UCSB cs240b project Fall 1999

PTMPI

Threaded MPI execution on cluster of SMP machines

Zoran Dimitrijevic
Department of Computer Science
University of California at Santa Barbara

E-mail: zoran@cs.ucsb.edu

Introduction

- Cluster of SMP machines
  - Each cluster node is SMP machine
  - Communication between the nodes is through etherenet TCP/IP
- Current MPI implementation for shared memory machines:
  - TMPI – threaded MPI execution – each MPI node is a thread inside one process
    - Fast
    - Not scalable – regular OS process can be running on just one machine
  - MPICH – each MPI node is a process – communication between nodes involve operating system activity
    - Slow
    - Scalable – each node can be running on different machine

Problem Statement

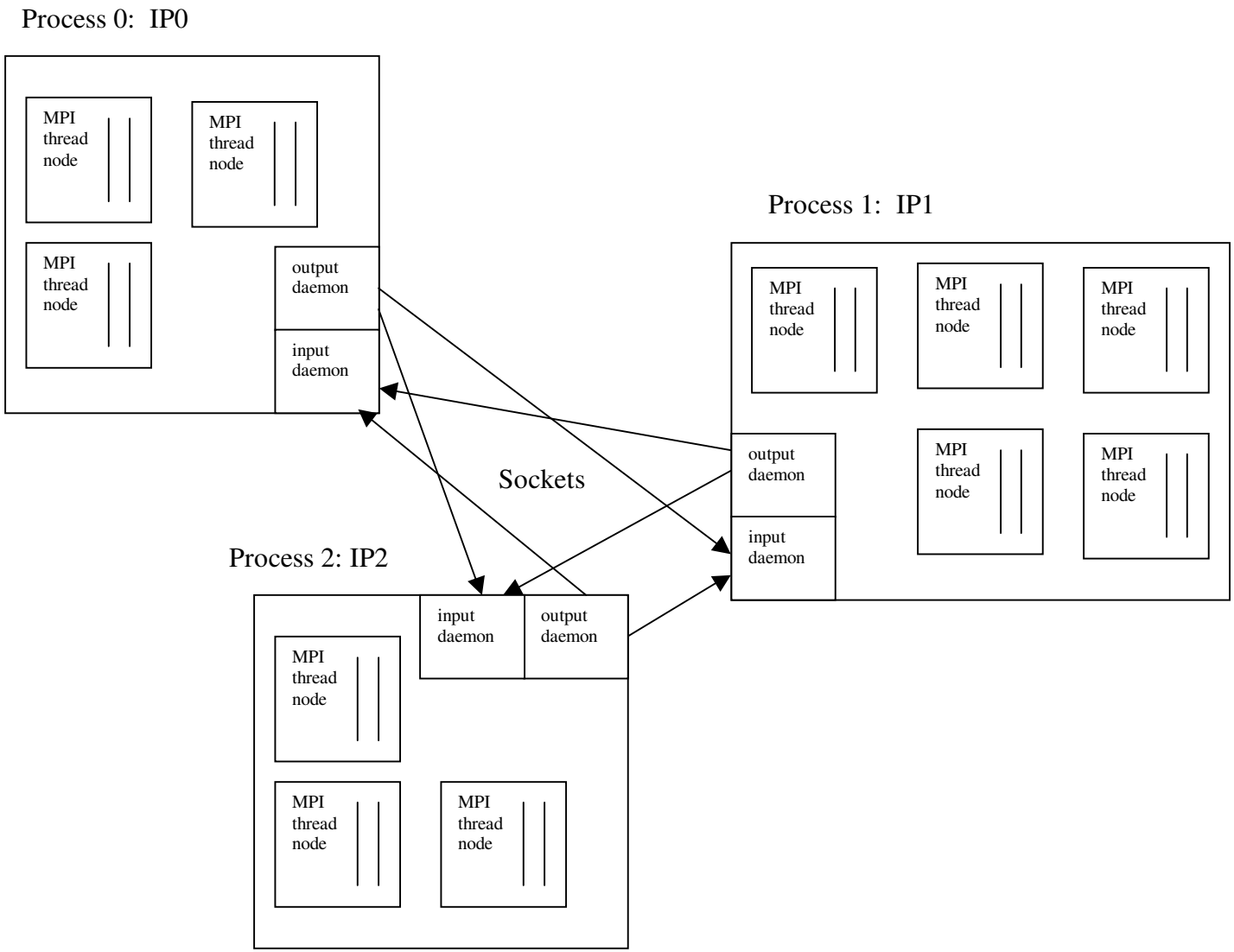
- System consists of several processes
  - Scalability – each process can run on different machine
  - Communication between the processes is through sockets
  - Processes can be running anywhere on the Net
- Each MPI node is a thread inside a process
  - Fast communication between the MPI nodes inside the same process – through shared memory
  - During the startup the nodes are created –
    each process can have different number of MPI nodes running inside it

Proposed Solution

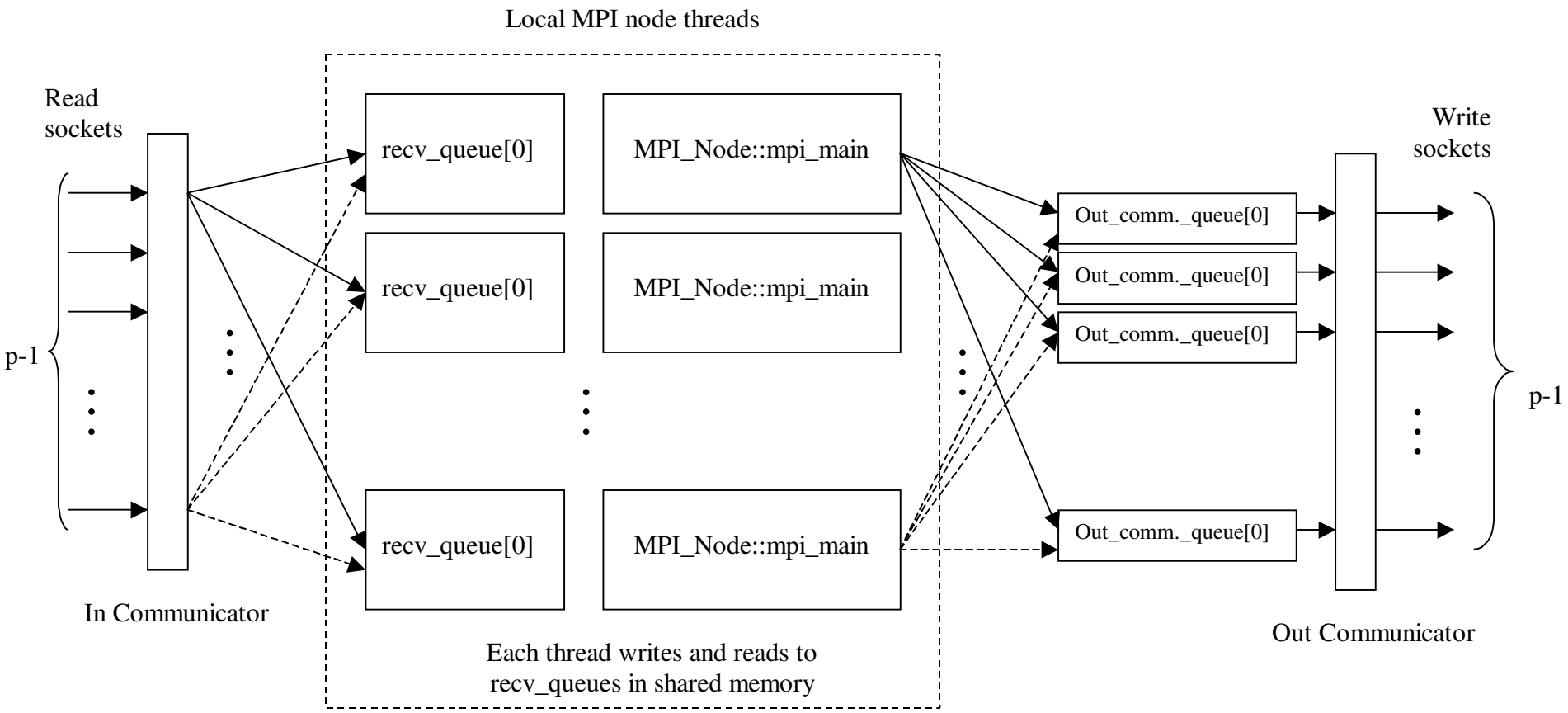
- PTMPI Startup:
  - Configuration is in the resource file
  - Each process is started with single initialization argument – process ID
  - Each process gets its IP and listenning port number
  - There are p processes in the system
  - Complete sockets graph is created – p(p-1)/2 sockets
  - Each process creates local_MPI_count receiver queues
  - Each process creates a thread for each MPI node running on it
  - Each process creates two communication threads:
    - In communicator – read from the sockets and dispatches messages
    - Out communicator – read from its queues (one per each MPI thread) and writes to sockets

- MPI Node Thread Startup:
  - Each MPI node is an instance of class MPI_Node
  - PTMPI main creates thread for each MPI node and passes the local ID to them
  - Each thread creates a new instance of class MPI_Node
  - SPMD in shared memory
    - All global data for MPI program must be copied for each thread
    - This is achieved since all MPI functions are friend function to class MPI_Node or defined in class MPI_Node, and all global MPI data are members of the class MPI_Node
    - All MPI global data can be placed in mpi_global_data.h which is included in MPI_Node class
  - Each thread calls method mpi_main(int argc, char **argv)
    - Arguments are passed from PTMPI main function exept first one (and the name is set to mpi_program)

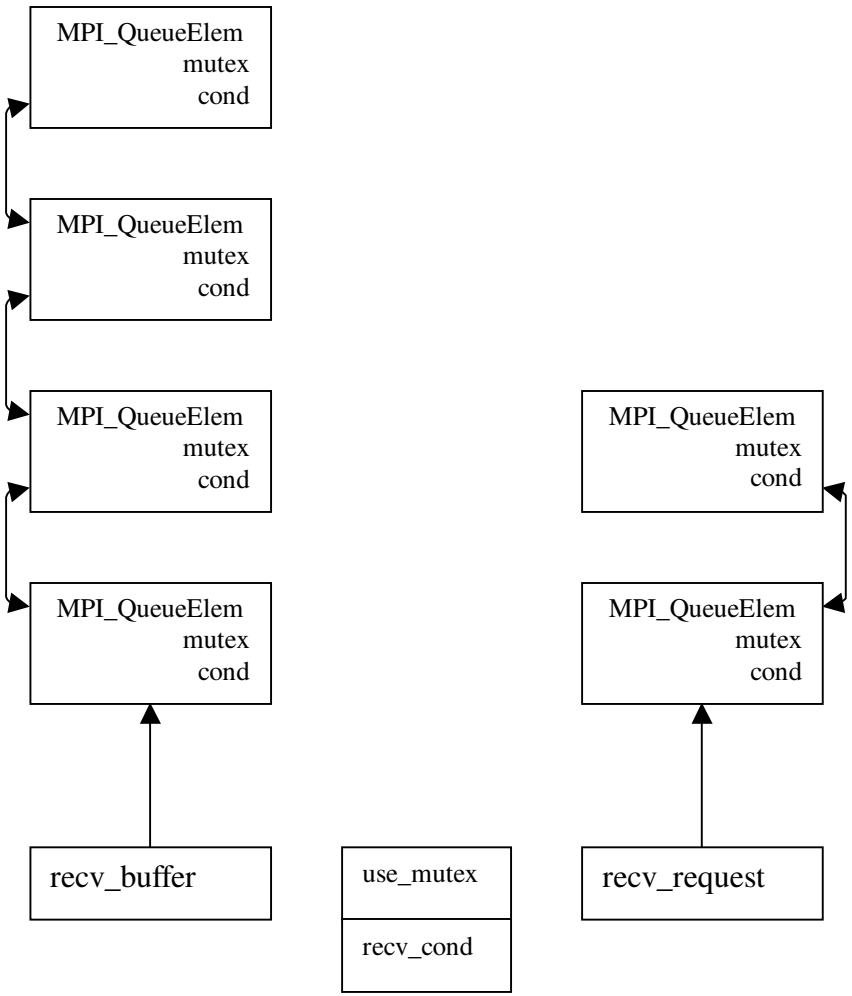
- PTMPI System Layout:



- Process node layout:



- Receiver Queues



- Messages: MPI_QueueElem
  - Goal: minimize the number of memory copy in system
  - All queues in the system are using the same class for elements
  - Broadcast does not copy the message
  - Threads are using mutex and condition members of MPI_QueueElem
  - Last waiter free the message if the message is buffered and deletes the element

- MPI functions implemented:
  - MPI_Init
  - MPI_Comm_rank
  - MPI_Comm_size
  - MPI_Finalize
  - MPI_Send
  - MPI_Isend
  - MPI_Recv
  - MPI_Irecv
  - MPI_Wait
  - MPI_Broadcast

Initial Performance Evaluation

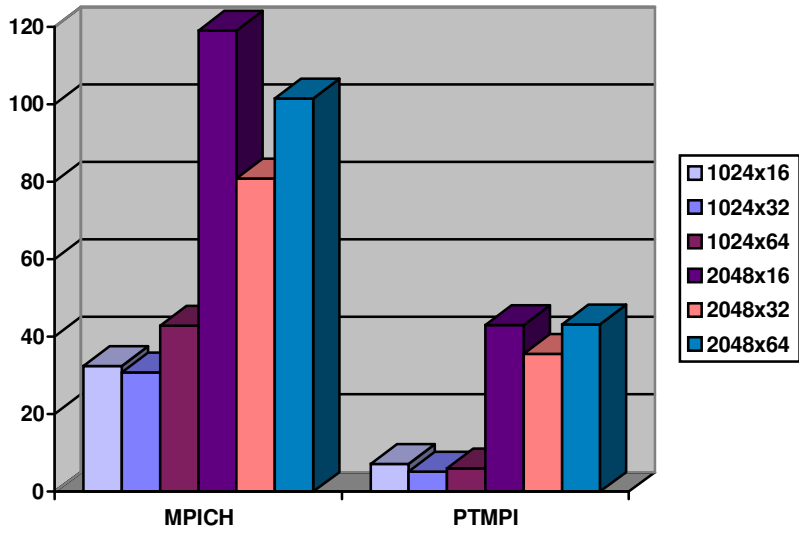


Figure 3: Block-based matrix multiplication execution time in seconds for 16 MPI nodes running on four two-processor SMP nodes.

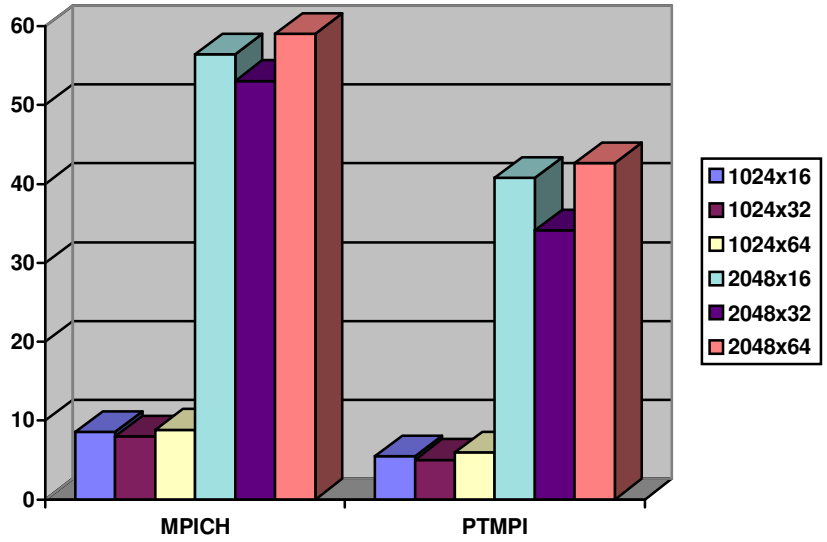


Figure 4: Block-based matrix multiplication execution time in seconds for 8 MPI nodes running on four two-processor SMP nodes.

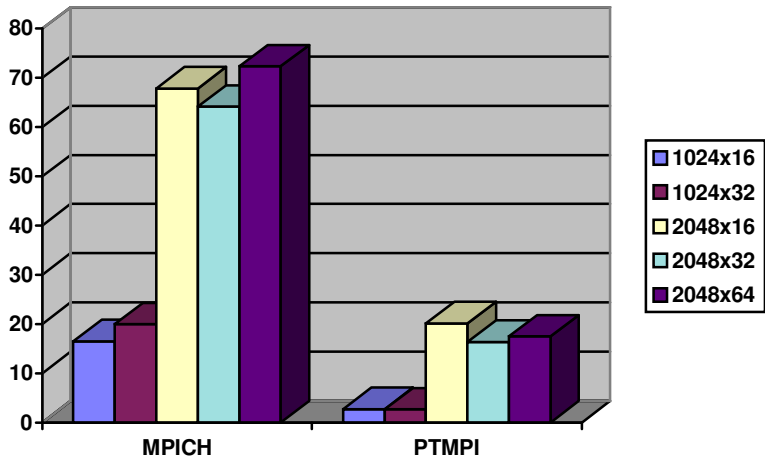


Figure 5: Block-based matrix multiplication execution time in seconds for 32 MPI nodes running on four four-processor SMP nodes.

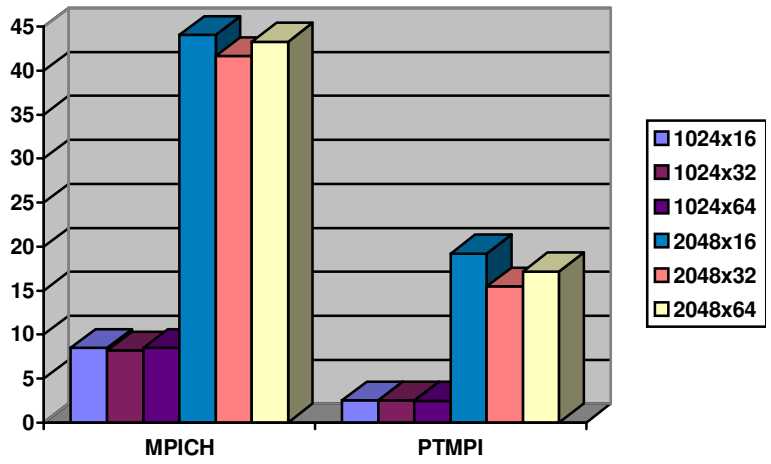


Figure 6: Block-based matrix multiplication execution time in seconds for 16 MPI nodes running on four four-processor SMP nodes.

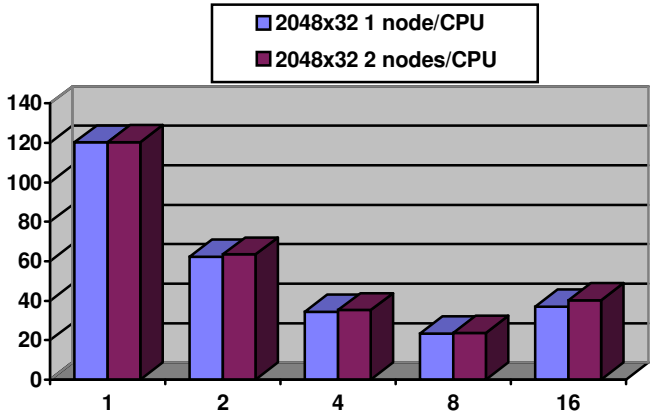


Figure 7: PTMPI block-based matrix multiplication execution time in seconds as function of number of two-processor SMP nodes.

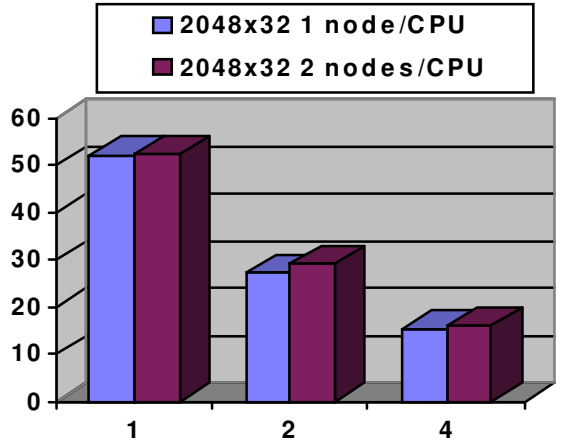


Figure 8: PTMPI block-based matrix multiplication execution time in seconds as function of number of four-processor SMP nodes.

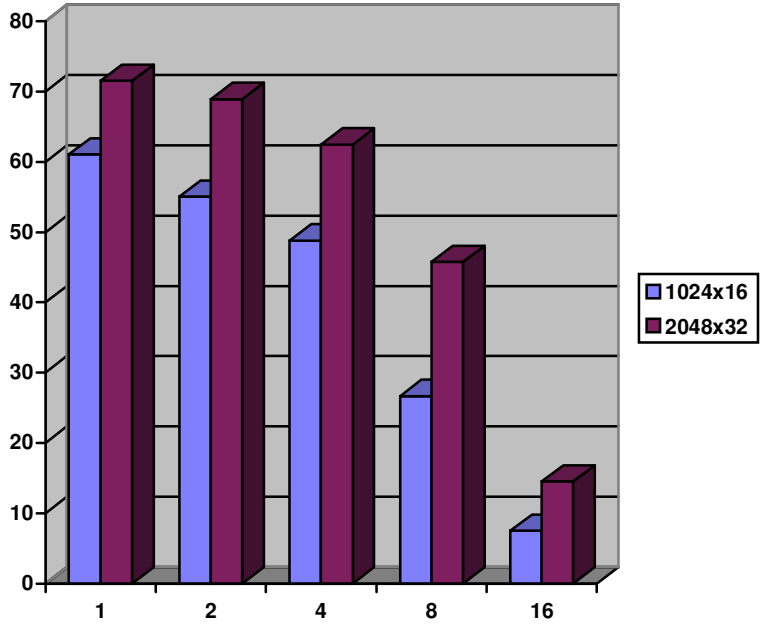


Figure 9: PTMPI block-based matrix multiplication MFLOPS rate as function of number of two-processor SMP nodes (one thread per processor).

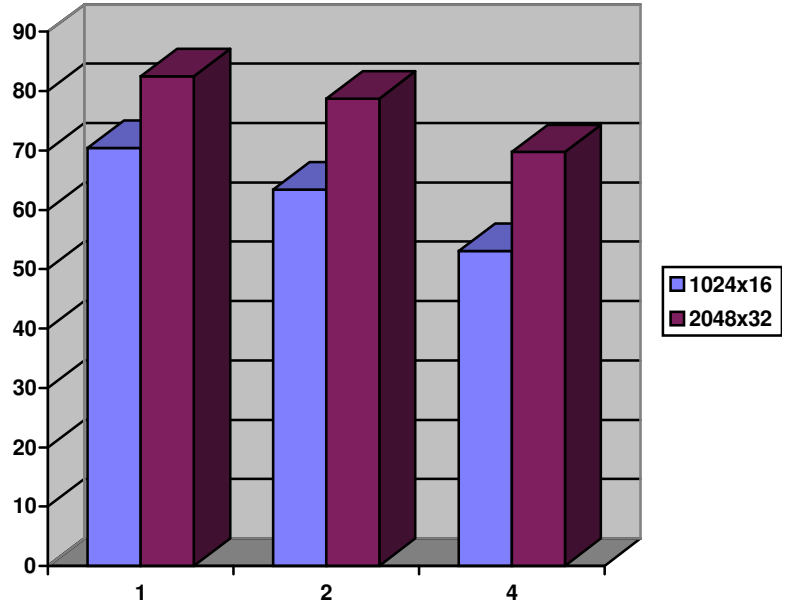


Figure 10: PTMPI block-based matrix multiplication MFLOPS rate per processor as function of number of four-processor SMP nodes (one thread per processor).

Conclusions and Future Improvements

- Basic MPI functions are implemented
- Current MPI_node to process is basic one, it is expected that smart mapping can significantly improve execution speedup for some applications
- Since the communication between the threads is faster than through sockets, MPI gathering function need to be implemented
- Spin waiting for send and receive inside the process if running on real SMP
- Sending only message header through the socket if the message is big, and waiting for message data request when the receiver is ready