ViSMI: Software Distributed Shared Memory for InfiniBand Clusters

Chistian Osendorfer[1], Jie Tao[2], Carsten Trinitis[1], and Martin Mairandres[1]

[1]Lehrstuhl für Rechnertechnik und Rechnerorganisation
Technische Universität München
Boltzmannstr.3, 85748 Garching, Germany
{osendorfer, trinitic, mairandr}@cs.tum.edu

[2]Institut für Rechnerentwurf und Fehlertoleranz
Universität Karlsruhe
Kaiserstr.12, 76128 Karlsruhe, Germany
tao@ira.uka.de

Abstract

This paper describes ViSMI, a software distributed shared memory system for cluster systems connected via InfiniBand. ViSMI implements a kind of home-based lazy release consistency protocol, which uses a multiple-writer coherence scheme to alleviate the traffic introduced by false sharing. For further performance gain, InfiniBand features and optimized page invalidation mechanisms are applied in order to reduce synchronization overhead. First experimental results show that ViSMI introduces good performance comparable to similar software DSMs.

1. Introduction

Parallel architectures develop in two main directions: Tightly-coupled systems with processors connected via a local bus and loosely-coupled machines with processors interconnected through a high-speed local area network. The former architecture maintains a hardware-based global shared memory and thereby allows for programming the machines using parallel models semantically similar to serial programming languages but is restricted to small systems. The latter, in contrast, enables to build larger, scalable systems but usually deploys explicit communication paradigms in source code.

The advantages of both categories could be combined forming parallel platforms with scalability and cost-efficiency, which at the same time can use shared memory models. This requires to establish distributed shared memory (DSM) on top of cluster systems, which supports a global memory abstraction visible to all processor nodes in the cluster. This can be implemented by hardware, but mostly has to rely on software methodology.

This paper describes a software-based distributed shared memory called ViSMI (Virtual Shared Memory for InfiniBand clusters). A critical issue concerning DSM are the consistency protocols that guarantee coherence between replicas of shared data. Among those various schemes, Home-based Lazy Release Consistency (HLRC) uses a multiple-writer scheme to alleviate the traffic caused by false sharing. ViSMI implements such an HLRC protocol and establishes a virtual global memory space for shared memory programming. ViSMI is built on top of a cluster connected via InfiniBand [5], a high-performance interconnect technology. Besides its low latency and high bandwidth, InfiniBand supports Remote Data Memory Access (RDMA), allowing to access remote memory locations via the network without any involvement of the receiver.

This work is based on an existing implementation of HLRC [14], which has been developed on top of the Virtual Interface Architecture (VIA) [1], a user-level memory-mapped communication model with low overhead that excludes the operating system kernel from the communication path. We have replaced the VIA-related operations within this HLRC implementation with the corresponding functions provided by VAPI [12], a VIA-like communication interface for InfiniBand. In addition, we have extended HLRC to allow for optimizations for both system and programmers.

We make the following contributions:

- We implement ViSMI, an implementation of the HLRC protocol on clusters connected via InfiniBand, a reliable interconnection technology that supports interprocess communication at the network hardware level.
- We develop mechanisms for full use of InfiniBand fea-

tures, e.g. RDMA and hardware supported multicast, and mechanisms for less inter-node traffic.

- We evaluate ViSMI with standard benchmark applications and compare the performance with similar work in this area. Initial experimental results have shown the feasibility of ViSMI.

Currently, an SPMD model is supported, which applies ANL-like M4 macros to express parallelism in applications. We are working on OpenMP [3], a standardized, portable shared memory programming model, and expecting to enable the parallel execution of OpenMP programs on ViSMI in the near future.

The remainder of this paper is structured as follows: Sections 2 and 3 introduce the InfiniBand network architecture and the target system. This is followed by a brief description of consistency models on software distributed shared memory systems and related work in Section 4. Section 5 gives details on ViSMI, including implementation and optimization techniques. This is followed by the initial experimental results in Section 6. The paper concludes with a short summary and some future directions in Section 7.

2. InfiniBand Network Architecture

InfiniBand [5] is a point-to-point, switched I/O interconnect architecture with low latency and high bandwidth. Its first specification was completed in 2000 by an association of larger and smaller IT companies. This architecture defines a System Area Network (SAN) that connects processing nodes and I/O nodes. Processing nodes are attached to the network via Host Channel Adapters (HCA), while I/O nodes are connected to the fabric via Target Channel Adapters (TCA). Channel adapters are either connected directly or through switches. The basic InfiniBand connections are serial at data rates of 2.5 Gbps with low latency. Four or twelve of these serial point to point connections can be bundled, creating 4X connections at 10 Gbps or 12X connections at 30 Gbps, respectively. With its high bandwidth and low latency, the InfiniBand architecture is an outstanding interconnect for high performance clusters.

For communication operations, the InfiniBand architecture provides both channel and memory semantics. While the former refers to traditional send/receive operations, the latter allows the user to directly read or write data elements from or to the virtual memory space of a remote node without involving the remote host processor. This scenario is generally referred to as Remote Direct Memory Access (RDMA). Semantics of various operations are defined via InfiniBand *verbs*, and the Mellanox implementation of the InfiniBand verbs API, called VAPI [12], is usually deployed to support both the classic communication model and the RDMA operations.

3. Reference Cluster

For the evaluation of the work described in this paper, the InfiniBand research cluster [1] installed at Technische Universität München (TUM) has been used (see Figure 1). The configuration data of its key components are listed in Table 1.

The cluster has 6 Xeon nodes which are used partly for interactive tasks and partly for computation. Each of the nodes has two 32-Bit Intel Xeon DP processors running at 2.4 GHz and 4 GBytes of main memory. The mainboard of the nodes is a Super P4DPi–G2 from SuperMicro Computer Co. with an Intel E7500 chipset and two Gigabit Ethernet ports. Each node has a local IDE disk for booting and local files. In addition, file systems of the nodes are mounted on all other nodes in order to provide a common home directory and other common files. An MTPB23108 Host Channel Adapter card from Mellanox Technologies is plugged into the 133 MHz PCI-X slot of the mainboard and provides two 4X InfiniBand ports.

In addition, the cluster has 4 Itanium 2 (Madison) nodes which are primarily used for computation. Each of these nodes has four 64-Bit Intel Itanium 2 processors running at 1.3 GHz and 8 GBytes of main memory. The chipset on the Itanium boards is an Intel 8870. Each Itanium 2 node has a fast local SCSI disk for booting and local file storage, and a Gigabit Ethernet port. As for the Xeon based nodes, a Mellanox MTPB23108 Host Channel Adapter card is connected via a 133 MHz PCI-X slot of the mainboard and provides two 4X InfiniBand ports.

An InfiniBand switch MTEK43132 from Mellanox Technologies is the core of the InfiniBand fabric. The switch which currently has 24 ports can be upgraded to 96 ports. All InfiniBand connections are 4X with a theoretical peak bandwidth of 10 Gbps.

A file server with a RAID disk system provides a common home directory and other common files. A dedicated login node serves as a gateway to the University Ethernet network. The six Xeon nodes, the four Itanium2 nodes, the file server, and the login node are connected through a private Ethernet (which is currently limited by a 100 Mbit/sec Ethernet switch).

The operating system on all nodes is Red Hat Linux release 7.3 currently running kernel version 2.4.21. The compilers available are gcc version 2.96, PGI Fortran release 4.0-3, and Intel C++ and Fortran 7.0. All cluster–wide administrative data is maintained via the Lightweight Directory Access Protocol (LDAP) [2].

Currently, we have established a message passing environment on top of the cluster, and also designed an infrastructure for shared memory programming. As a first step

[1] http://infiniband.cs.tum.edu/

[2] http://www.ldap.org/

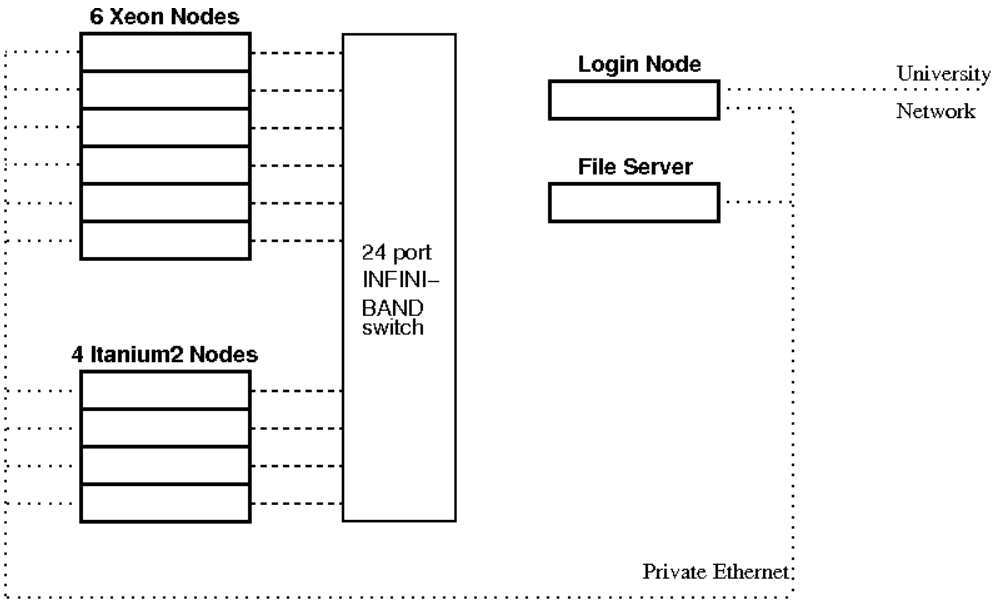

Figure 1. The InfiniBand cluster setup at TUM in a lab environment.

	IA-32 Nodes	IA-64 Nodes
Number of nodes	6	4
Processor	Intel Xeon DP 2.4 GHz	Intel Itanium2 QP 1.3 GHz
Number of Processors per node	2	4
L1 data cache size (line size)	8 Kbyte (32 bytes)	16 Kbyte (64 bytes)
L2 cache size (line size)	512 Kbyte (32 bytes)	256 Kbyte (128 bytes)
L3 cache size (line size)	N/A	3 Mbyte (128 bytes)
Processor system bus	64 bit, 400 MHz data rate	128 bit, 400 MHz data rate
Main memory	4096 MBytes	8192 MBytes
Memory bus	128 bit, 200 MHz data rate	256 bit, 200 MHz data rate
Chipset	Intel E7500	Intel 8870
Disk subsystem	single IDE disk	single SCSI disk
Network	4X INFINIBAND & FE	4X INFINIBAND & FE
LINUX kernel version	2.4.21 SMP i686	2.4.21 SMP ia64
MPI software	MPICH	MPICH

Table 1. Configuration data of both types of nodes

towards this infrastructure, we have implemented ViSMI, our software-based distributed shared memory system. Details of this implementation will be presented in Section 5.

4. Software Distributed Shared Memory

The basic idea behind software DSMs is to provide programmers with a virtually global address space on cluster architectures. This idea was first proposed by Kai Li [10] and implemented in IVY [11]. As memory is actually distributed across the cluster, the required data could be located on a remote node, and multiple copies of shared data could exist. The latter leads to consistency issues, where a write operation on shared data must be visible to other processors. In order to deal with this problem, software DSMs usually rely on the page fault handler of the operating system to support page protection mechanisms that implement invalidation-based consistency models.

The concept of memory consistency models is to precisely characterize the behavior of the respective memory system by clearly defining the order in which memory operations are performed. This has lead to a large amount of work in this area, and as a result various consistency models have been implemented using both hardware and software approaches. These include the strict Sequential Consistency and several relaxed consistency models.

Sequential Consistency was first defined by Lamport [9] as: "A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its programmers". This provides an intuitive and easy-to-follow memory behavior, however, the strict ordering requires the memory system to propagate updates early and prohibits optimizations in both hardware and compilers. Hence, other models have been proposed to relax the constraints of Sequential Consistency with the goal of improving the overall performance.

Relaxed consistency models [2, 4, 7, 8] define a memory model for programmers to use explicit synchronization. Synchronizing memory accesses are divided into *Acquires* and *Releases*, where an *Aquire* allows the access to shared data and ensures that the data is up-to-date, while *Release* relinquishes this access right and ensures that all memory updates have been properly propagated. By separating the synchronization in this way, invalidations are only performed by a synchronization operation, therefore reducing the unnecessary invalidations caused by an early coherence operation.

A well-known relaxed consistency model is Lazy Release Consistency (LRC) [8] in which the invalidations are propagated at the acquisition. This allows to perform any communication of write updates only when the data is actually needed. To reduce the communications caused by false sharing where multiple unrelated shared data locate on the same page, LRC protocols usually support a multiple-writer scheme. Within this scheme, multiple writable copies of the same page are allowed, and a clean copy is generated after an invalidation. Home-based Lazy Release Consistency (HLRC) [14], for example, implements such a multiple-writer scheme by specifying a home for each page. All updates to a page are propagated to the home node at synchronization points such as lock release and barrier. Hence, the page copy on home is up-to-date.

In the area of software distributed shared memory, a significant amount of work has been carried out since it was first proposed: The related system with Sequential Consistency was implemented in [11]. Quarks [16] is an example using relaxed consistency. This is a system aiming at providing a lean implementation of DSM to avoid complex high-overhead protocols. Treadmarks [7] was the first implementation of shared virtual memory using the LRC protocol on a network of stock computers. Shasta [15] is another example that uses LRC to achieve a distributed shared memory on clusters. For the HLRC protocol, representative work includes Rangarajan and Iftode's HLRC-VIA [14] for a GigaNet VIA-based network and VIACOMM [6], which implements a multithreaded HLRC also over VIA. We choose the former as the basis of a software DSM for InfiniBand clusters due to its simplicity and the similar communication abstractions between VIA and InfiniBand.

When we finished this work, we found out that the Network-Based Computing research group at Ohio State University had implemented an InfiniBand-based software DSM called NEWGENDSM [13]. It is also built on top of HLRC-VIA, and with InfiniBand features considered. In Section 6 we compare its performance to that of ViSMI.

5. ViSMI: Implementing a software DSM for InfiniBand

ViSMI implements a Home-based Lazy Release Consistency protocol on top of our InfiniBand cluster. As mentioned before, this work is based on an existing implementation of HLRC over VIA [14], the HLRC-VIA. Besides the modification relating to the different interface specification of both VIA and InfiniBand, we have taken into account the specific feature of InfiniBand for benefiting from its hardware support in interprocess communication. In addition, we have implemented optimizations in order to further alleviate the inter-node traffic and the overhead caused by page fault handling.

Overview. The protocol is actually based on a request/reply model, where requests are issued synchronously and received asynchronously. Hence, each node main-

tains an additional thread, besides the application thread, to handle the incoming communication. This communication thread is only active when a request occurs. We use the event notification scheme of InfiniBand to achieve this.

Another important issue with HLRC is to propagate the updates to a page that could have multiple writable copies. Within ViSMI, this is implemented using a *diff*-based mechanism, where the difference (*diffs*) between each dirty copy and the clean copy, which is created before the first write, is computed and propagated using hardware-based multicast provided by InfiniBand.

HLRC Implementation. The goal of a software DSM is to realize a shared virtual space accessible to all processors on a cluster. This shared space is organized at page granularity and allocated at the runtime. HLRC manages this space by specifying a home node for each shared page, and all page requests and runtime updates have to be sent to the home node. Figure 2 illustrates this relationship.

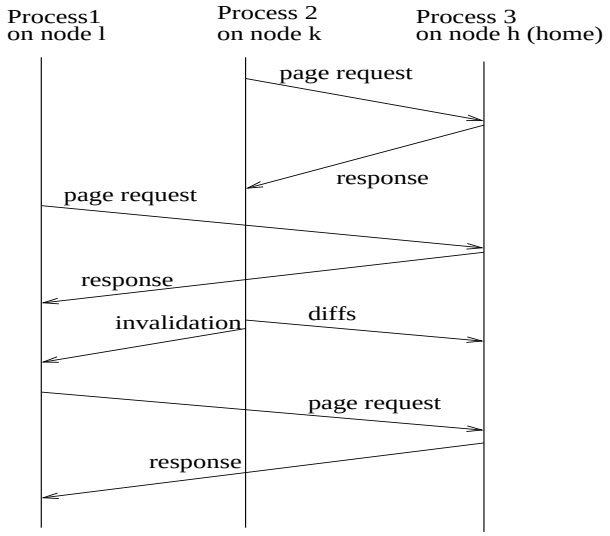

Figure 2. Communication between processes.

To access a remote page, an application sends a request to its home and waits for an acknowledgment that also contains the data. Both request and acknowledgment are sent using RDMA Write operations. After that, the application is allowed to modify the page copy located on the local node. This continues until a synchronization point (lock release, barrier) is reached, where the updates are sent to their home node to form a clean page, and all dirty copies have to be invalidated. For this, each node stores the modifications between two synchronization events. For further accesses to the page, a new page request is required.

For page management, each node maintains a data structure for storing information about a shared page, e.g status and the home node. A negative status on the home node indicates that the page is invalid and a page fault signal is issued. The result is the merging of all updates to the page.

Optimizations. As described above, HLRC defines an invalidation policy, where page updates are merged on the home node and all other copies have to be invalidated. As this results to new page requests and transfer of the whole page, we implemented an optimized scheme, in which updates are sent to all nodes and each node writes the updates to its local copy. In this way, the local copies need not be invalidated prohibiting the requirements to fetch the same page again. This can lead to a significant reduction of the overhead caused by frequent page fetching.

Programming Model. ViSMI currently supports an SPMD model for parallelizing sequential codes. This model applies the ANL-like M4 macros, those used in the SPLASH-II Benchmarks suite [17], to define operations with respect to parallel execution, such as environment initialization, process creation, synchronization, and memory allocation. To implement this model, all these macros are substituted with the corresponding functions provided by our HLRC implementation.

6. Performance Evaluation

ViSMI has been evaluated on our InfiniBand cluster using a standard benchmark suite. We measured both the speedup and the execution time breakdown, and compared these experimental results with HLRC-VIA [14], the basis of this work, and NEWGENDSM[13], the sole implementation of software DSM on InfiniBand clusters.

Experimental Setup. The hardware environment is the cluster described in Section 3. Since ViSMI is currently based on a 32-bit address space, only the six nodes with 32 bit Xeon DP processors could be applied in the experiments. However, for some applications, e.g. FFT, the number of processor has to be a power of 2. Hence, we used 4 processors to perform the experiments.

Benchmark Applications. We use the SPLASH-2 Benchmarks suite [17] to verify the efficiency of ViSMI. The applications chosen are Barnes, FFT, LU, and Radix. They represent the different access pattern of applications. A short description, the working set size, and the shared memory size of these applications are shown in Table 2.

Experimental Results. First, we measured both sequential and parallel execution time. We then calculated speedup and efficiency, where efficiency is obtained by dividing the speedup by the number of processors. Since the other systems to be compared use a different number of processors to perform the experiments, it is not possible to directly compare the speedup. Table 3 illustrates the results.

This table contains two blocks of data. The first block shows the performance of applications with ViSMI, and the

Applications	Description	Working set size	Shared memory size
Barnes	N-body problem	32768 bodies	40MB
FFT	Fast Fourier Transformations	2**20 data points	49MB
LU	LU-decomposition for dense matrices	2048×2048 matrix	33MB
Radix	Integer radix sort	8M keys	66MB

Table 2. Description of selected applications.

	ViSMI Performance			Comparison of Scaling Efficiency		
	Serial time	Parallel time	Speedup	ViSMI	HLRC-VIA	NEWGENDSM
Barnes	5.54s	1.68s	3.3	0.82	0.788	0.806
FFT	2.78s	1.07s	2.6	0.65	0.725	–
LU	315.12s	80.8s	3.9	0.975	0.925	–
Radix	2.22	1.01s	2.2	0.55	0.538	0.288

Table 3. Execution time, speedup, and efficiency of applications.

second block shows the comparison of efficiency with the other two systems. In terms of performance of ViSMI, it can be observed that the speedup varies between applications.

Examining the concrete data, it can be seen that LU performs better than the other applications. This is caused by its feature of intensive computation and coarse-grain parallelism. Radix behaves poorly due to its fine-grain access pattern. However, the same behavior has also been presented with other systems, where NEWGENDSM shows an efficiency of only 0.288 with radix. Other performance data in the second block of Table 3 also shows that ViSMI behaves so well as or even better than the other DSMs [3].

Overall, the various behavior of applications is caused by their diverse parallel nature and thereby resulted different number of page fetching and overhead of synchronization operations. This can be observed in Figure 3 which shows the normalized breakdown of the execution time.

In Figure 3, *Computation* denotes the time for actually executing the application, *Page Fetch* denotes the time for fetching pages, *Lock* denotes the time for performing locks, *Barrier* denotes the time for barrier operations, *Handler* denotes the time needed by the communication thread, and *Overhead* denotes the time for other protocol activities. It can be seen that LU shows the highest proportion in computation time, hence the best speedup that has been seen in Table 3.

Figure 3 also shows a general case where page fetching and barrier operations introduce most overheads. This could be improved by appropriate home assignment and optimized barrier. While the former aims at placing the pages on the node that dominantly accesses them to reduce the requirement for page fetching, the latter has the goal of decreasing the time needed by processes to wait for the completion of others by barriers. Currently, we are working on these optimizations, and a significant performance gain for the applications with poorer behavior is expected.

[3] NEWGENDSM has only data about Barnes and Radix

7. Conclusions

Over the last years, shared memory models have been increasingly used for programming cluster systems. A prerequisite for this, however, is a shared memory abstraction visible to all processor nodes in the cluster. In order to investigate shared memory computing on InfiniBand-based clusters, we have implemented such a distributed shared memory called ViSMI. ViSMI implements a home-based lazy release consistency protocol and is built on top of an existing system with adaptation to the InfiniBand architecture and optimizations.

ViSMI is the first step towards our goal: efficient cluster computing based on InfiniBand. On top of ViSMI, an execution environment for OpenMP will be established in the next phase of this research work. In addition, the current version of ViSMI will be extended. This includes to enable 64-bit address space as well as further optimizations with respect to data locality and barrier operations.

References

[1] Compaq Corporation, Intel Corporation and Microsoft Corporation. Virtual Interface Architecture Specification, 1997.

[2] A. L. Cox, S. Dwarkadas, P. J. Keleher, H. Lu, R. Rajamony, and W. Zwaenepoel. Software Versus Hardware Shared-Memory Implementation: A Case Study. In *Proceedings of the 21th Annual International Symposium on Computer Architecture*, pages 106–117, April 1994.

[3] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science & Engineering*, 5(1):46–55, January 1998.

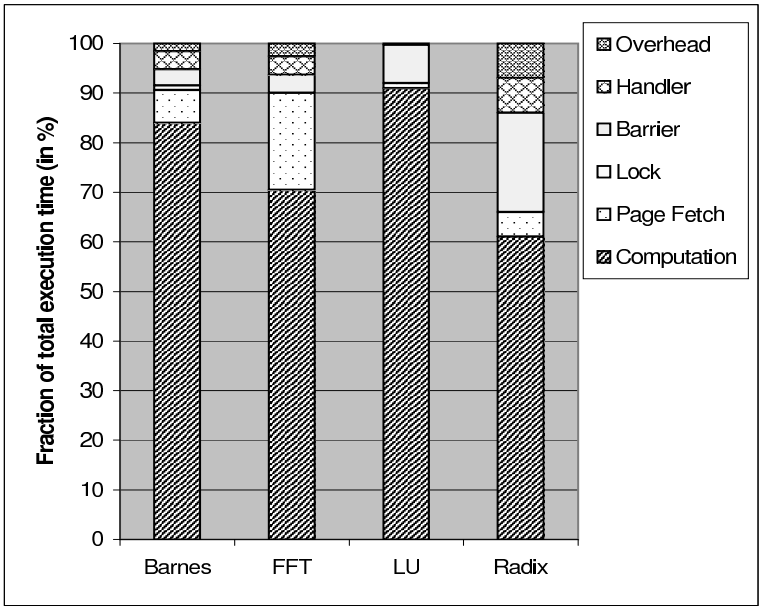

Figure 3. Normalized execution time breakdown.

[4] L. Iftode and J. P. Singh. Shared Virtual Memory: Progress and Challenges. In *Proceedings of the IEEE, Special Issue on Distributed Shared Memory*, volume 87, pages 498–507, 1999.

[5] InfiniBand Trade Association. InfiniBand Architecture Specification, Volume 1, November 2002.

[6] V. Iosevich and A. Schuster. Multithreaded home-based lazy release consistency over VIA. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 59–68, April 2004.

[7] P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory On Standard Workstations and Operating Systems. In *Proceedings of the 1994 Winter Usenix Conference*, pages 115–131, January 1994.

[8] P. J. Keleher. *Lazy Release Consistency for Distributed Shared Memory*. PhD thesis, Department of Computer Science, Rice University, January 1995.

[9] L. Lamport. How to Make a Multiprocessor That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, 28(9):241–248, 1979.

[10] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. PhD thesis, Yale University, September 1986.

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

[12] Mellanox Technologies Incorporation. Mellanox IB-Verbs API (VAPI)–Mellanox Software Programmer's Interface for INFINIBAND VERBS, 2001.

[13] R. Noronha and D. K. Panda. Designing High Performance DSM Systems using InfiniBand Features. In *DSM Workshop, in conjunction with 4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, April 2004.

[14] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architecture: Implementation and Performance. In *Proceedings of the 4th Annual Linux Showcase, Extreme Linux Workshop*, pages 341–352, Atlanta, USA, October 2000.

[15] D. J. Scales, K. Gharachorloo, and C. A. Thekkath. Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory. In *Proceedings of the 7th Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 174–185, October 1996.

[16] M. Swanson, L. Stroller, and J. B. Carter. Making Distributed Shared Memory Simple, Yet Efficient. In *Proceedings of the 3rd International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 2–13, March 1998.

[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.