An Improved DSM System Design and Implementation

T. Ramesh and Chapram Sudhakar
National Institute of Technology,
Warangal, INDIA

In this paper, an Improved Distributed Shared Memory (IDSM) system, a hybrid version of shared memory and message passing version is proposed. This version effectively uses the benefits of shared memory in terms of ease of programming and message passing in terms of efficiency. Further it is designed to effectively utilize the state-of-art multicore based network of workstations and supports standard PThread interface and OpenMP model for writing shared memory programs as well as MPI interface for writing message passing programs. This system has been studied using standard SPLASH-2 (Stanford ParalleL Applications for SHared memory - 2), NPB (NAS Parallel Benchmarks), IMB (Intel MPI Benchmarks) benchmarks and some well known parallel algorithms. Its performance has been compared with JIAJIA DSM system that uses efficient scope consistency model for shared memory programs and with MPI library (MPICH2) on network of Linux systems for MPI programs. Improved Lazy Release Consistency model implemented in IDSM system is compared with Lazy Release Consistency model of TreadMarks system. In many cases IDSM was found to perform much better than those systems.

Keywords: Distributed Shared Memory System, Memory Consistency Model, OpenMP, PThreads, Network of Workstations

1. INTRODUCTION

Network of workstations provide a better alternative for expensive super computers for building parallel programming environment. It can be used as a message passing system or shared memory system through accompanied software DSM system. There are mainly two categories of DSM implementations [Protic et al. 1995] - Hardware level and Software level. In hardware level DSM implementations communication latency is considerably reduced, because of fine-grained sharing that minimizes the effects of false sharing. Searching and directory functions [Lenoski et al. 1992] are much faster. Examples for hardware DSM implementations are Memnet (Memory as Network Abstraction) [Delp et al. 1991], SCI (Scalable Coherent Interface) [Gustavson 1992] and KSR family of systems [KSR 1992]. In software level DSM implementations, the mechanism is quite similar to that of virtual memory management, except that on a page fault, the data is supplied from local memory of the remote machine. Software based DSMs can be implemented at several levels such as Compiler, Runtime library and Operating system levels. In the compiler level implementations, the access to shared data is automatically converted into synchronization and coherence primitives, as in Linda [Ahuja et al. 1986]. In the case of user level runtime packages, the DSM mechanism is provided in the form of runtime library routines, which can be linked to the application. Examples for this category are IVY [Li and Hudak 1989] and Mermaid [Zhou et al. 1990]. At Operating System level, DSM can be implemented inside or outside the kernel. In the first case the DSM mechanisms are incorporated into the operating system kernel where as in the second case this mechanism is incorporated into the specialized software controller processes. Mirage [Fleisch and Popek 1989] is an example for inside the kernel implementation and Clouds [Ramachandran and Khalidi 1991] is an example for outside the kernel implementation. Operating system level (inside the kernel) implementations are generally

Authors' Addresses: T. Ramesh, Department of Computer Science and Engineering, National Institute of Technology - Warangal, INDIA.
Chapram Sudhakar, Department of Computer Science and Engineering, National Institute of Technology - Warangal, INDIA.

considered to be more efficient among all software DSM implementations. The semantics of the operating system can be standardized and the applications need not be modified for porting to a new DSM system [Protic et al. 1995]. This standardization simplifies the programming effort in distributed applications.

The proposed IDSM system is inside the kernel implementation. This system supports various memory consistency models [Tanenbaum 2003; Steinke and Nutt 2004; Adve and Gharachorloo 1996; Keleher et al. 1992] and provides the user the flexibility of selecting the suitable memory consistency model for different parallel processing applications. Main features and advantages of the developed IDSM system are given below.

— IDSM is implemented at kernel level and hence is much faster and efficient.

— Kernel of IDSM system is fully preemptive and multithreaded. Delays in processing important events related to memory consistency and other networking functions are reduced to a minimum.

— Light weight threads are supported transparently over network of work-stations using standard PThread interface (and OpenMP) which makes development of applications for the new DSM system easy.

— Within an application true-multithreading is supported and using that it can be benefited by fully utilizing the computing power of multicore and multiprocessor systems.

— It supports multiple consistency models providing the flexibility to an application in choosing the suitable consistency model.

— An Improved Lazy Release Consistency model is introduced which is observed to be an efficient model.

— Memory consistency related events are processed concurrently and processing of several such events that are generated by different threads can be in progress simultaneously.

— A simple two-layer efficient communication protocol supports faster transmission of messages.

— The IDSM system supports PThread, MPI and OpenMP [Board 1998; Lu et al. 1998] programs without any modifications.

— The IDSM system supports mixed mode programs also, that uses both message passing and shared memory features.

— Compared to TreadMarks DSM system, IDSM uses very small amount of memory for maintaining the information about each shared page and hence IDSM is much more scalable. According to the design of process, if sufficient memory is available thousands of threads can be supported.

2. BACKGROUND

In this section, features of two popular software DSM systems - TreadMarks and JIAJIA, are presented. TreadMarks uses Lazy Release Consistency (LRC) model where as JIAJIA uses Scope Consistency model.

2.1 TreadMarks

TreadMarks, distributed shared memory system [Amza et al. 1996] is developed at Rice University. It runs at user level on network of UNIX workstations without any additional privileges. In LRC model [Carter 1994; Keleher et al. 1992], that is used in TreadMarks system, when a processor acquires a lock and access the data item which is invalid, then only it requests for the updates of the corresponding data item from its recent modifier. Thus only the required processor gets the recent modifications for the necessary data items. In this model number of consistency messages are much reduced compared to original eager release consistency model. In TreadMarks implementation of LRC [Keleher et al. 1992], piggybacking of notification of modified

pages onto the lock granting message is done for reducing the number of messages. TreadMarks uses multiple writer protocol to handle the false sharing and to increase the concurrency. There are some limitations for LRC model implementation, which are described in Section 3.5.1, and improved in Improved Lazy Release Consistency model (ILRC) model.

2.2 JIAJIA

JIAJIA [Hu et al. 1999] is a home-based software DSM system and it implements lock based cache coherence protocol which does not require a directory. It maintains coherence through accessing write-notices kept on the lock. Compared to directory based protocols, lock based protocols are simpler and hence more efficient and scalable. JIAJIA supports Scope Consistency model [Tanenbaum 2003; Steinke and Nutt 2004; Adve and Gharachorloo 1996], which is even better than LRC model [Hu et al. 1999]. Multiple writer technique is employed to reduce false sharing. When a processor acquires a lock, intervals after the last acquire, related to only that lock are received along with the lock granting message. On a release, the processor performs a comparison of all cached pages modified in the critical section with their corresponding twins [Hu et al. 1999; Keleher et al. 1992] to get page differences. These are sent to their associated homes. Release message and write notices are sent together usually to reduce the number of messages. Compared to LRC protocol the number of messages required for lock acquire operations and normal data accesses are less in Scope Consistency model [Hu et al. 1999]. But JIAJIA requires sending page differences back to their respective homes of the associated pages on a release operation. It is free from the overhead of maintaining the directory.

3. DESIGN

The proposed IDSM system architecture is shown in the Figure 1. It is designed as a multi-threaded kernel. The lowest level is hardware abstraction layer that consists of low level memory management, low level task management, basic device drivers and scheduler. Low level memory management module is responsible for core allocation and freeing. Task management module provides processor level task related functionality such as hardware context creation, context switching etc. Low level device drivers provide very basic functionality to make use of the devices such as clock, storage, NIC etc. Scheduler is responsible for scheduling the low level tasks using priority based round robin scheduling algorithm. At the second level local process, virtual memory, file system and communication management services are provided. A local process contains address space and a set of kernel level threads. Local process management module provides functionality for creation, termination and waiting/joining of processes and threads. Virtual memory management module provides services for paging based memory operations such as memory segment creation, expansion and page table initialization and page attribute manipulations etc. Communication system is specially designed for efficiency using simple two layer reliable packet delivery protocol. Using these second level services in the third level, sequential process and parallel process interfaces are provided to the application programs. Parallel process interface provides services for remote thread creation, termination, distributed synchronization related locks, barriers, conditions and a set of memory consistency models. Some of the important modules of this system are described in the following subsections.

3.1 Communication Management

The communication module is designed to suit the needs of both common networking applications as well as parallel applications. At the lowest level in the communication software, Ethernet frame delivery service is provided. On top of this, reliable packet delivery protocol is developed. Using this reliable packet delivery service, different types of high level communication services such as stream services, datagram services, multicasting within a process group and broadcasting among the similar type of processes of different nodes are provided. Reliable packet delivery service is capable of delivering various types of packets between two end points. End points used at this level are low level internal sockets which are identified with their type, node number and port

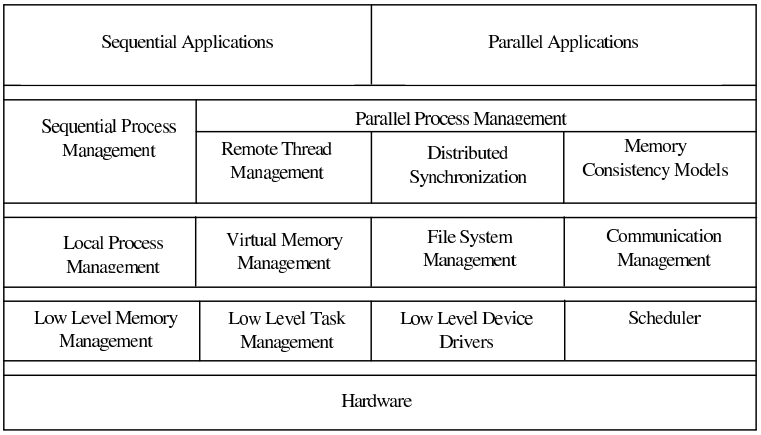

Figure 1 : Layered Architecture of IDSM Systsm.

number. Type can be either stream type, datagram type or default type. A default socket is created for each thread of the parallel process. The reliable packet delivery protocol is based on sliding window protocol and it has the functionality for eliminating duplicate packets, for retransmitting the lost packets and for flow control. It makes use of checksums for detecting errors in the received packets. Using this reliable packet delivery service two types of interfaces are developed namely - socket interface and parallel process communication interface. Socket interface is similar to that of BSD socket interface, which can be used to support message passing and networking applications. Parallel processing interface contains only two primitives - *send* and *receive*.

*int send(int desttid, char *buf, unsigned int len, int groupid, int tag, short flags);*
*int receive(char buf[], int maxlen, int *fromptr, int flags, int msgtype, int groupid,*
*int *tagptr, int waitmsec);*

This interface is useful for explicit communication among the distributed threads of a shared memory process. The *send* primitive provides functionality for sending a message to a particular thread (Unicast) or to all threads of that process (Multicast) or to all similar daemon threads (Broadcast) of all nodes. The *receive* primitive provides functionality for receiving the messages either in the order of arrival or selectively based on the source and type of the message. Using these basic communication functions all other types of libraries such as BSD socket functions and MPI communication functions are developed.

3.2 Local Processes and Global Processes

The proposed DSM system supports two types of processes namely Local Processes and Global Processes. Local process is bound to a single node and it has only local threads. Global process is bound to many nodes and has both local threads and remote threads. For increasing the level of concurrency / parallelism in an application the system supports kernel level threads. The system can accommodate large number of threads in each process. But in practice, there is no need for having large number of threads than the available number of processors, as it increases the overhead there by increases the execution time of the parallel process. (This is true if sufficiently large number of processors are available. But this does not hold when very small number of processors are used than the available degree of parallelism). Once a global process is created, a new thread can be created in that process on any one of the available nodes. The selection of the available node can be based on current load conditions.

3.3 Name Server and Remote Thread Execution Server

Name server is responsible for allocating unique global process and thread identifiers. When a global process is being created, the home node registers its name with the name server and obtains unique global process identifier. Name server allocates the stack regions for the global threads. Requests for remote thread creation, termination, monitoring etc. are handled by the remote execution server. Remote execution server uses the services of local process module.

3.4 Synchronization Module

As this platform supports parallel and distributed applications there is a need for synchronization among the local processes as well as global processes. For the applications, through PThread interface - mutex objects, condition objects and barrier objects are provided for synchronization. Another common interface, POSIX semaphores, which is used in some standard parallel applications, is also provided. The same PThread and POSIX Semaphore interfaces can be used for distributed synchronization of parallel applications transparently.

3.5 Memory Consistency Models

For maintaining the consistency of replicated data in DSM systems several memory consistency models are proposed in the literature. Programmer must consider the type of memory consistency model supported by a particular shared memory system. The IDSM system provides several consistency models such as Sequential, PRAM, Causal, Weak, Entry and Lazy Release memory consistency models. LRC model is one of the efficient consistency models. But there are some limitations in its implementation in TreadMarks DSM system and in order to overcome those limitations a new Improved Lazy Release Consistency model has been proposed and implemented in IDSM system. Due to lack of space comparative performance of only LRC and ILRC models is presented in this paper.

3.5.1 *Limitations of LRC Model.* There are several limitations in the LRC implementation, when applied to parallel applications. If the application requires very large data which is modified frequently, then for an interval, size of page differences represented as write notice records may exceed the size of original page and hence LRC is not suitable for large data arrays. Even if smaller differences are there, collectively for a number of processes, which might acquire the lock in sequence before the current process, the total size of page differences together may exceed the size of original page. Computation of page differences and write notice record organization is shown in Figure 2. In this diagram a twin page and its use for creating page differences [Keleher et al. 1992] along with three write notice records are shown. At that point of time if a process makes a request for that page, then reply message to be sent, containing packed list of three write notice records (linked list shown at the bottom of the figure) together definitely exceeds the size of the page. The original LRC implementation is useful if one or few scalar variables are present in a page that is modified in a critical section, where the resultant page differences are smaller. The amount of space utilized by twin pages, which need to be maintained for longer periods in case page differences are not requested immediately, is an additional overhead to the write notice records, interval records and page differences. Computing page differences upon the requests made by other processes is time consuming and also response to such requests gets delayed. At the same time computing the page differences in advance might result in wastage of computing time in case if such page differences are not requested in future. ILRC model overcomes these memory space and CPU time problems.

3.5.2 *Improved LRC Model.* In the proposed ILRC model all the variables of the source program are divided into two categories viz., small and large data items. Synchronization wise related sets of small data items are kept together in one page. Different such small data sets are placed into different pages to eliminate false sharing. The compiler/linker includes the size element for each such set. This size indicates amount of memory usage for the variables in that

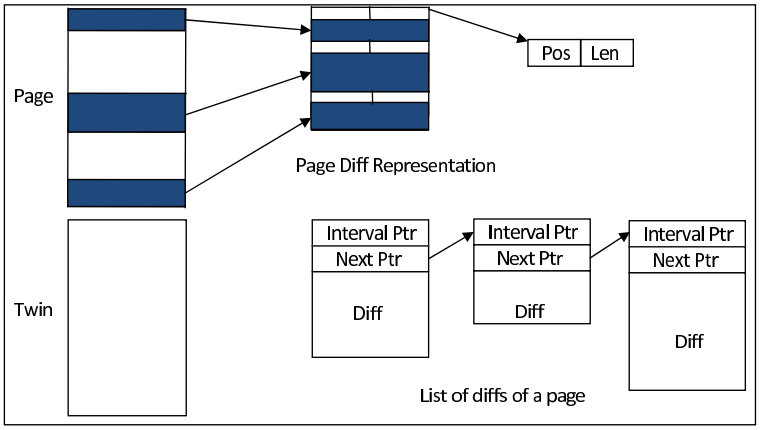

Figure 2 : Demonstrating How Multiple Page Diffs Together Exceeds The Size Of The Original Page.

page. This implementation might result in wastage of space in a page. However, in any parallel application existence of such type of small data item sets is generally low, there by resulting in limiting the total wastage of space. For large data items that may span over multiple pages, no additional information is required. Those pages are treated with simple write-invalidation protocol. This method partially eliminates the need for computing page differences and maintaining write notice records, page differences and interval records. But for each page it maintains type of the page, page manager, last acquired interval and state information which occupies a little space. This is a clear advantage in a parallel application that requires huge amounts of memory. This feature along with the process model which can facilitate large number of threads, allows the system to be more scalable compared to other DSM systems. All the related small variables that can be requested at a time are maintained in one page resulting in reduced communication. This requires explicit annotations in the source program (using attribute directives of GCC) to keep those sets of related small data items in separate pages. IDSM system uses its own type of elf executable file format, which contains an additional linker map section. When the $exec()$ system call of parallel process reads such an executable file it reads the linker map section and initializes the information about the pages appropriately. (Such a linker map is definitely required for supporting entry consistency model in IDSM system.) When a process performs an acquire (may be lock acquire or returning from barrier) operation, the process gets the list of current managers of the recently modified pages after the last acquire operation. The acquired process modifies its data structures to indicate the managers for each such modified page. For the pages which have not been modified since last request, for those pages, the processor is assumed to have up-to-date copies of those pages. If an older page is referenced, then a page fault occurs. The page fault handler determines whether the page is small data item set page or large data item set page. It sends a partial page request or full page request to the manager of that page depending on the page type. It can also request for ownership (page manager) if the faulted instruction is a write operation. In this approach the main limitation is, simple programming interface of original LRC is violated by introducing programmer annotations. Moreover providing annotations in the source program is found to be a difficult and error-prone task. Another limitation is for handling false sharing, while LRC is following an efficient approach based on page differences, ILRC follows division of synchronization wise unrelated data items into separate pages which require input (annotations) from the programmer. Combining the advantages of handling of false sharing problem and simple programming interface of LRC and memory and time efficiency of proposed ILRC, another algorithm is proposed based on automatic insertion of directives for splitting the simple variables into separate pages.

3.5.3 *Another Algorithm for ILRC.* If a page is containing multiple small unrelated data items that may be simultaneously processed by several threads, then same page may be required for all of those threads at the same time. If multiple writers are allowed to modify the same page, then assigning a page manager to a page based on the last writer approach is not possible. For solving this in LRC one of them is chosen to be the page manager and page differences are taken from each writer thread when ever up-to-date copy of the page is required. In this algorithm programmer annotations are not required. There are three types of pages identified in this algorithm namely - (i) Multiple writer small data item set pages (ii) Single writer small data item set pages (iii) Large data item set pages. From the linker map section of executable file, small data item set pages and large data item set pages can be separated while the parallel process is being created. If a page contains small data items and some part of a large data item (may be mixed because of no annotations are provided), then together such a page can be treated as small data item set page, without affecting the working of the ILRC algorithm. During the runtime, IDSM system classifies the small data item set pages as single writer or multiple writer small data item set pages according to their access pattern. After an acquire operation on a particular synchronization object, if some pages are requested then like in Scope Consistency model, those pages are associated with the corresponding synchronization object. If the same page is associated with two or more synchronization objects, then it can be treated as containing multiple small data items that are protected with different synchronization objects. Such a page is classified as multiple writer small data item set page, which is treated using page difference approach of LRC model. Based on the overlapped intervals also such a multiple writer small data item set page can be identified. Whenever a page is accessed by multiple threads for writing with overlapped intervals then that page can be classified as multiple writer small data item set page. Intervals that are created by acquire and release operations on the same synchronization object can be totally ordered by following $happened-before$ [Keleher et al. 1992] relation. Intervals created by the same processor can also be ordered according to the processor order using $happened-before$ relation. Any two intervals whose order cannot be determined using $happened-before$ relation are said to be overlapping intervals. If a page manager receives a request for the same page for writing within two or more overlapped intervals then it can send a hint along with the page copy indicating that the page must be treated as a multiple writer small data item set page. Initially all small data item set pages are assumed to be of single writer type. During the runtime multiple writer small data item set pages are identified by following one of the above described procedures. For the other remaining small data item set pages, their respective small data item sizes are computed using linker map information, using that partial page requests can be generated. It is assumed that either manually (annotations) or automatically by the preprocessor the small data items are split into separate pages by introducing align attribute in the data declarations for the effective use of the ILRC algorithm. Definitely automatic splitting of small data items into separate pages may not be better than manual splitting through annotations, in which related small data items can be optimally grouped into one page. Thus based on the access patterns along with the linker maps, different types of pages can be identified at runtime dynamically and the corresponding efficient approaches can be followed for handling the requests and events. Advantages of second method to classify the pages include - programmer annotations are not required, false sharing is handled and large data arrays are handled efficiently. When the program is consisting of only small data items and containing only fine-grained parallelism then, without proper annotations, the ILRC algorithm converts down into normal LRC algorithm.

Algorithms for handling some important memory consistency related events such as page fault handling, lock acquire operation, lock request processing and lock release operation are presented in the following Figures 3, 4, 5, and 6 respectively.

The choice of consistency model is generally decided by the application requirements. Use of any consistency model in IDSM system is fully transparent to the application, i.e., once the consistency model is selected, automatically system makes calls to the consistency model specific

```
...
...
// Valid data reference
if (p READ_ONLY)
{
    if (p is multiple items page)
        create twin

    change flags of the page as READ_WRITE
}
else
{
    if page table entry of p is not existing
    {
        // New complete copy required

        Allocate a page
        Modify the flags indicating the presence of the page
        request entire valid part of the page
    }
    else
    {
        // Old copy of the page is present

        if (p type is normal small data item set)
            send partial page request
        else if (p type is multiple writer small data item set)
            send page diffs request
        else if (p is large data set)
            send full page request.
        // Reply is received and processed.

        modify the page flags depending on reference type
    }
}
```

Figure 3 : Algorithm For Page Fault Handling

```
if (lock available locally)
{
    held = true; // This is the last acquirer
    return
}
else
{
    Send request with VV_a to lock manager
    Create new interval
    Wait for grant
    for each interval x covered by send_set
    {
        for each write notice covered by x
        {
            if page p READ_ONLY then
                Change p to INVALID
            else if page p READ_WRITE
            {
                remove from dirty list
                if (p is multiple writer small data item set)
                {
                    create write notice and page diff if necessary
                    deallocate the twin page
                }
                Change page p to INVALID
            }
        }
    }
}
```

Figure 4 : Algorithm For Lock Acquire Handling

```
if (local) then
{
    if (held) then
        save VV_a
    else
    {
        Create new interval
        Prepare send_set
        Send send_set and lock grant to requester
    }
}
else Forward request to last requester
```

Figure 5 : Algorithm For Lock Request Handling

```
Create new interval
Prepare dirty page list
Associate dirty list with the interval
held = FALSE
if (there has been a request) then
{
    prepare send_set
    Send lock grant and send_set to requester
    local = FALSE
}
```

Figure 6 : Algorithm For Lock Release Handling

functions in the necessary situations without any additional effort of the application. Multiple consistency models in this system are supported through the common interface that consists of set of operations as shown in Figure 7. This common interface links the remaining system and memory consistency module. The interface contains a set of function pointers to handle memory consistency model related events. These function pointers refer to the model specific event handling functions. Whenever a memory consistency related event such as page fault, lock or unlock happens, through the common interface object, specific event handler function of memory consistency model is called automatically. The sources of the memory consistency related events may be locally running threads or remote threads in which case they are received as remote requests, or may be replies for the previous locally generated requests. Some of the event handling functions of the interface may not be relevant for some memory consistency models. In such cases those unrelated function pointers refer to dummy (empty) functions.

The functionality of the memory consistency manager is divided into logically separate parts to reduce the complexity and to increase the concurrency, and each part of the functionality is implemented in separate handler threads. As the processing of memory consistency related events is subdivided into phases and multiple events are processed concurrently, more parallelism is available for the threads of an application.

4. PARALLEL PROCESS MODEL

The given parallel program starts its execution on the initiated system, known as the home machine for that process. The main thread of the process spawns the necessary number of threads to do the computation in parallel. These child threads can be created on other nodes for load balancing. Address space of the process consists of code, data, heap, stack and thread private memory regions. Data, heap and stack regions are shared across all the nodes and code region is fully replicated. Thread private region is not shared across the nodes. The common usage of such private memory can be found in threads that exchange information using explicit communication operations. If a thread wants to communicate explicitly using *send* / *receive* calls, without interfering with other thread's send / receive buffers, it can allocate its own buffer from the private memory region.

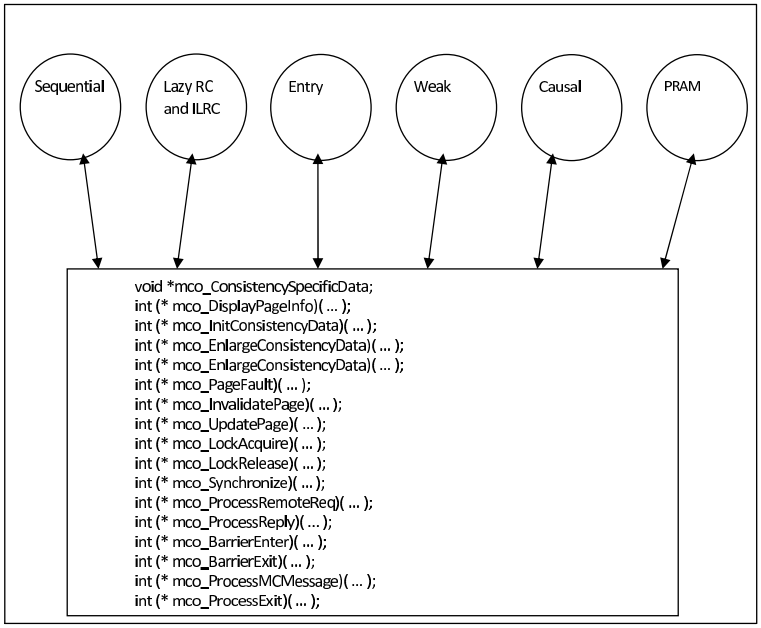

Figure 7 : Generic Interface of Memory Consistency Models.

5. IMPLEMENTATION

The proposed system has been implemented for a network of X86 based systems. The system has been implemented in C and Assembly languages. This has been cross compiled on a Linux machine using GNU C compiler. This system is developed as a stand-alone system. In the initial stages of development VMWare Workstation tool is used for testing the developed system. VMWare Workstation can emulate many of the systems including Intel X86 based PC. It is a small and very convenient tool that can run on many platforms like Linux, Windows, Solaris etc., and is particularly useful for testing the new Operating Systems. VMWare makes the IDSM system to run on any system under the virtual environment, so that separate installation is not required. Thus IDSM system becomes highly portable.

6. RESULTS

A total of three sets of programs are taken for comparative study of the IDSM system. First set contains SPLASH-2 benchmark programs for shared memory systems, second set contains Intel MPI Benchmark (IMB) programs and third set contains NAS Parallel Benchmark (NPB) programs for MPI. In addition to these programs common parallel processing algorithms such as Reduction, Sieve (for finding prime numbers), Matrix multiplication, Merge sort, Quick sort [Quinn. 2002] and Traveling Salesperson Problem (TSP), are also included for comparison. Description of the benchmark programs can be found in [Woo et al. 1995; Corporation 2006; Bailey et al. 1994]. Input data set size of each of the programs is shown Table I. As OpenMP enabled GCC compiler uses the thread library at the back-end, separate set of OpenMP programs is not considered (PThread programs only are considered) for testing the system.

The statistics are compiled using VMWare Workstation software. Time values of the VMWare Workstations are scaled based on the clock frequencies of the processor (obtained using GETTSC - Get CPU Time Stamp Counter instruction). Clock cycles count is used as a measurement of time, because in a virtual system processor, real time cannot be used for time measurement and further clock cycle count represent standard and high precision accurate time. All of the programs have been studied using 8 nodes, except for BT and SP of NPB, for which 9 nodes are

Table I: Data Set Sizes of the Applications Used for Comparison

Application	Data set size of app.	Application	Data set size of app.
SPLASH-2 and Common Applications			
Reduction	8388608	Sieve	4194304
Matmult	512x512	Merge	5242880
Quiksort	5242880	TSP	20
Bucketsort	4194304	LU	512x512
Radixsort	2097152	Barnes	16384
FMM	16384	Ocean-cont	258x258
Ocean-non	258x258	Water-nsq	512
Water-spat	512	FFT	65536
NPB Applications			
BT	24x24x24	EP	67108864
FT	128x128x32	IS	1048576
LU	33x33x33	MG	128x128x128
SP	36x36x36		

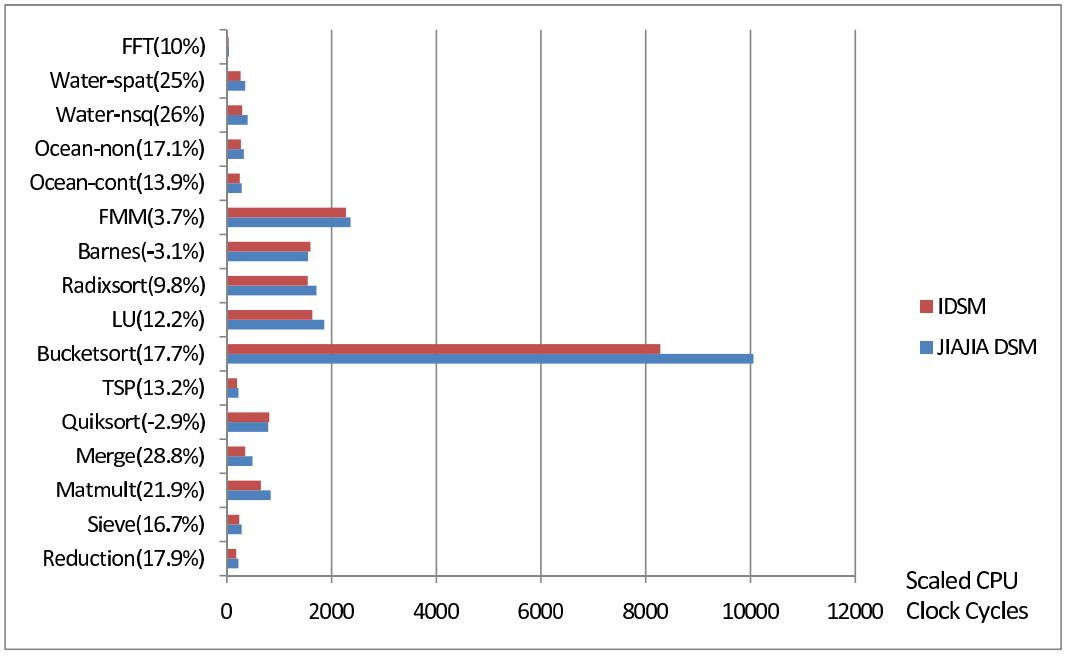

Figure 8 : Performance of Shared Memory Programs (Common Parallel Algorithms and Splash-2 Benchmark Programs).

used.

The comparative performances of shared memory, NPB and IMB programs are shown in Figures 8, 9 and 10 respectively. In those figures values shown inside the braces indicate percentage of improvement. Except for two cases for shared memory programs new system has performed better than Scope consistency based JIAJIA library on Linux systems. For shared memory programs percentage of improvement ranges from 3.7% (FMM of SPLASH-2) to 28.8% (Merge sort).

For MPI programs as the light weight two-layer communication protocol is used, the performance of MPI implementation of IDSM system is observed to be improved much better compared to Linux based MPI implementation (MPICH2). Actual performance improvement ranges from

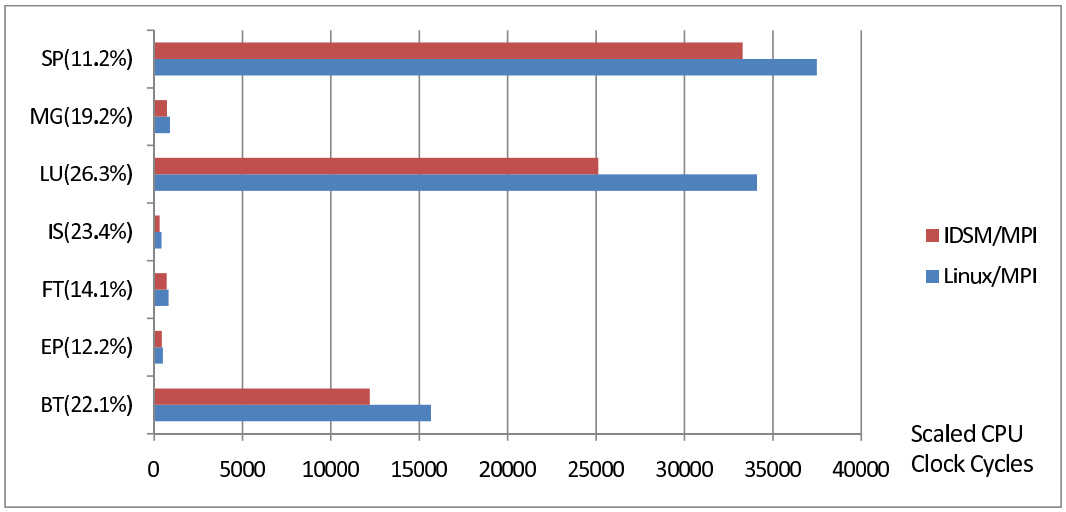

Figure 9 : Performance of NPB Programs.

11% (SP of NPB, Figure 9) to 37.1% (Bcast of IMB, Figure 10).

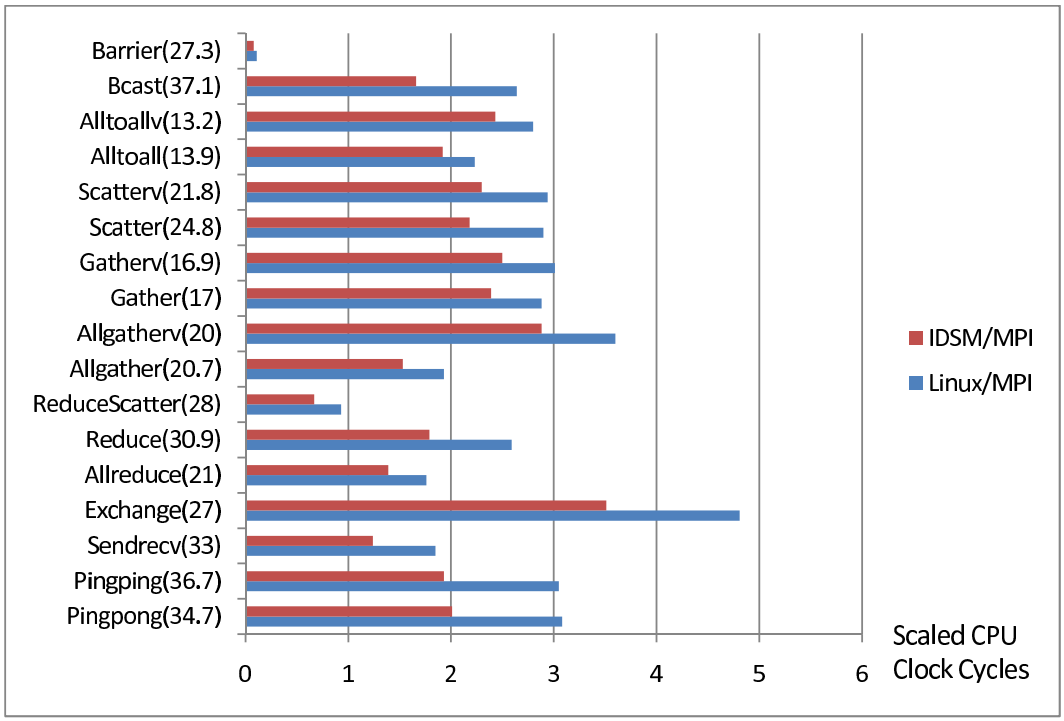

Figure 10 : Performance of IMB Programs (Data Set Size: 64KB and 2 processes involved, where ever pairs are required).

Comparison of LRC protocol of TreadMarks system (which is ported onto IDSM system) and ILRC is performed using SPLASH-2 benchmark programs and some common parallel algorithms and the corresponding results are shown in Figure 11. ILRC has performed consistently better than LRC and an improvement of 1.9% to 32.2% is observed. Hence these results prove that the

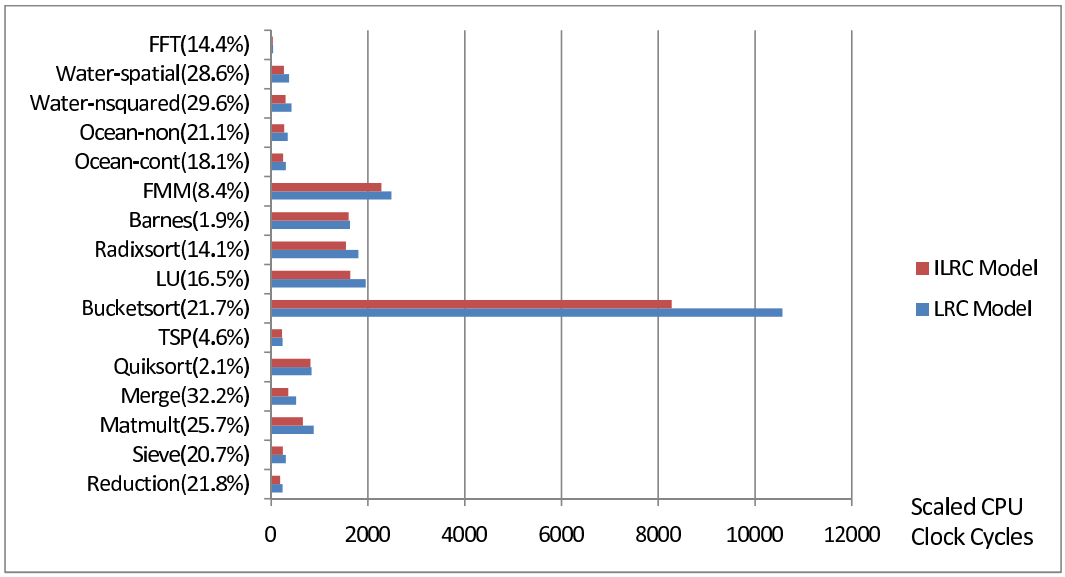

Figure 11 : Performance ILRC and LRC Compared (Both are compared under IDSM).

new model of DSM system along with ILRC model and light weight communication protocol is really useful in developing parallel applications for network of workstations.

Mixed mode programming is demonstrated using block matrix multiplication and TSP problems. In block matrix multiplication multithreading is used as in a shared memory program and each thread is responsible for computing a set of consecutive rows of the product matrix. For computing those rows, a thread requires the same set of consecutive rows of multiplicand matrix and the complete multiplier matrix. In the shared memory version, a thread obtains the required rows of the multiplicand matrix and the columns of the multiplier matrix automatically, whenever the corresponding elements are referred. In the mixed mode version, those rows and columns are obtained explicitly by using communication primitives. In shared memory version, multiplier matrix is transferred $P-1$ times from home node to other $P-1$ nodes of the system upon receiving corresponding memory consistency requests from other nodes. But total multiplier matrix can be transferred using single broadcast operation in mixed mode version, which reduces the count of messages required. There are no page fault related page copy request messages. Not only that the product matrix rows need not be transferred in both directions (from home node to other nodes and back from other nodes to home node) in mixed mode version. Threads on other nodes (other than home node) computes the rows of product matrix in their own private memory and at the end, those rows are sent to home node, which requires transfer of rows of product matrix in one direction only. Thus using the ease of shared memory programming in creating the threads and using efficiency of message passing in matrix data transferring, an improved mixed mode version of block matrix multiplication application is developed under the IDSM system.

Similarly in the case of TSP problem all data of the program is shared, except the current known best global bound value which is a scalar variable, whose value is frequently referenced by all the threads of the process. As it gets modified frequently, there is a heavy contention for that variable under a DSM model. This is avoided in mixed mode version by creating that global bound variable dynamically inside the thread private memory whose value is updated by only the corresponding thread. Whenever a new bound value is found by a thread, it is broadcasted to other threads, using that other threads can update their private copy. Thus mixed mode version of TSP program reduces the amount of communication. Performance of mixed mode versions of the block matrix multiplication and TSP problems is shown in Figure 12. Improvements of

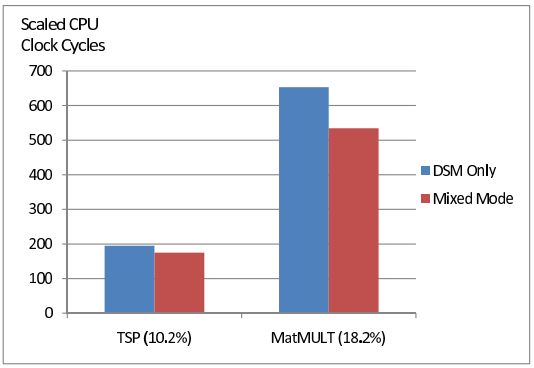

Figure 12 : Mixed Mode Program Performance Compared To DSM Only Program (Both are Compared Under IDSM System).

10.2% and 18.2% are observed for TSP and matrix multiplication programs respectively. The results are proving the effectiveness of mixed mode programming.

7. CONCLUSIONS AND EXTENSIONS

A new software based DSM system, IDSM system, has been developed and its performance is studied. In most of the cases the new system performed much better than the known other systems. Such a kind of system which supports multithreading over network of workstations transparently along with special message passing features is very essential for the effective usage of multicore based network of workstations. The main goal of the new system is to support the existing multiprocessor based PThread programs or network based MPI programs without any modification to the source code. At present the two-layer communication protocol supports only a LAN. It should be improved to support much larger networks connected with routers or gateways.

ACKNOWLEDGMENTS

We wish to thank MHRD, INDIA for their sponsoring of the R & D project titled "Implementation of Software Based Distributed Shared Memory System Using Page Based Memory management System".

REFERENCES

Adve, V. S. and Gharachorloo, K. 1996. Shared memory consistency models: A tutorial. *IEEE Computer 29,* 12 (Dec.), 66–76.

Ahuja, S., Carriero, N., and Gelernter, D. 1986. Linda and friends. *IEEE Computer 19,* 8 (May), 110–129.

Amza, C., Cox, A. L., Dwarakadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W., and Zwaenepoel., W. 1996. Treadmarks: Shared memory computing on network of workstations. *IEEE Computer 29,* 2 (Feb.), 18–28.

Bailey, D., Balszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. 1994. *The NAS Parallel Benchmarks.* NASA, USA, RNR Technical Report, RNR-94-007.

Board, O. A. R. 1998. *OpenMP C, C++ Application Program Interface.* OpenMP Architecture Review Board, http://www.openmp.org.

Carter, J. B. 1994. Efficient distributed shared memory based on multi protocol release consistency. Ph.D. thesis, Rice University., USA.

Corporation, I. 2006. *Intel MPI Benchmarks - User Guide and Methodology Description.* Intel Corporation, Intel Corporation Document Number :320714-001.

Delp, G., Farber, D., Minnich, R., Smith, J. M., and Tam, M. C. 1991. Memory as a network abstraction. *IEEE Network 5,* 4 (July), 34–41.

Fleisch, B. and Popek, G. 1989. Mirage: A coherent distributed shared memory design. In *Proceedings of the 12th ACM Symposium on Operating System Principles.* ACM, New York, USA, 211–223.

Gustavson, D. B. 1992. The scalable coherent interface and related standards projects. *IEEE Micro 12,* 1 (Jan.), 10–22.

Hu, W., Shi, W., and Tang., Z. 1999. Jiajia: A software dsm system based on a new cache coherence protocol. *High-Performance Computing and Networking, HPCN, Lecture Notes in Computer Science, 1593 1593,* 1 (Apr.), 463–472.

Keleher, P., Cox, A. L., and Zwaenepoel., W. 1992. Lazy release consistency for software distributed shared memory. In *In Proceedings of the 19th Annual International Symposium on Computer Architecture*. ACM, New York, USA, 13–21.

KSR. 1992. *Kendall Square Research Technical Summary*. KSR Corporation, Waltham, Massachusetts, U.S.A.

Lenoski, D., Laudon, J., and XXXXXX. 1992. The stanford dash multiprocessor. *IEEE Computer 25,* 3 (Mar.), 63–79.

Li, K. and Hudak, P. 1989. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems 7,* 4 (Nov.), 321–359.

Lu, H., Hu, Y., and Zwaenpoel., W. 1998. Openmp on networks of workstations. In *Proceedings of supercomputing'98*. ACM/IEEE, IEEE Computer Society Washington, DC, USA, 1–15.

Protic, J., Tomasevic, M., and Milutinovic, V. 1995. A survey of distributed shared memory systems. In *Proceedings 28th Annual Hawaii International Conference on System Sciences*. HICSS, http://csdl.computer.org/comp/proceedings/hicss/1995/6930/00/69300074abs.htm, 74–84.

Quinn., M. J. 2002. *Parallel Computing Theory and Practice*. Tata McGraw-Hill, Delhi, INDIA.

Ramachandran, U. and Khalidi, M. Y. A. 1991. An implementation of distributed shared memory. *Software - Practice and Experience 21,* 5 (May), 443–464.

Steinke, R. C. and Nutt, G. J. 2004. A unified theory of shared memory consistency. *Journal of the ACM 51,* 5 (Sept.), 800–849.

Tanenbaum, A. S. 2003. *Distributed Operating Systems*. Pearson Education, Delhi, INDIA.

Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta., A. 1995. The splash-2 programs: Characterization and methodological considerations. In *In the proceedings of 22nd Annual International Symp. On Computer Architecture*. ACM, New York, USA, 24–36.

Zhou, S., Stumm, M., and Mcinerney, T. 1990. Extending distributed shared memory to heterogeneous environments. In *Proceedings of the 10th International Conference on Distributed Computing Systems*. ICDCS, ICDCS, 30–37.

T. Ramesh is a professor in the department of CSE, NIT Warangal having 25+ years of experience in teaching. He is HOD of CSE. His areas of interest include - Parallel processing, Compilers, Grid computing and Algorithms. Worked as pricipal investigator for many R&D projects sponsored by Govt of INDIA and Industry.

Chapram Sudhakar is a lecturer in the department of CSE, NIT Warangal having 14 years of experience in teaching. His areas of interest include - Parallel Processing, Distributed Systems, Operating Systems, Compilers. Worked as Project Assistant and Co-Invistigator for two R & D projects sponsored by Govt of INDIA.