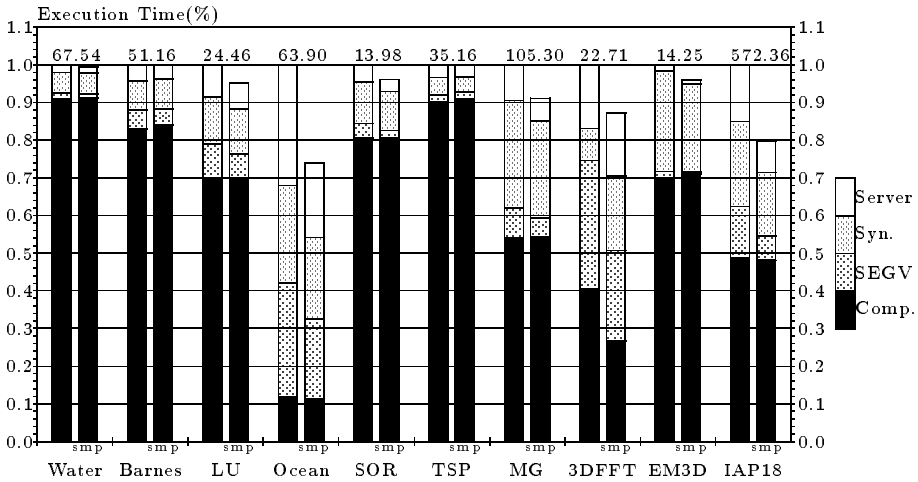

Fig. 2. Parallel Execution Time Breakdown

Each benchmark is run under JIAJIA both with (JIA_{smp}) and without (JIA) SMP optimization. The four nodes of two-processor SMPs are configured as eight separate hosts in JIA, and as four SMPs in JIA_{smp}.

Table 2 gives some statistics of the parallel execution. Message amount, remote get page request count, and diff count are listed in Table 2 for each run of each benchmark. Figure 2 shows parallel execution time results of the execution. For each benchmark, the parallel execution time of JIA rides on the top of the corresponding bar, and the parallel execution time of JIA_{smp} is represented as percentages of the parallel execution time of JIA. In Figure 2, the execution time of each parallel run is broken down into four parts: page fault (SIGSEGV service) time, synchronization time, remote request (SIGIO) service time, and computation time. The first three parts of time are collected at runtime as the overhead of the execution, and the computation time is calculated as the difference of the total execution time and the total overhead.

It can be seen from Table 2 and Figure 2 that, all of the ten benchmarks have less remote accesses, diffs, and consequently message amounts in JIA_{smp} than in JIA and seven of them achieve noticeable speedups with the SMP protocol.

Figure 2 shows that JIA_{smp} achieves significant speedup over JIA in Ocean (26.0%), MG (8.9%), 3DFFT (12.9%), and IAP18 (20.3%). The speedup of Ocean turns from negative in JIA to positive in JIA_{smp}. The common property of these four benchmarks is that their parallel speedup are not high. The eight-processor speedup of JIA is negative in Ocean, and is less than four in MG, 3DFFT, and IAP18. Therefore there is large room for performance improvement. Besides, the absolute number of remote accesses, diffs, and consequently message amounts reduced by the SMP protocol is great in Ocean, MG, and IAP18. Table 2 shows that JIA_{smp} causes 300MB less message amounts than JIA in Ocean

and MG, and 1.6GB less message amounts than JIA in IAP18. As a result of the great reduction of page faults, both the SIGSEGV time and server time are significantly reduced, as can be observed in Figure 2. The synchronization time of Ocean, MG, and IAP18 are also reduced due to the reduction of the number of diffs (diffs are generated at synchronization points). In FFT, though the message amount reduced by the SMP protocol is not as significant as the above three benchmarks, JIA_{smp} still reduces SIGSEGV time significantly. It can also be seen from Figure 2 that a great part of performance gains of JIA_{smp} over JIA in FFT is caused by the reduction of computation time in JIA_{smp}. This fact implies that FFT has better utilization of processor pipeline, cache, and TLB in JIA_{smp} than in JIA.

Figure 2 also shows that LU, SOR, and EM3D obtain moderate performance benefit (around 5%) from the SMP protocol. It can be seen from Table 2 that, though JIA_{smp} has only 43%, 44%, and 69% remote accesses of that required by JIA in SOR, EM3D, and LU, the absolute message amounts reduced are not significant. Besides, the performance of these three benchmarks is already high in JIA (the eight-processor speedup is 5.46 in LU, 6.4 in SOR, and 7.12 in EM3D) and the room for performance improvement is not large. Again, the execution time breakdown in Figure 2 shows that in LU, SOR, and EM3D, the performance gains of JIA_{smp} over JIA is mainly caused by the reduction of SIGSEGV time and server time as a result of remote accesses reduction. In EM3D, the synchronization time is also reduced in JIA_{smp} because JIA_{smp} generates much less diffs than JIA.

In Water, Barnes, and TSP, though the number of remote accesses and diffs is reduced with the SMP protocol, both the reduction range and the absolute message amounts reduced are not large. Besides, in these three benchmarks, performance is already high in JIA (the eight-processor speedups is 7.28 in Water, 6.29 in Barnes, and 6.16 in TSP) and the bottleneck for further performance improvement does not lie in the communication. Rather, synchronization is the major obstacle for further performance improvement in these three benchmarks. It can be seen from Table 1 and Figure 2 that lock is the major synchronization mechanism and synchronization overhead constitutes the major overhead in these three benchmarks. This implies that the lock waiting time dominates the overhead and (with the high bandwidth Myrinet) the moderate remote accesses reduction caused by JIA_{smp} has little influence on the performance of Water, Barnes, and TSP.

5 Conclusion and Future Work

This paper introduces the SMP protocol for the JIAJIA software DSM system. In the protocol, processes in the same SMP node keep their separate caches but combine their home pages together so that each process in the node "sees" a larger home.

Evaluation results show that compared to the JIAJIA without SMP optimization, the SMP protocol achieves significant speedups in seven out of ten

benchmarks. Quantitatively, the SMP protocol achieves a speedup of $4\% - 5\%$ in LU, SOR, and EM3D, about 10% in MG and 3DFFT, and more than 20% in Ocean and IAP18. Water, Barnes, and TSP do not benefit noticeably from the SMP protocol. It is expected that the SMP protocol of JIAJIA can perform better if there are more processors (e.g., four) within an SMP node.

It can also be learned from the above performance evaluation that communication bandwidth is critical to the performance of software DSMs such as JIAJIA. In the evaluation, the high bandwidth Myrinet makes JIAJIA not sensitive to the moderate reduction of remote accesses in Water, Barnes, and TSP. Compared to our previous evaluation results[4,6] on clusters connected by 100Mbps Ethernet, higher speedups are obtained in the cluster connected by Myrinet.

Our recent work about JIAJIA include further improving the coherence protocol of JIAJIA, supporting JIAJIA with faster communication mechanism such as special remote access hardware, implementing fault tolerance mechanism such as checkpoint in JIAJIA, building a pre-compiler to convert OpenMP programs to the API of JIAJIA, and porting more real applications on JIAJIA. Further information about JIAJIA is available at `www.ict.ac.cn/chpc/index.html`.

References

1. D. Bailey, J. Barton, T. Lasinski, and H. Simon, "The NAS Parallel Benchmarks", *Technical Report 103863*, NASA, Jul. 1993.
2. S. Dwarkadas, N. Hardavellas, L. Kontothanassis, R. Nikhil, and R. Stets, "Cashmere-VLM: Remote Memory Paging for Software Distributed Shared Memory", in *Proc. of the 13th Int'l Parallel Processing Sym.*, pp. 153–159, April. 1999.
3. A. Erlichson, N. Nuckolls, G. Chesson, and J. Hennessy, "SoftFLASH: Analyzing the Performance of Clustered Distributed Virtual Shared Memory", in *Proc. of the 996 Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.
4. W. Hu, W. Shi, and Z. Tang, "Reducing System Overhead in Home-Based Software DSMs", in *Proc. of the 13th Int'l Parallel Processing Symp.*, pp. 167–173, Apr. 1999.
5. W. Hu, F. Zhang, L. Ren, W. Shi, and Z. Tang, "Running Real Applications on Software DSMs", in *Proc. of the 2000 Int'l Conf. on High Performance Computing in the Asia-Pacific Region*, May 2000.
6. W. Hu, "Reducing Message Overheads in Home-Based Software DSMs", in *Proc. of the 1st Workshop on Software Distributed Shared Memory*, pp. 7–11. June 1999.
7. Y. Hu, H. Lu, A. Cox, and W. Zwaenepoel, "OpenMP for Networks of SMPs", in *Proc. of the 13th Int'l Parallel Processing Symp.*, pp. 302–310. Apr., 1999.
8. L. Iftode, "Home-based Shared Virtual Memory", *Ph. D. Thesis*, Princeton University, Aug. 1998.
9. P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepoel, "TreadMarks Distributed Shared Memory on Standard Workstations and Operating Systems", in *Proc. of the 1994 Winter Usenix Conf.*, pp. 115–131, Jan. 1994.
10. H. Lu, S. Dwarkadas, A. Cox, and W. Zwaenepoel, "Quantifying the Performance Differences Between PVM and TreadMarks", *Journal of Parallel and Distributed Computing*, Vol. 43, No. 2, pp. 65–78, Jun. 1997.

11. R. Samanta, A. Bilas, L. Iftode, and J. Singh, "Home-based SVM Protocols for SMP Clusters: Design and Performance", in *Proc. of the 4th Int'l Symp. on High Performance Computer Architecture*, Feb., 1998.
12. D. Scales, K. Gharachorloo, and A. Aggarwal, "Fine-grain Software Distributed Shared Memory on SMP Clusters", in *Proc. of the 4th Int'l Symp. on High Performance Computer Architecture*, pp. 125–136, Feb., 1998.
13. R. Stets et al., "Cashmere-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network", in *Proc. of the 1997 ACM Symp. on Operating Systems Principles*, pp. 170–183, Oct. 1997.
14. S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations", in *Proc. of ISCA'95*, pp. 24–36, 1995.