December 1999
Cs240b Project Report

PTMPI

Threaded MPI Execution on Cluster of SMP machines

Zoran Dimitrijevic
zoran@cs.ucsb.edu

Department of Computer Science
University of California at Santa Barbara

Abstract

This paper presents the design and performance evaluation of multithreaded standard MPI library implementation for cluster of SMP machines. The paper proposes shared memory communication between the MPI nodes running on the same SMP machine and socket communication only between the processes on the different machines. The performance evaluation is performed and performance improvements over standard MPI [MPICH] implementation are presented at the end of the paper.

1. Introduction

Cluster of SMP machines has become the attractive platform for scientific and parallel computing. Each node in the cluster is SMP shared memory machine, and communication between the nodes is usually through the fast network that provides TCP/IP.

Standard MPI implementation is not optimized for cluster of SMP machines, since each MPI node is implemented as separate operating system process, and communication between the MPI nodes on the same SMP machine is inefficient.

Better performance can be achieved if each node on the SMP machine is just a thread and not separate process. Paper about threaded implementation of MPI for shared memory machines [TMPI] proposes that each MPI node should be thread inside one process, which provides significant performance benefit.

The multithreaded MPI implementation is possible only on the globally shared memory machines. Cluster of SMP machines does not provide shared memory between the nodes, and exclusive shared memory communication is not possible.

Significant performance benefit can be achieved if communication between the MPI nodes running on the same SMP machines is through shared memory, and between the processes only if the processes are running on different SMP machines in the cluster.

2. Problem Statement

Standard MPI implementation does not take advantage of the cluster platform. Exclusive multithreaded implementation is not possible since cluster does not have shared memory. Goal of this research is to develop the library and examine performance benefits over the standard MPI implementation used on clusters now, if the combined approach is used.

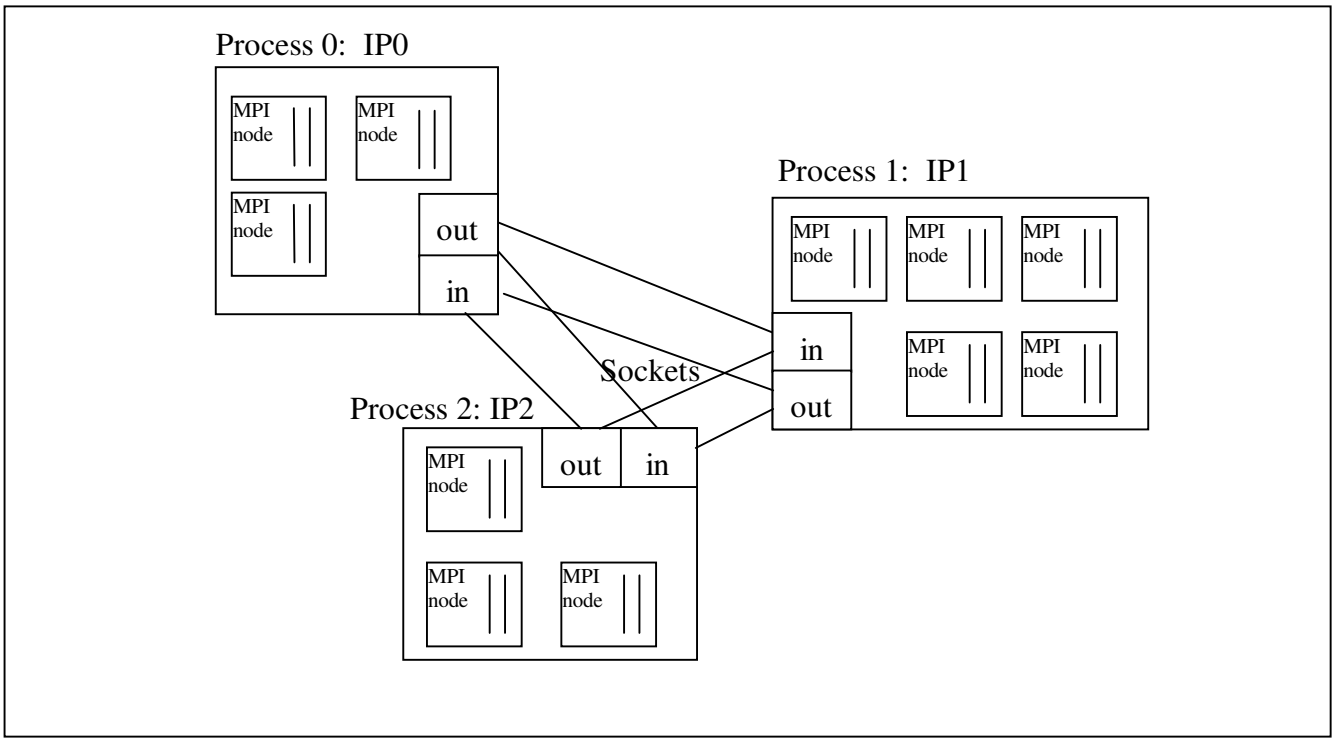
3. Proposed Solution

MPI implementation presented in this paper is shown on Figure 1. System consists of several processes. Each process can run on different machine. Communication between the processes is through standard sockets, and the interconnection network can be anything that provides socket interface.

Each MPI node is a thread inside a process. Communication between the MPI nodes inside the same process is trough queues in shared memory. Each process has two communicators threads, which provide the communication between the processes in the system.

During the system startup the processes are created, and complete graph of sockets is established for inter-process communication. This approach is used since the overhead of communication through the socket is minimized. There are p(p-1)/2 sockets in the system after startup procedure, where p is the number of processes in the system. Each process can have different number of MPI node threads running inside it, and two additional threads for in and out socket communication.

Each process creates one thread per each MPI node to be run inside it, and creates the new instance of the class MPI_Node inside the thread. All global data are duplicated for each thread since MPI standard provides SPMD paradigm. Each MPI node thread calls mpi_main function and the actual computing is started.



Layout of the process in the system is shown on Figure 2. In communicator receives messages from other processes in the system and dispatches them to destination thread. There are two queues per each MPI node thread.

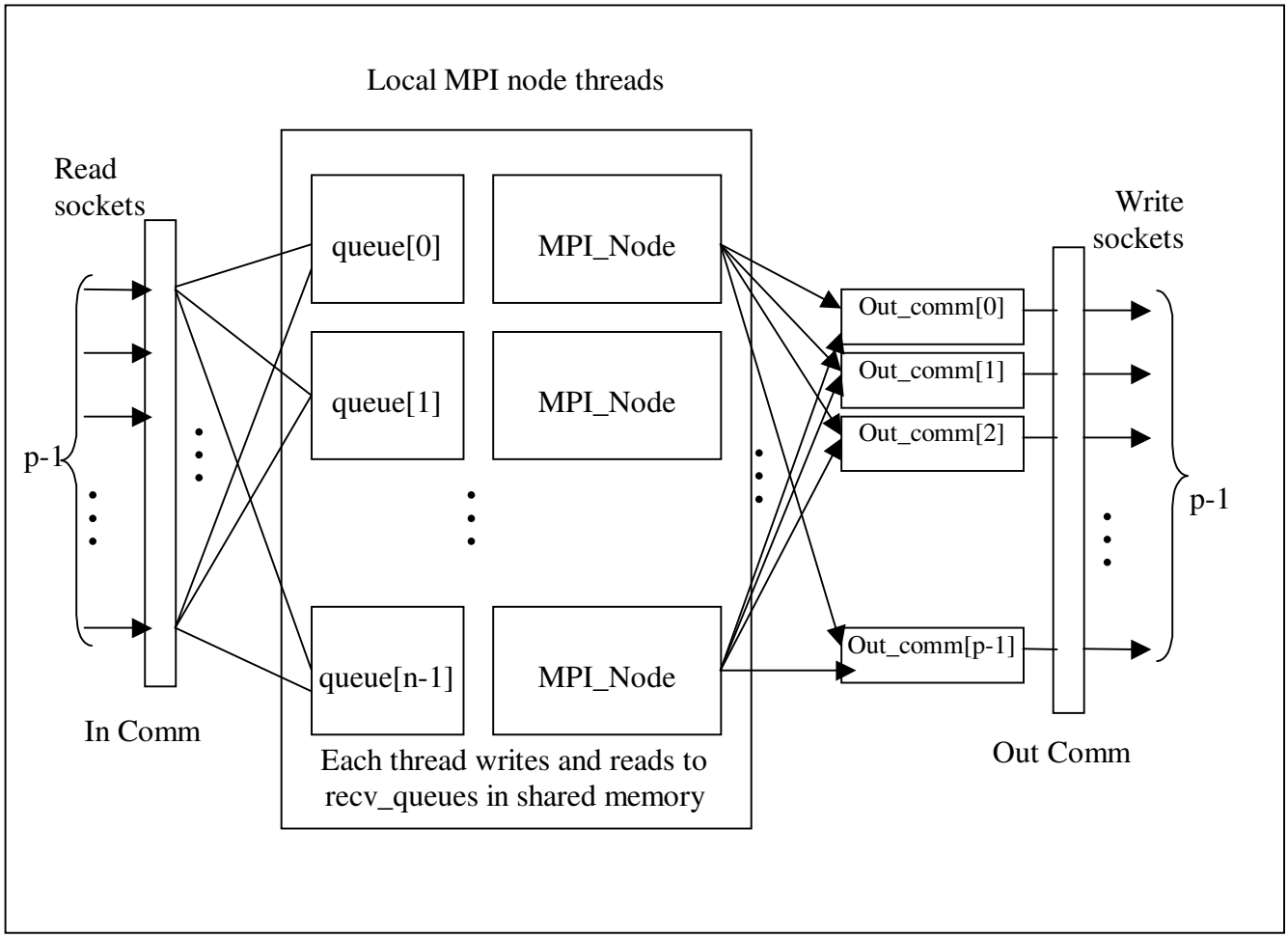
One queue is used for storing receive request issued by that thread before message arrival.

The other queue is for storing the information about the messages arrived before MPI node thread issued the receive request.

If the send is buffered the message is copied to the buffer which will be deallocated by the receiver receive function. If the send is not buffered the receiver receive function will signal sender when the message is copied to receiver local memory.

Messages to the MPI node inside the same process are managed by the sender thread, and all messages for non-local MPI node (running inside different process) are stored on the queue for that process and to be send by the out communicator thread. Out communicator thread sends all messages on the outgoing queues through appropriate socket to the destination process.

Broadcast request sends the message to each MPI node thread, and to all outgoing queues. In communicator sends the broadcast message to all local MPI node threads, and since the communication is through the shared memory, the unnecessary copy is avoided.



4. Initial Performance Evaluation

Following MPI functions are implemented: MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize, MPI_Send, MPI_Isend, MPI_Recv, MPI_Irecv, MPI_Wait, MPI_Bcast, MPI_Barrier, andMPI_Wtime.

Block based matrix multiplication program is used for performance evaluation. Matrix size of 1024 x 1024 and 2048 x 2048 were used, and block sizes of 16, 32, and 64.

Communication between the MPI nodes is achieved using broadcast mechanism. Program was compiled using MPICH and PTMPI, and run on UCSB Gargleblaster Cluster, using 4 processor and 2 processor SMP nodes. Each two-processor node is Pentium II 400MHz with 512MB or 1GB main memory, and each four-processor node is Pentium III 500MHz with 1GB main memory.

Figures 3-6 show performance improvement in terms of execution time between the MPICH and PTMPI. Significant speedup is achieved in all cases, but especially on four-processor SMP nodes. Figures 3 and 5 show that execution time if the number of MPI nodes is greater than number of processors, is much better if multithreaded implementation is used.

Figures 7-10 show scalability of the PTMPI on the cluster when running block-based matrix multiplication program. The minimum for the execution time of test program is reached at eight two-processor nodes, since the communication becomes too expensive for larger configuration. Figures 7-8 shows that running more MPI nodes per processor is not expensive in multithreaded implementation, since the communication cost between the MPI nodes on the same SMP machine is not large.

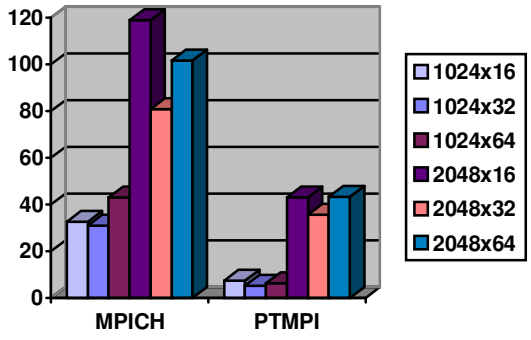


Figure 3: Block-based matrix multiplication execution time in seconds for 16 MPI nodes running on four two-processor SMP nodes.

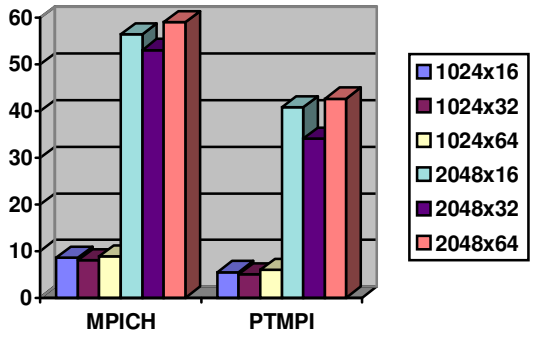


Figure 4: Block-based matrix multiplication execution time in seconds for 8 MPI nodes running on four two-processor SMP nodes.

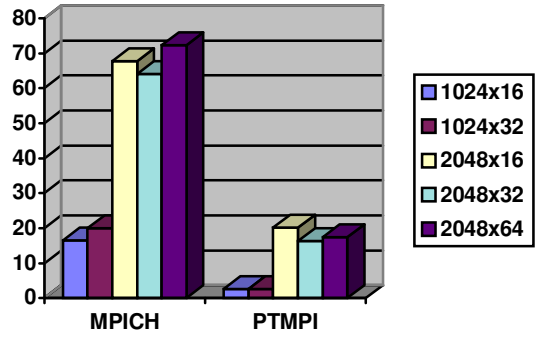


Figure 5: Block-based matrix multiplication execution time in seconds for 32 MPI nodes running on four four-processor SMP nodes.

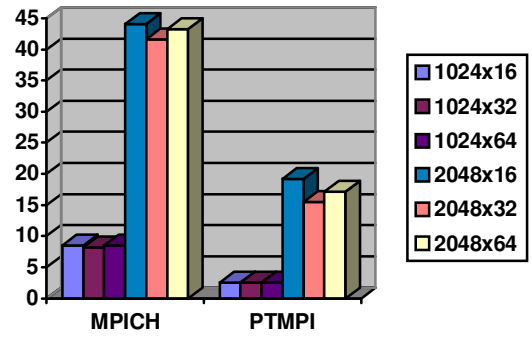


Figure 6: Block-based matrix multiplication execution time in seconds for 16 MPI nodes running on four four-processor SMP nodes.

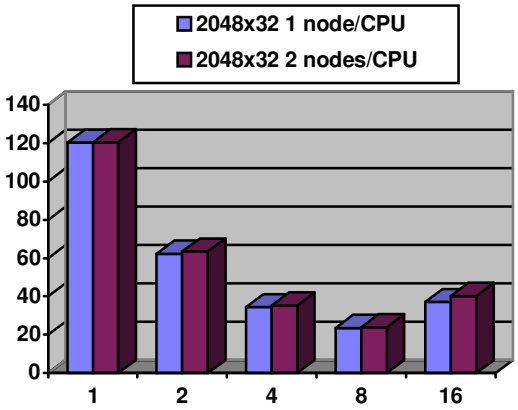


Figure 7: PTMPI block-based matrix multiplication execution time in seconds as function of number of two-processor SMP nodes.

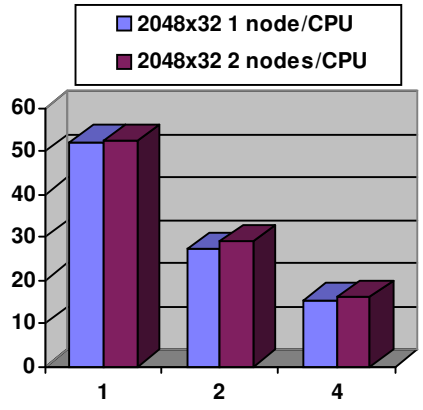


Figure 8: PTMPI block-based matrix multiplication execution time in seconds as function of number of four-processor SMP nodes.

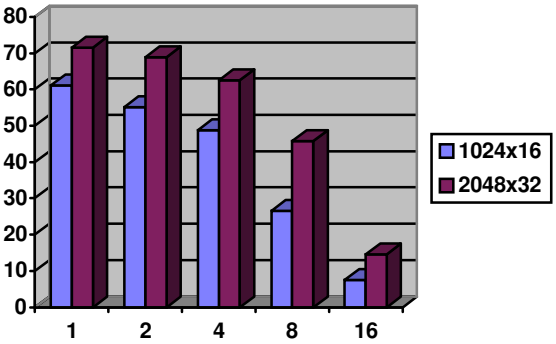


Figure 9: PTMPI block-based matrix multiplication MFLOPS rate as function of number of two-processor SMP nodes (one thread per processor).

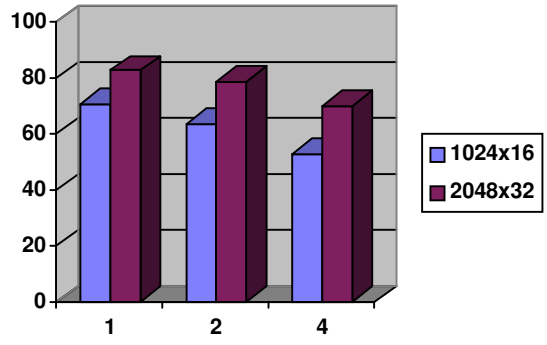


Figure 10: PTMPI block-based matrix multiplication MFLOPS rate per processor as function of number of four-processor SMP nodes (one thread per processor).

5. Future Improvements

The future research will include reevaluation of the performance, and concentrate on the performance benefits of spin waiting and coscheduling between the processes. Cost distribution will be included in this paper.

Communication between the processes needs to be improved to enable waiting for receive request from the MPI node before starting send of large message. Message size for delayed sending, and duration of spin waiting need to be evaluated.

System should be changed to support multiple communicators and dynamic creation of nodes. Dynamic changing of number of MPI nodes per process should be considered.

6. Conclusions

Initial performance analysis shows that multithreaded approach for the implementation of MPI functions on the cluster of SMP machines gives significant speedup.
Performance benefit for multithreaded MPI execution will be greater with increase of the number of processors per cluster SMP node.

7. References

[1] Culler, D., Singh, P. J., Gupta, A., "Parallel Computer Architecture," Morgan Kaufmann Publishers, 1997.

[2] Milutinovic, V., "The Best Method for Presentation of Research Results," *IEEE TCCA Newsletter,* September 1997, pp. 1-6.

[3] Milutinovic, V., "Microprocessor and Multimicroprocessor Systems," Copyright by Wiley, USA, 2000.

[4] Tang, H., Shen, K., Yang, T., "Compile/Run-time Support for Threaded MPI Execution on Multiprogrammed Shared Memory Machines," to appear in *the Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99).*

[5] Pacheco, P. S., "A User Guide to MPI," User Guide, University of San Francisco.

[6] Watson, D., "Gargleblaster Cluster User Guide," User Guide.

[7] K. Shen, H. Tang, T. Yang, "Adaptive Two-level Thread Management for Fast MPI Execution on Shared Memory Machines," to appear in *the Proceedings of ACM/IEEE SC'99.*