A Comparative Study of the MPI Communication Primitives on a Cluster

Arnab Sinha
Computer Sc. & Engg. Dept.
NIT Durgapur, West Bengal
sinharnab@gmail.com

Nabanita Das
ACM Unit
Indian Statistical Institute Kolkata
ndas@isical.ac.in

Abstract: MPI (Message Passing Interface) has become the de facto standard for implementing parallel programs on distributed systems. In MPI, the two basic communication primitives are point-to-point communication and broadcast respectively. In this paper, we evaluate and compare the performance of broadcast with point-to-point communication (both blocking and non-blocking) of the MPI-1 standard library on a cluster computer in communicating the same data block among all processors. The performance in terms of delay is compared by varying the number of processors, and the data block size. The tool *Jumpshot-4* is used for detailed measurement of the performance of MPI communications routines.

Keywords: Parallel Programming, MPI (Message Passing Interface), Speed-up, Communication Primitives, Cluster Computers

1 Introduction

Message passing systems simplify the concurrent software development on parallel computers by separating the hardware architecture from the software configuration of processes. In the last decade, several communication systems for multicomputers have been implemented. Some of them have been developed for a particular architecture whereas others are more general. Examples of these systems are Express, P4, PARMACS and ZipCode. The need of portable communication facilities for a very large set of parallel architectures finally leaded to the definition of the MPI (Message Passing Interface) standard library [1], which embodies the main features of those earlier systems.

Message passing libraries in MPI allow efficient parallel programs to be written for distributed systems. These libraries provide routines to initiate and configure the messaging environment as well as sending and receiving packets of data. Since the communication delay plays an important role to determine the completion time of a program, it is very important to investigate the delay associated with the different communication primitives of MPI, in details, for using them efficiently and appropriately within the programs.

The two popular communication methods of MPI are point-to-point (SEND/RECV which is blocking type and ISEND/RECV which is non-blocking type) and MPI BCAST (for broadcasting). Now it is interesting to study that in transferring the same data block to a number of processes, keeping all other conditions same, which of the above communication methods will perform better in terms of delay. Moreover, how this difference in delay varies with the number of processes and the size of data blocks.

This paper presents, evaluates and compares the performance of the basic point-to-point communication (blocking and non-blocking) and broadcast communication primitives of the MPI-1 standard library on a HCL Cluster computer using the tool *Jumpshot-4* [2]. The comparison is done on the basis of size of data block exchanged and the number of processors involved. The study reveals the interesting fact that for small data size (80 Kbytes), with $1 \leq P \leq 6$, the communication delay for BCAST is more than it is for the non-blocking point-to-point primitive ISEND, where P is the number of processors involved. The study on its application to a standard matrix multiplication program also supports this finding. It also reveals the fact that with small number of processors $P \leq 6$, ISEND is always marginally better than BCAST, and the improvement in

Arnab Sinha is a student of NIT, Durgapur and is presently working for M.Tech dissertation work at ISI, Kolkata under the supervision of Prof. Nabanita Das.

completion time increases with the increase in the data block size. However, with P > 6, BCAST outperforms ISEND in respect of completion time. Therefore, it is suggested that for number of processors $P \leq 6$, broadcast communication should be implemented with ISEND using linearly.

The rest of the paper is organized in the following way. Section 2 describes the test platform and tools used. Section 3 presents the experimental results on communication primitives alone. Section 4 shows the results on matrix multiplication algorithm. Section 5 concludes the paper.

2 Test Platform & Tools Used

The cluster computer used for this study consists of 18 nodes having Intel Pentium IV processors. The master node configuration is Intel motherboard with Intel E7210 chipset with single CPU of 3.0 GHz with 1MB L2 cache. It has 1 GB of ECC DDR RAM. The operating system is Red Hat Enterprise Linux ES3.0 Standard.

The slave nodes have Intel motherboard with Intel 865G chipset. Their processors operate at clock speed of 3.0 GHz with hyper threading support and 1MB L2 cache. The memory is 512 MB DDR RAM and the operating system is Red Hat Enterprise Linux 3.0. All the nodes form a star connected topology using HCL Gigabit switches as shown in Fig.1.

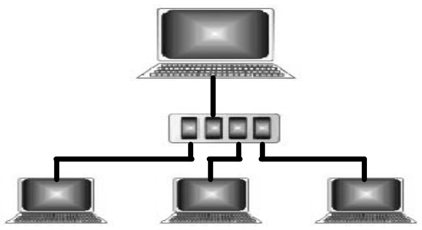

Figure 1: Star Network configuration

The MPICH implementation is from ANL mvapich-1.2.6 compiled with gcc-3.2.3-3.x86_64. Log files in .clog format are generated using MPE and viewed using *Jumpshot-4* [2] for doing postmortem performance analysis, especially the time delay, of the executed parallel programs.

3 Experiments and Results

Knowing the communication overhead of the MPI communication primitives before hand, has dual benefits. It enables the programmers to write efficient parallel software and secondly, to obtain a model to assess the overhead introduced from those communication types for different message sizes and processor numbers. The measurements obtained from each experiment concern the minimum, maximum and average values of communication latency.

We are interested to compare the basic communication operations of MPI BCAST, SEND and ISEND respectively. The objective is, when in a program the master needs to transfer the same data block to all slave nodes, that is often required during initialization, which communication primitive will be efficient in terms of latency, and how the difference in latency depends on the data block size and the number of processors.

Two simple MPI programs are executed where blocks of data are communicated using the MPI primitives namely BCAST and SEND/ISEND in a simple linear loop. While executing these codes for different sizes of data blocks and number of processors, we generate the logfiles. Each logfile is then viewed using *Jumpshot-4*, analyzed and the time elapsed for communicating the data block from master to slave processors is measured.

Figures 2 and 3 show the snapshots of the logfiles generated using BCAST and SEND respectively for transferring 80 Kbytes of data to 12 processors. Figures 4 & 5 show the variation of communication time with number of processors for transferring data blocks of 80 Kbytes and 180 Kbytes respectively for BCAST, SEND and ISEND operations. Each point on the graph represents the average time for 10 executions.

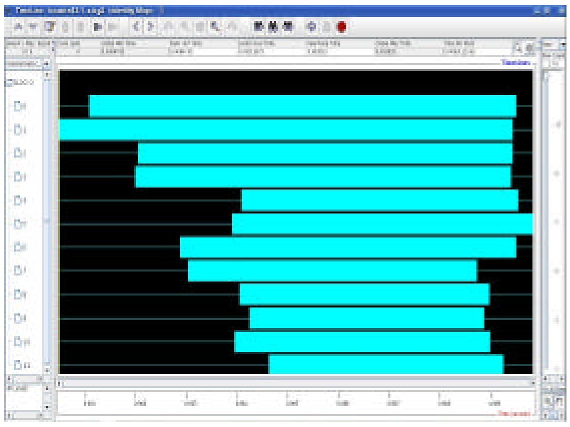

Figure 2: Logfile viewed with *jumpshot* tool using BCAST

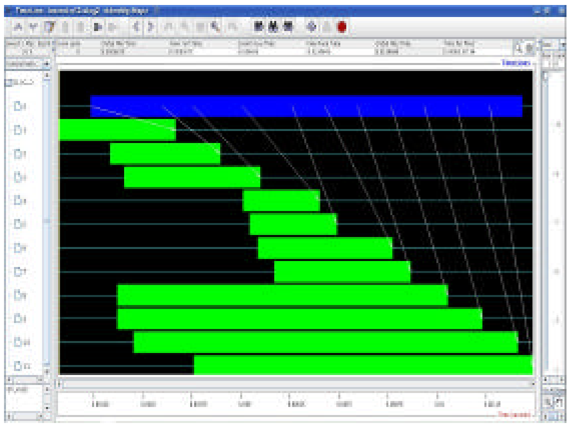

Figure 3: Logfile viewed with *jumpshot* tool using SEND

From the graphs it is evident that communication times required by SEND and ISEND are comparable. It is expected since the processors have no computation load, and has to wait equally for the completion of communication in both cases. It is also clear that with number of slave processors P $\geq$ 8, the communication time required for same data block size is less for broadcasting than for SEND/ ISEND. The difference is more with larger data block and with more number of processors. More interestingly, for data size of 80 Kbytes, the non-blocking point-to-point communication has latency even lesser than BCAST with 4 $\geq$ P $\geq$ 8. The study reveals the fact that BCAST saves communication time significantly compared to SEND/ISEND for P $\geq$ 8. However for less number of processors, and less data block size, it is evident from the results that ISEND (non-blocking point-to-point communication) performs better.

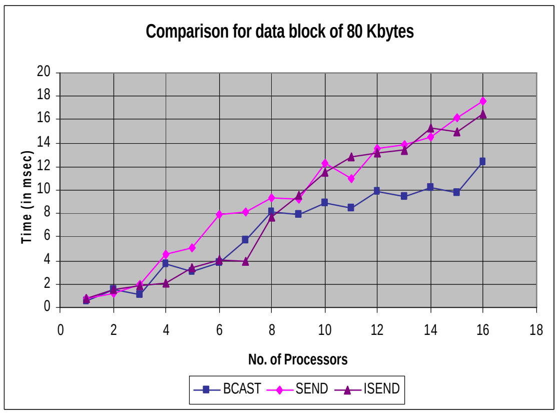

Figure 4: Communication latency vs number of processors for 80 Kbytes data

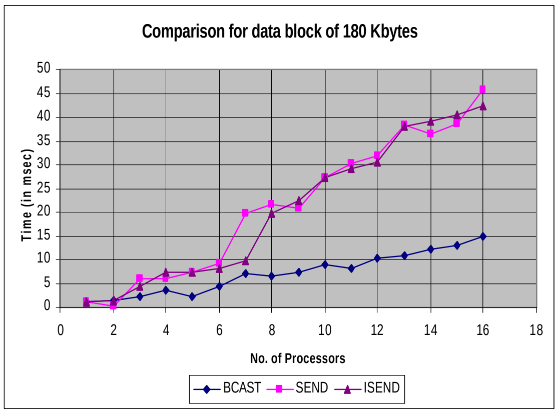

Figure 5: Communication latency vs number of processors for 180 Kbytes data

4 Study on Matrix Multiplication Algorithm

Next, to verify the above observation, experiment is done on a simple parallel algorithm of matrix multiplication where to multiply two N × N matrices A and B, A is distributed row-wise over the processors uniformly i.e. at most ⌈N/P⌉ consecutive rows to each processor (*alocal*), and the B matrix is broadcast to all processors [3]. Each processor then computes definite rows of the product matrix C, (*clocal*) as shown in Figure 6. Broadcasting of B matrix is implemented by point-to-point SEND / ISEND in linear loop or BCAST respectively for comparing the time of completion in the master.

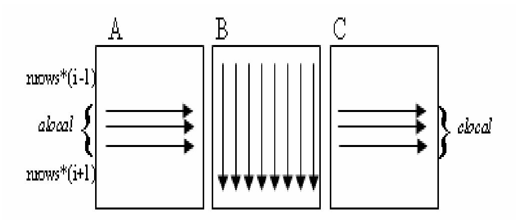

Figure 6: A slave processor –i with submatrix *alocal* of A, and matrix B, calculates submatrix *clocal* of C

The outline of the algorithm is given below:

```
float a(dim,dim), b(dim,dim), c(dim,dim)
nrows = dim/(numprocs-1)
if( myid .eq. master ) then
! Intialize matrices A & B
call Broadcast/Send(B to all) … (I)
do i=1,numprocs-1
call Send(nrows rows of A to i)
end do
do i=1,numprocs-1
call Receive(nrows rows of c from i) end do
else ! Processors other than master
allocate ( alocal(nrows,dim),
clocal(nrows,dim) )
call Broadcast/Recv(B to all) … (II)
call Receive(alocal from master)
call jikloop ! clocal=alocal*B
call Send(clocal to master)
endif
```

This program is executed varying the dimension of the matrices, the number of processors used and using point-to-point SEND / ISEND in loop and BCAST respectively for the distribution of matrix B (in the statements marked as I and II in algorithm).

Again repeating the experiment the communication time and the total completion time in the master are measured and the averages for 10 executions are plotted as shown in Figures 7-10 respectively.

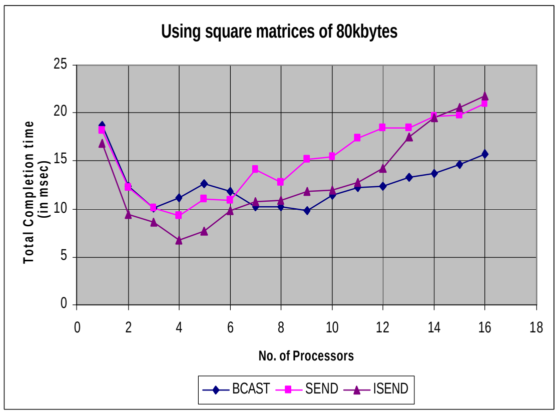

Figure 7: Completion time vs number of processors for matrix size of 80 Kbytes

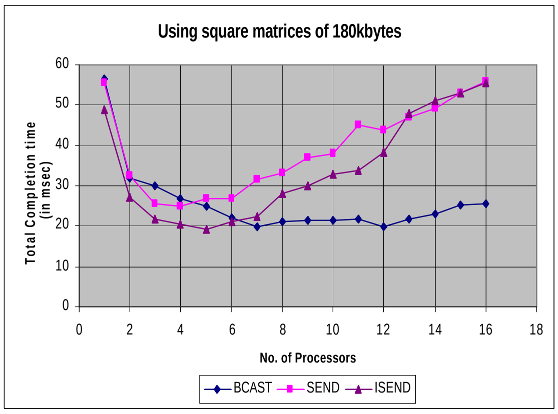

Figure 8: Completion time vs number of processors for matrix size of 180 Kbytes

From Figures 7 and 8, it is evident that with P $\geq$ 8, broadcast results the least time of completion for both matrix sizes, though the time deteriorates for P $\geq$ 9, as then the communication overhead overrules the benefit of parallel processing. It is also clear that for P $\leq$ 6, ISEND results the least completion time, revealing the fact that for less number of processors it is advantageous to use non blocking point-to-point communication for broadcasting same data to all.

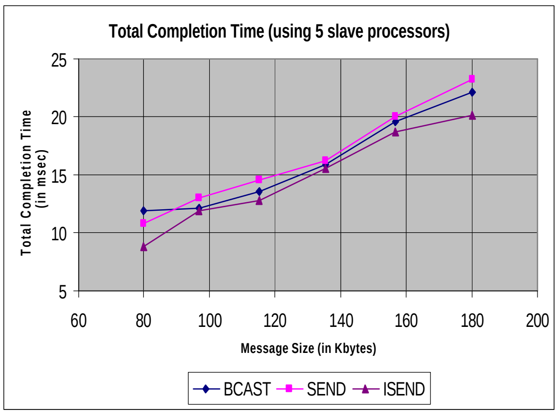

Figure 9: Completion time vs data block size with P = 5

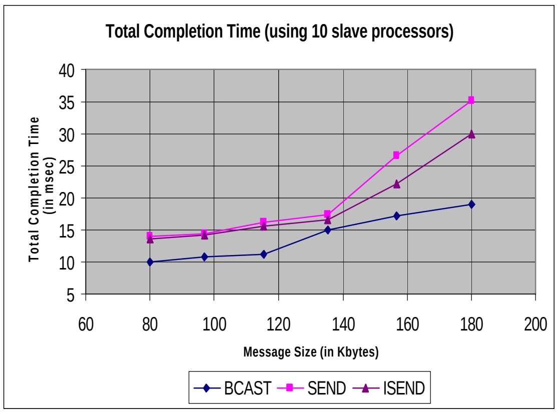

Figure 10: Completion time vs data block size with P = 10

Table 1: Comparison between ISEND & BCAST Best primitive with % improvement in time

		Matrix Size (in Kbytes)			
		80	**115.2**	**156.8**	**180**
No of Processors	**3**	ISEND 16.49%	ISEND 9.30%	ISEND 17.49%	ISEND 27.22%
	5	ISEND 26.01%	ISEND 5.90%	ISEND 4.86%	ISEND 9.01%
	7	BCAST 5.26%	BCAST 2.41%	BCAST 12.50%	BCAST 10.26%
	10	BCAST 25.82%	BCAST 27.40%	BCAST 22.41%	BCAST 36.46%
	16	BCAST 28.08%	BCAST 31.71%	BCAST 39.16%	BCAST 48.21%

It is to be noticed that in both cases when compared with Figures 4 and 5 respectively, it is observed that the difference between ISEND and SEND are prominent in the later cases. It is obvious, since for matrix multiplication algorithms, processors can better utilize the waiting time in sending data by using non-blocking send, showing better performance compared to the blocking send. Figures 9 and 10 show the variation of completion time with message size using P as a parameter. It clearly shows that ISEND marginally outperforms BCAST with P=5, whereas with P=10, BCAST results the least time of computation. Moreover, the savings in completion time increases with the increase in data block size. Finally, Table 1 summarizes the percentage improvement in completion time for matrix multiplication using ISEND and BCAST respectively.

5 Conclusion

From the above experimental study on the given cluster configuration described in section 2, it is interesting to note that with number of processors P $\leq$ 6, broadcasting can be implemented using point-to-point ISEND linearly to save time appreciably. Table 1 shows that for matrices of size 80 Kbytes, ISEND results a savings of 26% over BCAST with P = 5, and 16% with P = 3. So, the experimental results reveal the fact that for small volume of data and for less number of processors, it is better to implement broadcasting in terms of non-blocking point-to-point communication in parallel programs using MPI whereas for other cases broadcasting saves time significantly.

REFERENCES

1. Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., Dongarra, J.J., MPI: The Complete Reference, The MIT Press, 1996.
2.http://www.mcs.anl.gov/perfvis/software/viewers/index.htm#Jumpshot-4
3. Michael J. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2003.
4. http://en.wikipedia.org