Appraising the Communication Performance of a DSM Cluster with a Low-level Application Programming Interface

Bernady O. Apduhan Yasushi Shimono Itsujiro Arita

Department of Artificial Intelligence
Kyushu Institute of Technology
Iizuka, 820-8502 Japan
{bob, yasushi, arita}@mickey.ai.kyutech.ac.jp

Abstract

In this paper, we study the communication performance of a DSM cluster computing environment (DSE) with a Low-level Application Programming Interface (LAPI) using shared memory routines, and investigate its influence on the system's scalability. We used some shared memory communication tests with low and high contention to represent commonly used communication patterns. The shared memory routines sat on top of DSE's communication primitives. These were implemented using LAPI or socket APIs. Communication was via a high performance interconnection switch. The experiments are presented and preliminary results of a DSM cluster implemented with LAPI and its performance compared to using socket APIs are discussed.

1 Introduction

Parallel computing on networked workstations or PCs has been gaining more attention in recent years. The performance improvements of workstations or PCs and high-speed networks are paving the way for the widespread usage of cluster-based parallel systems.

Currently, message-passing and distributed shared memory (DSM) are the two prevailing programming models for parallel computing in a cluster computing environment. In the message-passing paradigm (e.g., MPI or PVM), the programmer must be aware of the data location and the timing, and must determine what to communicate, to which processor. This makes it cumbersome to program with the message-passing paradigm, particularly for applications with complex data structures.

On the other hand, software DSM systems [8] provide a shared memory abstraction on top of the native message-passing facilities. An application can be written as if it will be executed on a shared memory multiprocessor, using typical read and write operations to access shared data. The message-passing operation is left to the underlying DSM system. Although it is convenient to program in DSM, DSM systems tend to generate more communication and are seen as less efficient than message-passing systems. With message-passing, the communication is wholly dealt with by the programmer, who must be knowledgeable on the data usage pattern. In DSM, the system has little knowledge of the application program and should be conservative in deciding what to communicate. Because sending messages between workstations is expensive, this extra communication can induce serious performance degradation.

Among the popular DSM systems developed are IVY [13], Treadmarks [11], Midway [12] and CRL [10]. Each DSM system has its own design goals and problem domains. Our laboratory has likewise developed an experimental DSM test bed, called DSE, to study the problems and issues of a DSM-based cluster computing system [2].

In general, parallel applications on network-based computing systems are most sensitive to communication overhead. A number of approaches have been proposed to bypass the conventional protocol stack and directly access streamlined communication protocols running on high-speed networks.

Current DSEs are implemented at the UNIX user level and use TCP sockets and expensive UNIX system calls for communication. This arrangement provides portability at the expense of poor system performance, especially for fine-grain communication. The availability on the IBM SP2 system of the Low-level Application Programming Interface (LAPI), which was designed to provide low-latency on short messages, provides an appropriate testbed to study the communication performance enhancement of DSE. Our primary intent is to explore the performance capabilities of LAPI using a High-Performance Switch (HPS) [1] in the IBM SP2, and to study its influence on the performance of a DSM cluster system.

This paper is organized as follows. We describe DSE and its software organization in Section 2, and provide an abridged description of LAPI and the experimental testbed in Section 3. In Section 4, we present the experiments, discuss the preliminary performance results and compare with use of the sockets API. We describe related work in Section 5, and in Section 6 we give our concluding remarks.

2 DSE and Software Organization

DSE is a DSM cluster of networked workstations or PCs running on UNIX-based OS platforms (SunOS, Solaris, AIX, Linux, Free-BSD), interconnected by a local area network [2]. DSE is implemented at the UNIX user level for portability and availability. The DSE system model and software organization are shown in Figures 1 and 2.

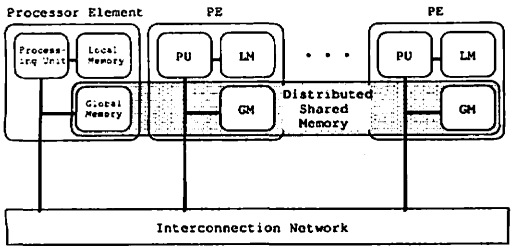

Figure 1: The DSE System Model

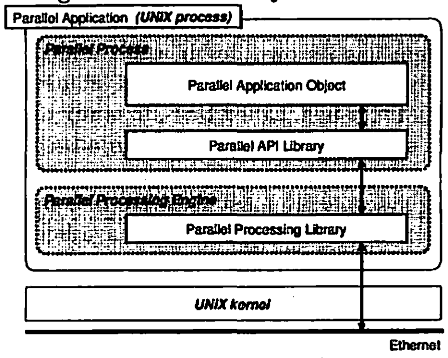
Parallel Application (UNIX process)
Parallel Process
Parallel Application Object
Parallel API Library
Parallel Processing Engine
Parallel Processing Library
UNIX kernel
Ethernet

Figure 2: The DSE Software Organization

3 LAPI and Experimental Setup

The IBM SP2 system is a network of RS/6000 workstations interconnected via an adapter to a high-performance, multistage, packet-switched network (called SP switch, or HPS) with a bi-directional data transfer rate of up to 110 Mbps between node pairs, as well as by 100 Mbps Ethernet. Each node has its own copy of the standard operating system AIX and other standard RS/6000 system software [1].

The Low-level Applications Programming Interface (LAPI) is a non-standard application programming interface designed to provide optimal communication performance on the SP switch. It is available as part of the SP software [3][5]. LAPI is an asynchronous communication mechanism meant to supply flexibility to users when writing parallel programs with dynamic and unpredictable communication patterns. LAPI is designed to be an efficient (low latency, high bandwidth) interface. LAPI functionality includes data communication as well as synchronization and ordering primitives. The LAPI functions are divided into three groups: (1) A basic "active message" infrastructure that allows programmers to install a set of handlers that are invoked and executed in the address space of a target process on behalf of the process originating the active message; (2) A set of defined functions that provide a Remote Memory Copy (RMC) interface for direct remote memory access; and (3) A set of control functions for the initialization and eventual orderly shutdown of the LAPI layer. More detailed descriptions of LAPI functions can be found in [4].

The DSE was ported to SP2 using LAPI, henceforth called *DSE-LAPI*, to exploit the communication performance of the new interface, as depicted in Figure 4. To obtain a comparative communication performance evaluation, we also ported DSE to SP2 using typical TCP sockets, henceforth called *DSE-TCP*, as depicted in Figure 3. Both implementations used the high-speed SP switch.

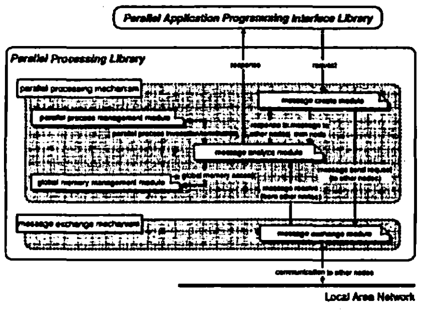
Parallel Application Programming Interface Library
Parallel Processing Library
response
request
parallel processing mechanism
message create module
parallel process management module
parallel process
message analysis module
global memory management module
global memory access
message send request (to other nodes)
message receive (from other nodes)
message exchange mechanism
message exchange module
communication to other nodes
Local Area Network

Figure 3: The DSE Parallel Processing Library (using Sockets)

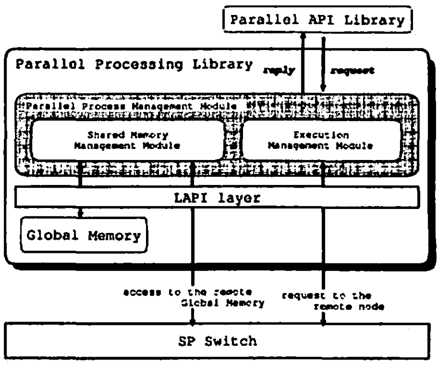
Parallel API Library
Parallel Processing Library
reply
request
Parallel Process Management Module
Shared Memory Management Module
Execution Management Module
LAPI layer
Global Memory
access to the remote Global Memory
request to the remote node
SP Switch

Figure 4: The DSE Parallel Processing Library (using LAPI)

4 Experiments and Results

We conducted the communication performance evaluation using some commonly used shared memory routines or collective communication operations. In DSM, collective operations are used to support synchronization, shared-data invalidation and update, and were therefore a good instrument for evaluation. The distant message access and barrier synchronization routines represent low-contention operations, while the broadcast and all-to-all routines represent high-contention operations. These routines were implemented on top of DSE's primitives.

4.1 Distant Message Access

In this test, a process in node N ($N > 0$) read shared memory data on node 0. In a scalable system, no performance difference should occur in sending messages to close or distant nodes. The results in Figures 5, 6 and 7, show that the communication time of DSE-LAPI is significantly faster than DSE-TCP, where the latter uses expensive send() and receive() system calls to communicate between DSE kernels, and also between the DSE kernel and a DSE process. This procedure incurs a large communication processing overhead and consumes most of the communication time.

The higher communication performance exhibited by DSE-LAPI can be attributed to the Active Message [6] style of communications in LAPI and the Remote Memory Functions (RMC) on the SP switch. LAPI provides one-sided communication and processes are not required to synchronize explicitly. Therefore, reading or writing shared data and other processing are executed simultaneously. This test executes a simple one-to-one communication and clearly shows the high performance of LAPI, and efficient implementation of a DSM with LAPI.

4.2 Broadcast Communication

In this test, the process in node 0 broadcast shared data to other processes. In a scalable system, the execution time would grow linearly with the number of nodes. The broadcast communication test incurs a load on the node holding the broadcast data.

In DSE-TCP, the DSE kernel on node 0 processes all the read messages sent from other nodes and sends the broadcast data to these nodes. Here, the DSE kernel on node 0 must process many messages and invokes expensive system calls many times. As shown in Figures 8, 9, 10, the execution time of DSE-TCP grows exponentially. Notice that the execution time of DSE-LAPI was faster than DSE-TCP in each case and remained constant regardless of the number of nodes. The LAPI RMC functions, which provide one-sided communication and direct remote node memory access, explicitly reduce the memory access load.

4.3 All-to-All Communication

In this test, all nodes broadcast data simultaneously with high contention. Thus the communication traffic on the network was higher than for the broadcast test. In a scalable system, the execution time would grow linearly with the number of nodes. Each execution time of DSE-TCP is longer than DSE-LAPI when the message size is small or when few nodes are used, as shown in Figures 11, 12. The invocation of expensive system calls is seen as the primary cause. In DSE, the data size used by most applications is less than or equal to 1024 bytes. Therefore, the DSE system must provide efficient communication for a fine-grain processing environment. On DSE-TCP, the DSE kernel uses expensive system calls and processes all incoming messages, whereas, on DSE-LAPI, the nodes communicate with each other efficiently using one-sided communication, such as RMC. This is desirable in a fine-grain communication environment.

4.4 Barrier Synchronization

This test evaluated the performance of the DSE synchronization barrier. As shown in Figure 14, the execution time of DSE-TCP is faster than DSE-LAPI, but tends to become slower as the number of processors is increased. On DSE-TCP, each node sends a barrier synchronization message to the node holding the synchronization variable, and this latter node's DSE kernel checks the message queue and processes the messages. The queued messages are passed to the module by passing a pointer to this message. On DSE-LAPI, the requests are sent to the node holding the synchronization variable by invoking the completion handler and placing the requests in the synchronization queue. The system checks the synchronization queue and processes the requests.

On LAPI, the messages are received by the event notification handler in the UNIX kernel. Interrupts for event notification and synchronization queue handling occur frequently in this implementation, causing synchronization latency. This implementation of the synchronization barrier should be improved in the future. It would have been interesting to further evaluate the scalability performance of the system with more processors, but computer availability constraints made this impossible.

5 Related Work

Some previous studies on LAPI and/or DSM on IBM SP2 are as follows. Banikazemi, et al. [7] studied the exploitation of the LAPI library for efficient implementation of standard MPI. They discussed the implementation and mismatches between the requirements of standard MPI and LAPI functionality. Karlsson, et al. [9] performed a comparative characterization of communication patterns in some applications using MPI and Treadmarks on

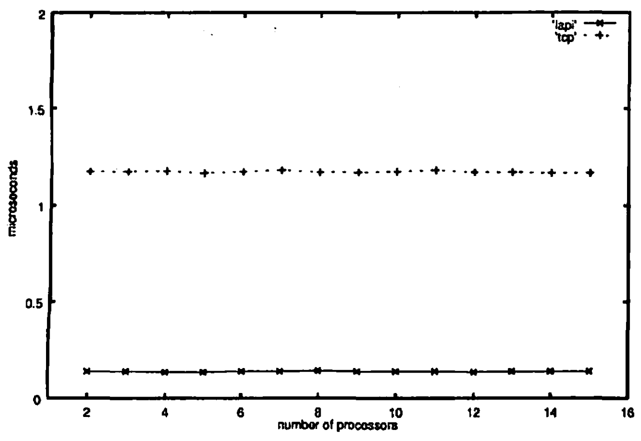

Figure 5: Distant message access (size = 4 Bytes)

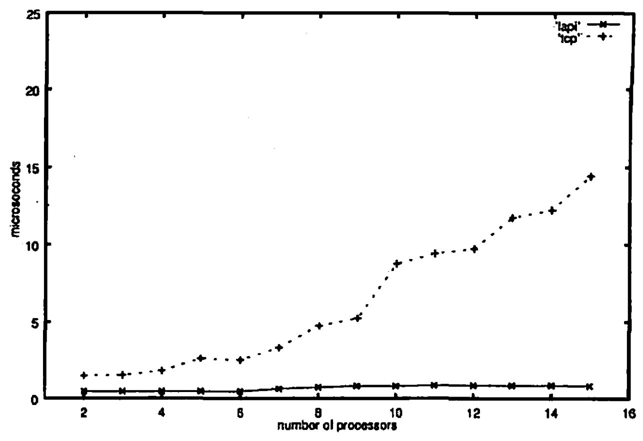

Figure 8: Broadcast (size = 4 Bytes)

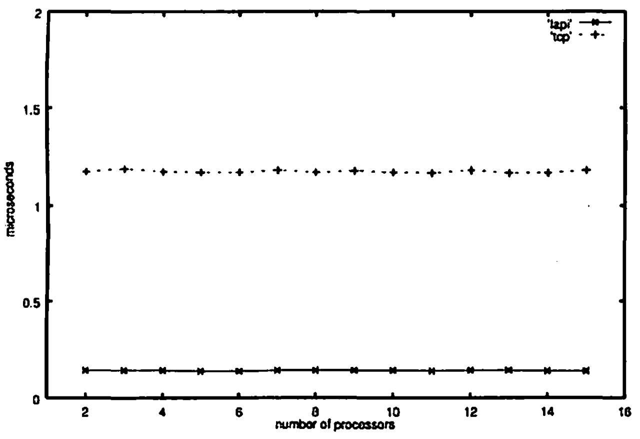

Figure 6: Distant message access (size = 64 Bytes)

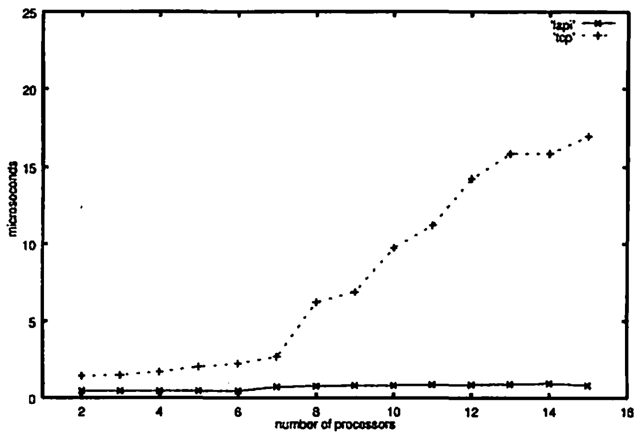

Figure 9: Broadcast (size = 64 Bytes)

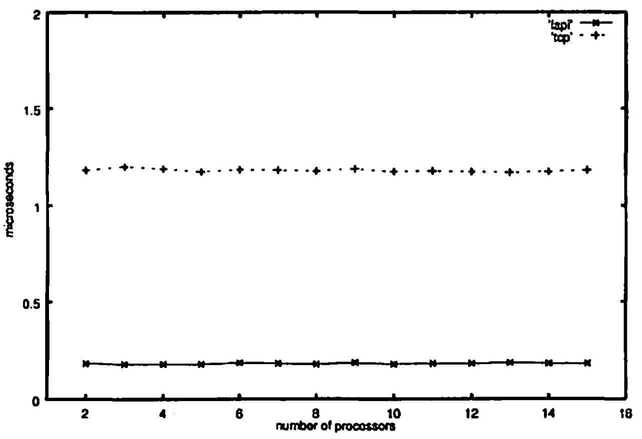

Figure 7: Distant message access (size = 1024 Bytes)

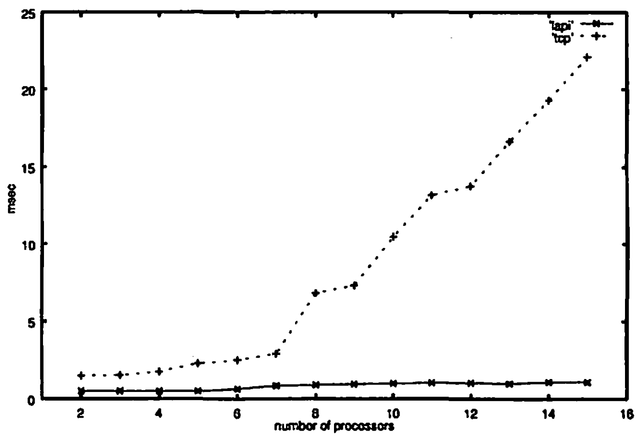

Figure 10: Broadcast (size = 1024 Bytes)

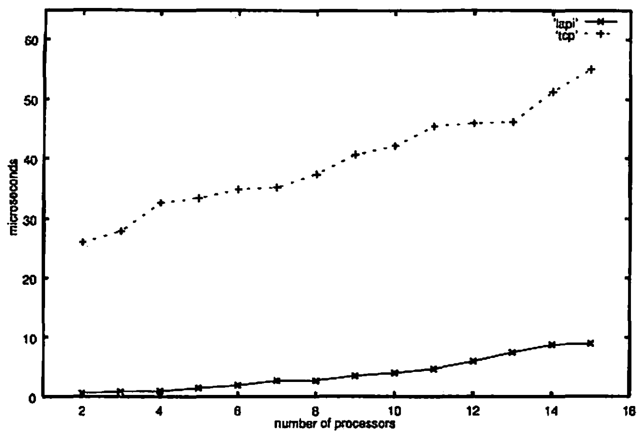

Figure 11: All-to-all communication (size = 4 Bytes)

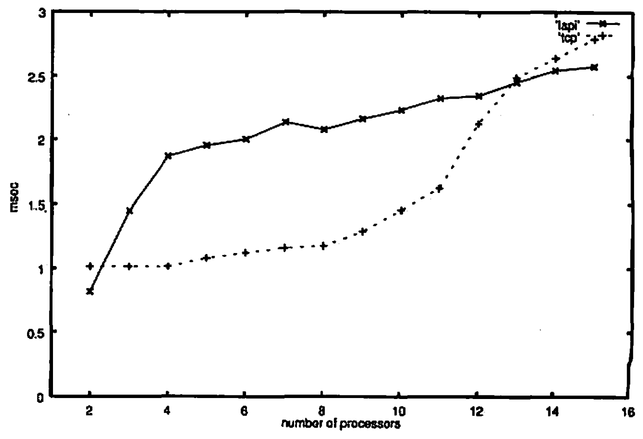

Figure 14: Synchronization barrier (size = 1024 Bytes)

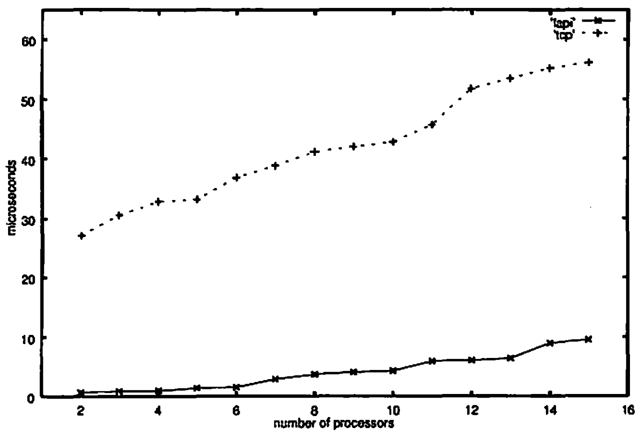

Figure 12: All-to-all communication (size = 64 Bytes)

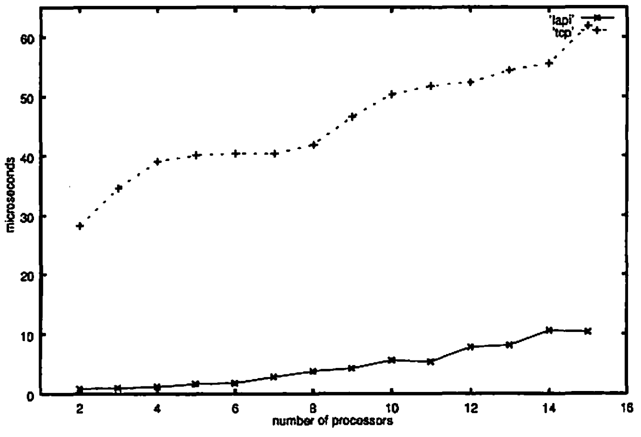

Figure 13: All-to-all communication (size = 1024 Bytes)

an IBM SP2. The study revealed that Treadmarks programs tend to cause a more even network load compared with MPI programs, and discussed ways to improve performance.

Perhaps the work most related to our study is that by Gautam Shah, et al. [5], which included the development of IBM's LAPI library in an effort to optimize the performance of Pacific Northwest National Laboratory's Global Arrays (GA) toolkit and its applications on the IBM RS/6000 SP system. The GA library was ported to exploit the performance of LAPI.

While we share some of our objectives with these studies, we differ significantly on targeted goals. Our goal is to study the performance of a software-DSM system with low-level APIs, such as LAPI, in an effort to gain first-hand experience in implementation issues and their intricacies in developing a highly efficient and portable software-DSM system with a single system image using commodity standard APIs.

6 Concluding Remarks

We described the implementation of a DSM cluster computing environment with LAPI on an IBM SP2 system using an SP Switch. We then evaluated the communication performance using commonly used shared memory routines, and compared it with another implementation using the sockets API. Preliminary performance with LAPI shows promising results in almost all experiments and clarified some related issues.

The results verify the usability of LAPI as a non-standard low-level interface to provide optimal communication on the dedicated SP switch. Likewise, the high performance can be attributed to the active message style infrastructure and the Remote Memory Copy interface inherent in LAPI. The former reduce communication latency and the latter provide direct shared memory access which further provides an overlapping communication and computation processing. The high performance exhibited by LAPI substantiate the merit of using a low-level programming interface on a DSM cluster, and provide insights and intricacies in development.

Future research directions include implementation of the collective communication operations in the DSE kernel to further exploit LAPI performance, and a systematic study of the impact of communication performance on parallel applications in a DSE-LAPI cluster.

References

[1] IBM. *IBM Systems Journal*, Vol. 34, No. 2, 1995.

[2] Tatsuya Asazu, Bernady O. Apduhan, Itsujiro Arita, Towards a Portable Cluster Computing Environment Supporting Single System Image, In *Proc. ICPP'99-MMNS Workshop*, pp. 488-493, Sept. 1999.

[3] IBM. *PSSP for AIX : Command and Technical References*, rel 2.4, document GC23-3900-05, IBM Corporation, 1998.

[4] IBM. *PSSP for AIX : Administration Guide: The Communications Low-Level Application Programming Interface*, rel 2.4, document GC23-3897-05, IBM Corporation, 1998.

[5] Gautam Shah, et al, Performance and Experience with LAPI - A New High-Performance Communication Library for the IBM RS/6000 SP, In *Proc. IPPS '98*, pp. 260-267, March 1998.

[6] T. von Eiken, D.E. Culler, S.C. Goldstein, K.E. Schauser, Active Messages: A Mechanism for Integrated Communication and Computation, In *Proc. International Symposium on Computer Architecture*, pp. 256-266, 1992.

[7] Mohammad Banikazemi, Rama K. Govindaraju, Robert Blackmore and Dhabaleswar K. Panda, Implementing Efficient MPI on LAPI for IBM RS/6000 SP System: Experiences and Performance Evaluation, In *Proc. IPPS '98*, pp.183-190, 1998.

[8] Kai Li , Paul Hudak, Memory Coherence in Shared Virtual Memory Systems, *ACM Transactions on Computer Systems* , Vol. 7, No. 4, pp. 321-359, Nov. 1989.

[9] Sven Karlsson, Mats Brorsson, A Comparative Characterization of Communication Patterns in Applications Using MPI and Shared Memory on an IBM SP2, *Proc. 2nd Int'l. Workshop on Network-Based Parallel Computing*, pp. 189-201, Las Vegas, Nevada, Jan. 31 - Feb. 1, 1998.

[10] Kirk L. Johnson, Frans Kaashoek, Deborah A. Wallach, CRL: High-Performance All-Software Distributed Shared Memory, *Proc. SIGOPS '95*, pp. 213-228, Colorado, USA, Dec. 1995.

[11] Cristiana Amza, Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel, Treadmarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, pp. 18-28, Feb. 1996.

[12] Brian N. Bershad, Matthew J. Zekauskas, and Wayne A. Sawdon, The Midway Distributed Shared Memory System, *Proc. 38th IEEE Computer Society International Conference(COMPCON'93)*, pp. 528-537, February 1993.

[13] Kai Li, IVY: A Shared Virtual Memory System for Parallel Computing, *Proc. of the International Conference on Parallel Computing*, pp. 94-101, 1988.