# Reducing Diff Overhead in Software DSM Systems using RDMA Operations in InfiniBand *

Ranjit Noronha and Dhabaleswar K. Panda

Dept. of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{noronha,panda}@cis.ohio-state.edu

## Abstract

*Software DSM systems do not perform well because of the combined effects of increase in communication, slow networks and the large overhead associated with processing the coherence protocol. Modern interconnects like Myrinet, Quadrics and InfiniBand offer reliable, low latency (around 5.0 $\mu$s point-to-point), and high-bandwidth (upto 10.0 Gbps in 4X InfiniBand). These networks also support efficient memory- based communication primitives like RDMA-Read and RDMA-Write. These supports can be leveraged to effectively reduce overhead in a software DSM system. In this paper, we explore techniques for reducing the diff overhead. These techniques are employed in a protocol called PIPE, which uses RDMA-Write. Application level evaluation shows a maximum improvement of upto 35% in parallel speedup.*

**Keywords:** *DSM Systems, Cache coherency protocol, InfiniBand, System Area Networks*

## 1 Introduction

In recent years, clusters have been widely deployed for providing low-cost high performance computing for a wide-range of applications. There is a range of high-end networking technologies available to connect the machines within a cluster. Technologies like Myrinet [9], Quadrics [4] and InfiniBand [2] offer point-to-point latency of the order of 5.0 $\mu$s for small messages and very high unidirectional bandwidth of the order of 10 Gigabits per second (with InfiniBand 4X) for large messages. In addition to the basic communication primitives, these networks offer a variety of services and operations. For example, Myrinet and Quadrics have a programmable network interface card. InfiniBand and Myrinet support hardware-based remote atomic operations [10]. All these networks also support Remote Direct Memory Access (RDMA) operations. RDMA allows a process to read or write a location in the memory space of another process over the network. RDMA does not require receiver intervention for posting descriptors, an important consideration while designing scalable software systems.

Though considerable research has focused on the development of Software Distributed Shared Memory Systems [24, 14, 8], implementations of SDSM systems such as TreadMarks [15, 6] and HLRC [13, 23] have not been found to be scalable. These SDSM systems are communication intensive, and depend critically on performance of networking technologies like Fast Ethernet, Giganet [11] and the earlier generation of Myrinet [9]. Heavy-weight protocols like TCP could not keep up with the communication rate of SDSM's [21]. Modern networks are fast out-pacing the capacity of processors to keep them filled to capacity. These modern networks might not only impact the performance of SDSM systems but allow one to explore new previously inconceivable protocols for SDSM. Communication in SDSM can largely be characterized by the client-server request-response model shown in Figure 1. One of the protocol activities is diffing, which is creating and sending modifications to pages back to the page manager. At the page manager, the modifications are applied to the original page. After this is complete, an acknowledgment is sent back to the sender. This activity constitutes considerable overhead, specially if there is fair

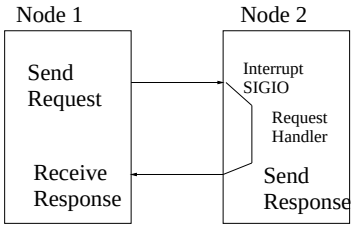


**Figure 1. DSM client server communication model.**

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CCR-0204429 and #CCR-0311542.

amount of sharing within the application.

Modern day interconnects allow users to read and write data elements from the user space of another process using RDMA operations. RDMA may be used in software DSM systems for communication. Diffing is an important software DSM protocol activity, constituting considerable overhead. The general mechanism for a diff, consists of comparing a modified page to its clean copy called a twin. Then a run-length encoding of the differences is created. This encoding is then sent to the server or home node using RDMA Write. The server or home node then applies these diffs to the page. We have proposed a scheme, which partially packs and partially pipelines diffs. This helps improve the network utilization and reduce some of the congestion in the network.

The rest of this paper is organized as follows. Section 2 describes the implementation of HLRC and its main features along with an overview of the networking interconnect InfiniBand. Section 3 presents design possibilities of HLRC with InfiniBand mechanisms. Section 4 explores the design issues and alternatives used while implementing pipelined diffing using RDMA operations. Section 5 evaluates the design using various applications. Section 7 presents conclusions and future directions.

## 2 Background Information

In this section we discuss the basic concepts behind the SDSM package HLRC with an emphasis on its communication model primitives. We also take a look at the InfiniBand standard with a focus on the main communication operations provided by this interconnection technology.

### 2.1 Overview of HLRC

Since the development of the first sequentially consistency SDSM system IVY [16], there has been a large body of research into the issues with SDSM. Unfortunately, SDSM has not been found to be scalable, largely because of the effects of protocol and communication overhead. The lazy release consistency model was the next advance postponed coherence activities to synchronization points, reducing the amount of communication. The home based lazy release consistency protocol (HLRC) [13] improved upon LRC by assigning pages to homes, with a home node being updated with modifications at every synchronization point.

HLRC reduced not only the communication associated with non-home based protocols, but also the memory footprint. In HLRC every page and lock is assigned a home node. At every synchronization point, the diffs for a particular page are sent to the home node and the memory for the diffs are released. A home node can be assigned in a variety of ways; the default behavior is that the default home of the page assigns it to the node that first requests that page.

An implementation of HLRC [23] over the Virtual Interface Architecture (VIA) [5] (HLRC-VIA) was carried on GigaNet [1]. The implementation of HLRC-VIA was multithreaded. The application thread would compute while an associated signal handler would take care of coherence activity on a page fault or miss. A separate thread would listen for incoming requests from other remote processes such as page fetches and lock requests. HLRC-VIA makes use of the RDMA constructs provided by VIA. Request messages or messages for services are sent via RDMA Write with immediate data. This generates an asynchronous request at the receiver which then processes and responds to this request by either forwarding this request to some other nodes or itself satisfying the request through several RDMA Write operations. The requester meanwhile polls a particular location in memory (to which the remote server writes using RDMA Write) to see whether the request has completed. In this paper we use a version of HLRC-VIA modified to work over the InfiniBand fabric. The next section briefly discusses the InfiniBand architecture.

### 2.2 Overview of InfiniBand

The InfiniBand standard is a framework for a System Area Network for connecting processing and I/O nodes. It defines various communication and management functions that are necessary to operate the interconnection fabric. InfiniBand uses a switched, channel-based interconnection fabric, which allows for higher bandwidth, more reliability and better QoS support. Interface to the fabric is through a Host Channel Adapter (HCA) on the processing node. Semantics of various operations are defined via InfiniBand Verbs. The Mellanox implementation of the InfiniBand Verbs API called VAPI [3] supports the basic send-receive model and the RDMA operations read and write. There is also support for atomic operations and multicast. More details on InfiniBand can be obtained from [2].

## 3 HLRC Design Possibilities with InfiniBand Mechanisms

Let us now examine the potential for integrating network based support into HLRC. HLRC duplicates activities either already provided or which could be done with less overhead by network level services in InfiniBand. Figure 2 shows some of the matches between InfiniBand level primitives and HLRC protocol activities. More specifically the following should be possible :

1. Asynchronous handling could be eliminated through the combination of atomic operations and RDMA Read support. Page fetching operations could potentially benefit from this type of support.

2. Diff propagation in HLRC uses RDMA Write with immediate data which requires activation of the asynchronous handler. Diff processing can be potentially eliminated by performing RDMA Read operations. In this design, whenever a particular portion of a page is needed, it can be fetched from the current owner by issuing an RDMA Read operation. The owner does not have to be interrupted to perform this operation.

3. Diffs propagate via RDMA Write. Diffs for several pages may be packed together into a single message and sent to the manager node. Another technique is to break the diff computation into several small messages and send them separately. Potentially, computation and communication overlap can be achieved.

4. Write notice and Barrier notification propagate via RDMA Write. Since these go to all other nodes, hardware based multicast could provide an efficient basis for this operation. This could potentially reduce significantly the amount of traffic needed for synchronization in an SDSM system.

5. Locking could be achieved through the use of remote atomic operations. This could potentially benefit applications which frequently use locks; as the need to frequently process lock requests at the manager node and the last owner is eliminated.

6. Asynchronous request messages could potentially propagate via higher priority service levels achieving better response time.

In this paper we focus on the 3rd option; diff packing and pipelining. Options 1 and 2 are explored in [22] and the protocol ORIG is derived from this work. Since other features (such as reliable multicast and service levels) are not yet completely operational in current generation InfiniBand hardware, we plan on investigating other enhancements in the future.

## 4 Design of PIPE

In this section we discuss the design of our proposed protocol termed as PIPE. We start out by examining the existing protocol termed ORIG as defined in [22]. Following that, in Section 4.3 is a description of the design of PIPE. Finally, various design trade-offs for PIPE are considered in Section 4.4.

### 4.1 Base protocol

The base protocol we are using is the home based lazy release consistency protocol. In this protocol, every page and lock is assigned a home. All requests for accesses to a page or a lock go to the home node. Similarly all updates for

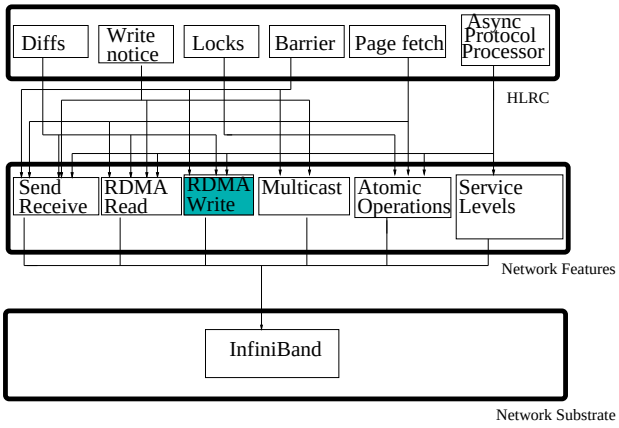


**Figure 2. The SDSM primitives which could benefit from network support**

a page and a lock go to the home node. Updates or diffs for a page propagate to the home node at synchronization points such as a lock release or a barrier. In the next few sections, we discuss the existing protocol ORIG and the pipelined version PIPE. To understand the protocols, we define the following structures:

- Let *modified(X,n)* refer to the n'th position in page X (which has been modified since last reading a copy of the page).
- Let *clean(X,n)* refer to the n'th position in the twin of X where twin refers to a clean copy of page X.
- Let *buffer(t,n)* refer to the n'th position in communication buffer t.
- Let *Available(t)* indicate the status of the communication buffer t. The value TRUE either means that it has never been used, or that it has been ACKed by the receiver
- Let *MaximumBuffers* be the maximum number of communication buffers available for diffing.
- Let *CurrentBuffer* be the current buffer which should be used for communication. Assume it is initially zero.
- Let *RWRITE(source,dest,t,s,len)* denote an RDMA Write descriptor initiated at node source bound for node dest, using buffer t, starting at location s and of length len.
- Let *TS(P)* denote the timestamp of page P.

We now examine diffing in ORIG.

### 4.2 Diffing in ORIG

Now we go through the protocol steps when computing and applying diffs. As shown in Figure 3, node N0 arrives at a synchronization point such as a barrier or a lock. At this point, node N0 must propagate all updates it has made to all pages to the home node N1. Assume node N0 has modified pages X and Y. It computes a diff for a particular page say X. This diff is a run-length encoded string of the differences between the original page and the modified page. It sends these differences to the home node N1 through RDMA Write along with a message containing the timestamp for that page. If there are sufficient buffers remaining, it computes the diff for page Y and sends it to N1 along with the timestamp. The home node applies these diffs and sends an ACK back for each diff received, which indicates that the buffers may reused.

We now examine the protocol activity in more detail. We use the notation defined in Section 4.1 above. Let us

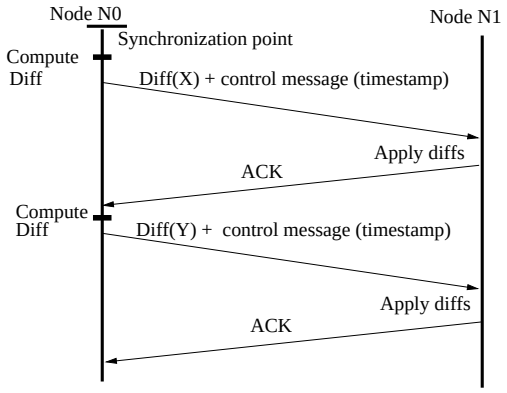


**Figure 3. The original ORIG protocol (for an example scenario)**

further assume that *modified(X,i..j)* differ from *clean(X,i..j)* and that *modified(Y,l..m)* differ from *clean(Y,l..m)*. In this case, ORIG performs the following actions :

1. Set Page = X.
2. Wait until *Available*(*CurrentBuffer*) is *TRUE*.
3. Copy *modified(Page,i..j)* into *buffer(CurrentBuffer,0..(j-i-1))* and append *TS*(Page).
4. Create and issue *RWRITE(0,1,CurrentBuffer,0,(j-i) + length(TS))*.
5. Increment *CurrentBuffer* modulo *MaximumBuffers*.
6. Set Page = (next page to be diffed)
7. Start with item 2.

At node N1, on receiving a corresponding DIFF message from N0, ORIG updates the timestamps and applies the diffs for the given page and sends an ACK back. This sets the flag *Available* for the corresponding buffer on N0 to *TRUE*. In other words, the buffer may be reused.

We will now see how ORIG can be enhanced.

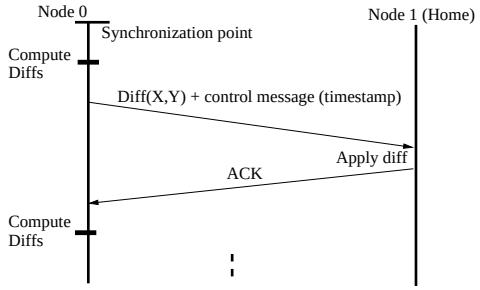


**Figure 4. The proposed protocol PIPE (for the example scenario)**

### 4.3 PIPE protocol

In this section, we discuss our proposed protocol PIPE. Figure 4 shows a sample protocol activity. PIPE first packs a numbers of diffs together and then uses RDMA Write to propagate the diffs to the destination node. Again consider the original diff creation and application activity shown in Figure 3. In this case, PIPE on node N0 will pack diffs for pages X and Y into a single buffer along with their timestamps (assuming the buffer is large enough) and send it to node N1.

Consider the protocol steps in detail. Assume the same terminology as in Section 4.1. Additionally, define the parameter *DiffBufferLength(t)* which is in essence the length of communication buffer t. PIPE executes the following algorithm:

1. Set Page = X.
2. Wait until *Available*(*CurrentBuffer*) is *TRUE*.
3. Set dlength = 0
4. Set Temp = ( dlength + current diff size + length(TS) )
5. if Temp $\leq$ DiffBufferLength(CurrentBuffer )
   (a) Copy *modified(Page,i..j)* into *buffer(CurrentBuffer,dlength..dlength+(j-i-1))*.
   (b) Append *TS*(Page).
   (c) Add current diff size and length(TS) to dlength.
   (d) If there are modified pages remaining, set Page to next modified page and continue with step 4.

6. Create and issue *RWRITE(0,1,CurrentBuffer,0,dlength)*.
7. Increment *CurrentBuffer* modulo *MaximumBuffers*.
8. Set Page = next modified page.
9. If there are no more pages remaining, exit.
10. Continue with item 2.

At node N1 as the respective diffs and timestamps come in PACK unpacks them and applies them to the respective pages.

### 4.4 Tradeoffs in PIPE design

In this section we discuss some of the tradeoffs while designing PIPE. In particular we will examine the impact of pipeline depth, and packed diff size discussed in the next two sections.

#### 4.4.1 Pipeline Depth

As discussed in Section 4.3, PIPE computes several diffs, marshals them into a single message, sends it and then proceeds to compute the next set of diffs. However at some point, it might run out of buffers in which to place the diffs, especially if there are many large size diffs. At this point, it needs to wait for an ACK from the receiver, indicating which buffers have been freed up. The number of packed diffs PIPE can send before waiting for an ACK is called the pipeline depth.

The pipeline depth is limited by the number and size of receive buffers that can be pinned down. Receive buffers large enough to hold the largest size packed diff must be posted corresponding to each stage of the pipeline. A longer pipeline might be useful when the destination is heavily loaded and cannot process the incoming diffs fast enough. A shorter pipeline is desirable when the network is congested and the sender is quickly constrained by the shorter pipeline.

#### 4.4.2 Packed Diff Size

The size of each packed diff potentially affects performance. It is a tradeoff between effective network bandwidth utilization and overlap between diff creation time and propagation time. The larger the packed diff, the longer it takes for the updates to become available. However large messages allow for better utilization of network bandwidth. The size of the packed diff should be large enough so that the time required to compute it dominates the time to post the corresponding send. On the other hand, the packed diff size should not be too small, otherwise send descriptor posting time will dominate and network bandwidth will be poorly utilized.

## 5 Performance Evaluation

This section evaluates the performance of PIPE with respect to ORIG. Evaluation is in terms of overall execution time, and diff creation time, and is discussed in the following sections. First we describe the hardware setup in 5.1. In section 5.2 application level evaluation is presented. Following that effect of the pipelining depth is discussed. Finally, some of the remaining bottlenecks are discussed.

### 5.1 Experimental Test Bed

The experiments were run on a 16 node cluster connected through an InfiniScale MTS-2400 24 4x Port InfiniBand Switch. The HCAs are Mellanox InfiniHost MT23108 DualPort 4X HCA's. Eight out of the 16 machines are SuperMicro SUPER P4DL6's, each with dual Pentium Xeon 2.4 GHz processors, 512 MB of main memory and a 133 MHz PCI-X bus. The other 8-nodes are SuperMicro X5DL8-GG's, each with dual Pentium Xeon 3.0 GHz processors, 1 GB of main memory and a 133 MHz PCI-X bus. The kernel is a SMP version of Linux 2.4.22.

### 5.2 Application level evaluation

In this section, we evaluate our design using three applications; Barnes-HUT (Barnes), Non-contiguous LU decomposition (LU), Non-contiguous Ocean simulation (Ocean) from the SPLASH-2 benchmark suite [26] and Integer sort (IS) from the TreadMarks [15] SDSM package. The application sizes used are shown in Table 1. All

| Application | Parameter | Size |
|---|---|---|
| Barnes | Bodies | 32678 |
| IS | num of keys | $2^{24}$ |
| LU | Matrix Dimension | 1024 |
| Ocean | Grid Size | 258 x 258 |

**Table 1. Application sizes**

other parameters were kept the same as originally described in [26]. In the next section, we will discuss some of the important application characteristics. Following that, the performance numbers for Barnes, IS, LU and Ocean are discussed.

#### 5.2.1 Application Characteristics

In this section, we discuss some of the application characteristics relevant to our design. The first application Barnes, is an N-Body simulation using the hierarchical Barnes-Hut method. It contains two main arrays, one containing the bodies in the simulation and the other the cells. Sharing patterns are irregular and true. As shown in Table 2, a fairly

| Application | Barnes | IS | LU | Ocean |
|---|---|---|---|---|
| Average Diff Traffic (MegaBytes) | 1.83 | 29.55 | 10.9 | 9.16 |
| Average Number of Diffs | 6060 | 7680 | 15114 | 14327.56 |
| Average Diff Size (bytes) | 317 | 4034 | 756.21 | 670.38 |
| Average Number of Intervals | 13 | 17 | 129 | 937 |
| Average Number of Diffs Per Interval | 466.15 | 451.764 | 117.16 | 15.29 |

**Table 2. Per node application characteristics for a 16 node run**

large amount of diff traffic is exchanged at barriers, which are the synchronization points.

IS implements Bucket Sort. There is a global array containing the buckets, and a local array which the local node uses to sort its data. After each iteration, each node places its data in the global array and copies the data relevant to it into its local array. As shown in Table 2, a large numbers of diffs are exchanged at intervals. Also each diff has an average size of 4K.

The LU program factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The factorization uses blocking to exploit temporal locality on individual submatrix elements. LU sends a large numbers of diffs. The total diff traffic is of the order of 11 MB.

The Ocean program simulates large-scale ocean movements based on eddy and boundary currents. Ocean also sends a large number of diffs with diff traffic of the order of 9MB. Ocean also has the highest number of intervals because of its use of locks.

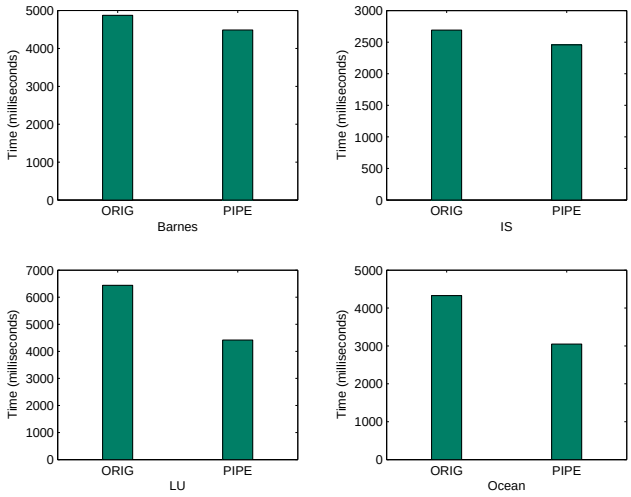


**Figure 5. Overall execution time for the applications when using ORIG and PIPE**

#### 5.2.2 Effect on Execution time

In this section, we discuss the performance results for the applications Barnes, IS, LU and Ocean.

Figure 5 shows the execution time for different applications at 16 nodes. *PIPE* has lower overall execution time than *ORIG* for all applications. For Barnes, when using PIPE the execution time reduced by 8%. For IS, the execution time reduced by a factor of 8.7% when using PIPE. In the case of LU, there was a reduction in execution time of 35% when ORIG was replaced by PIPE. Finally, for Ocean, when using PIPE, the execution time reduced by 29%.

#### 5.2.3 Pipeline Depth

Experimentation with diff pipeline depth showed that increasing the depth beyond two stages did not significantly improve performance. This can mainly be attributed to optimal pipelining being achieved at two stages. Here both the sender and receiver are busy creating and applying diffs respectively and there is no wait time for diff buffers at the sender.

#### 5.2.4 Remaining Bottlenecks

In this section, we discuss some of the remaining bottlenecks in the DSM system. As can be seen from Figure 6, the applications spend a considerable amount of time waiting at barriers. This can mainly attributed to the sequential phases in the application, leading to a load imbalance.

## 6 Related Work

Ever since the proposal for the first SDSM system IVY [16] there has been considerable research conducted into the SDSM systems. However SDSM was not found to be scalable. The benefits of implementing HLRC over low level protocol like VIA was examined in [23]. This implementation sends diffs in an eager fashion as soon as it was computed. TreadMarks implements the homeless lazy release consistency protocol [15]. Updates are sent in the form of diffs and are packed. However, the diff size is limited by the MTU (32K) of the communication substrate UDP. Treadmarks is implemented directly over a low level communication protocols like VIA and GM over Myrinet,

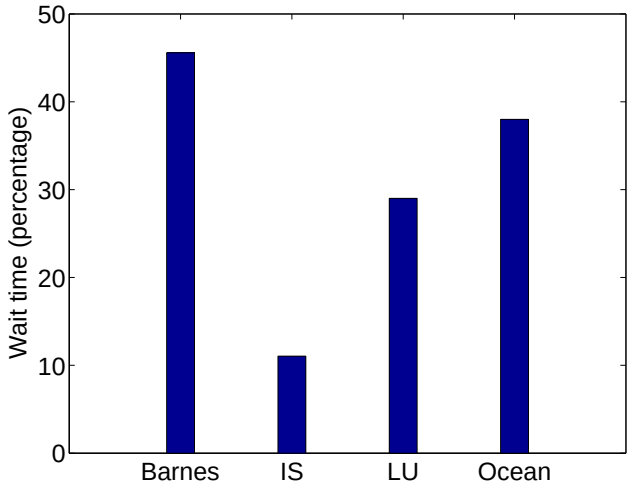


**Figure 6. Application wait time at a barrier, shown as a percentage of overall execution time.**

and was shown to substantially reduces wait times and improves scalability in [7, 21]. However, the limit of 32K on the packed diff size is still imposed. A study of the effect of replacing diff packing with an RDMA Write based protocol is studied in [22]. Using a technique called network based Remote Write in the Cashmere SDSM system was studied in [24]. Network interface support was used to perform virtual memory mapped communication in addition to DMA based communication along with protected, low-latency user-level message passing in the SHRIMP project [19]. A proposal for using active-memory support for SDSM systems to achieve software DSM with hardware DSM performance is discussed in [12]. Reducing the effect of false sharing is discussed in [20]. Investigations into using switching technology to improve performance in SDSM systems is discussed in [17, 18]. Using kernel-level access to InfiniBand primitives to improve sequentially consistent SDSM performance is discussed in [25].

## 7 Conclusions and Future Work

In this paper, the effect of reducing the overhead of diff propagation was studied. Diff were partially packed together and then sent over the network. This helped improve the overlap of communication and computation and make more effective use of network bandwidth. These were integrated into a protocol called PIPE. Experimental evaluation using applications showed an improvement in parallel speedup upto 35 %.

Significant improvements can still be made. It might be possible for the home node to directly read the diffs from other nodes and then integrate them on request. This might help reduce overhead. Further rewriting the application to parallelize sequential phases could produce significant improvement in parallel speedups.

## References

[1] Giganet. *www.giganet.com*.

[2] Infiniband Trade Association. *www.infinibandta.org*.

[3] Mellanox Technologies. *www.mellanox.com*.

[4] Quadrics Ltd. *www.quadrics.com*.

[5] Virtual Interface Architecture Specification. http://www.viarch.org.

[6] C. Amza, A. Cox, et al. Treadmarks: Shared Memory Computing on networks of workstations. *IEEE Computer, 29(2):18-28*, feb 1996.

[7] M. Banikazemi, J. Liu, . K. Panda, and P. Sadayappan. Implementing TreadMarks over VIA on Myrinet and Gigabit Ethernet: Challenges Design Experience and Performance Evaluation. *Int'l Conference on Parallel Processing*, sep 2001.

[8] J. Bjoerndalen, O. J. Anshus, B. Vinter, and T. Larsen. Comparing the performance of the pastset distributed memory system using TCP/IP and M-VIA. *The Second International Workshop on Software Distributed Shared Memory*, 1995.

[9] N. J. Boden, D. Cohen, et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, pages 29–35, Feb 1995.

[10] D. Buntinas, D. K. Panda, and W. Gropp. NIC-Based Atomic Operations on Myrinet/GM. *SAN-1 Workshop, held in conjuction with High Performance Computer Architecture (HPCA)*, 2002.

[11] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000 Mbps.

[12] M. Heinrich and E. Speight. Providing Hardware DSM Performance at Software DSM Cost. *Technical Report No. CSL-TR-2000-1008, Cornell University, Ithaca, NY*, November 2000.

[13] L. Iftode. Home Based Shared Virtual Memory. *PhD Thesis, Technical Report TR-583-98, Princeton University*, 1998.

[14] A. Itzkovitz, A. Schuster, and Y. Talmor. Harnessing the power of fast low-latency networks for software dsms. *The First Workshop in Software Distributed Shared Memory*, 1999.

[15] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the 1994 Winter Usenix Conference*, Jan. 1994.

[16] K. Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proceedings of the International Conference on Parallel Processing*, pages 94–101, Los Alamitos, CA, 1988.

[17] O. Lysne. Deadlock avoidance for switches based on wormhole networks. *Proc. of the 1999 International Conference on Parallel Processing*, 1999.

[18] M. Thottethodi, A. Lebeck, and S. Mukherjee. BLAM: A High-Performance Routing Algorithm for Virtual Cut-Through Networks. *International Parallel and Distributed Symposium*, April 2003.

[19] M.A. Blumrich, C. Dubnicki, E.W. Felten, Kai Li, M.R. Mesarina. Two Virutal Memory Mapped Network Interface Designs. *Proc. of the Hot Interconnects Symp.*, 1994.

[20] L. Monnerat and R. Bianchini. Effi ciently Adapting to Sharing Patterns in Software DSMs. *High-Performance Computer Architecture (HPCA)*, Feburary 1997.

[21] R. Noronha and D. K. Panda. Implementing TreadMarks over GM on Myrinet: Challenges, Design Experience and Performance Evaluation. *Workshop on Communication Architecture for Clusters (CAC'03), held in conjuction with IPDPS '03*, April 2003.

[22] R. Noronha and D.K. Panda. Designing High Performance DSM Systems using Infi niBand Features. *Workshop on Distributed Shared Memory on Clusters (DSM' 04), held in conjuction with CCGRID '04*, April 2004.

[23] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architectur: Implementation and Performance. *Proc. of the Annual Linux Showcase, Extreme Linux Workshop, Atlanta*, October 2000.

[24] R. Stets, S. Dwarkadas, L. Kontothanassis, U. Rencuzogullari, and M. L. Scott. The Effect of Network Total Order, Broadcast, and Remote-Write Capability on Network-Based Shared Memory Computing. In *International Symposium on High-Performance Computer Architecture*, 2000.

[25] T. Birk, L. Liss and A. Schuster. Effi cient Exploitation of Kernel Access to Infi niBand: a Software DSM Example. *Hot Interconnects*, August 2003.

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