Performance Analysis of Collective Communication Routines for Infiniband Network Based Supercomputer: Shadow at Mississippi State University

Md Masrul Huda[1, 2], Edward A. Luke[2,3,*]

[1]Dave C. Swalm School of Chemical Engineering, Mississippi State University, MS State, MS

[2]Center for Advanced Vehicular Systems, Mississippi State University, MS State, MS

[3]Computer Science and Engineering, Mississippi State University, MS State, MS
[*] Corresponding author: *luke@cse.msstate.edu*

Collective communication is a high-level abstraction of point-to-point communication where generally more than two processes participate. Performance of parallel applications often depends on efficient implementation of these routines. Here, we present a comparative study of performance analysis of All-to-All and All-to-All personalized collective communication for different algorithms. We also present the comparison of implemented algorithms with vendor implementation (Intel-MPI) and available popular Open source implementation of MPI standard: MPICH and Open-MPI.

1. Introduction

Shadow is Cray super computer at Mississippi State University. Which has 4800 Intel Ivy Bridge processor cores and 28,800 Intel Xeon Phi cores[1]. It has a peak performance of 593 teraFLOPS (trillion calculations per second). A wide array range of people uses this super computer for their scientific research. It is very important to establish a benchmark for highly demanding collective communication routines for it's user, so that they can optimized their computational kernels to achieve maximum performance. A myriad number of algorithms[2-4]. All of these algorithms have their merits and demerits, depending upon underlying architecture, parallel application, message size and scalability of program etc.

Here, We present three algorithms for All-to-all communication: 1) Naïve-Non blocking 2) Recursive Doubling in Hypercube network for both power of two and non-power of two and 3) Ring algorithms. For All-to-all personalized communication we implemented four algorithms: 1) Naïve-Nonblocking [2], 2) Wraparound[2] 3) E-Cube routing and 4) Hypercube Implementation based on mesh algorithms.

2. All-to-All communication

In case of All-to-All to broadcast each processor has same message to distribute for all processors. A typical representation is shown on fig:-1(Collected from Vipin *et al*). It is frequently used in Matrix multiplication, matrix-vector multiplications.

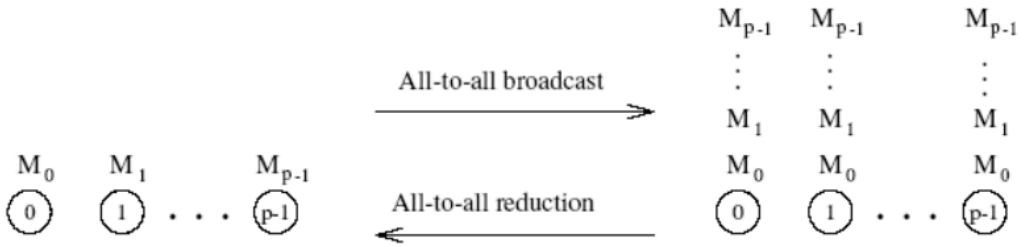


Figure-1: All-to-All communication

2.1. Naïve NonBlocking algorithm:

It is Naïve implementation of All-to-All communication, where each processors posts P-1 non-blocking communication request, assuming underlying MPI implementation utilize intelligent point-to-point communication so that, it gain performance. This algorithms is motivated from Thakur *et al.*'s implementation of Nonblocking all-to-all personalized communication. It would be difficult to model cost model for that algorithm as we do not know how will communication be completed. Pseudo-code is as follow:

```
for (i=0;i<p;++i){
    int partner=i;
    MPI_Irecv();
    MPI_Isend();
}
MPI_Waitall();
```

2.2 Recursive Doubling in Hypercube network

At every step every processor exchanges data with it's neighbor , and message keep doubling at each step. It takes overall logP steps. Total cost for that communication is $t_s \log P + t_w m(p-1)$.

```
Result=my_msg;
For(i=0;i<d){
    Partner=my_id XOR 2^i ;
    Exchange msg with partner;
    Concate  msg with result;
}
```

2.3 **Ring algorithms**

Message is circulate to adjacent neighbor and it takes (p-1) steps. Message size remain constant throughout the communication. Total cost for that communication is T=(ts+tw)(p-1). Each processor receive message from left and send message to right neighbor.

```
For (i=1;i<p;++i){
     Send to right;
     Recv from left;
}
```

2.4 Result:

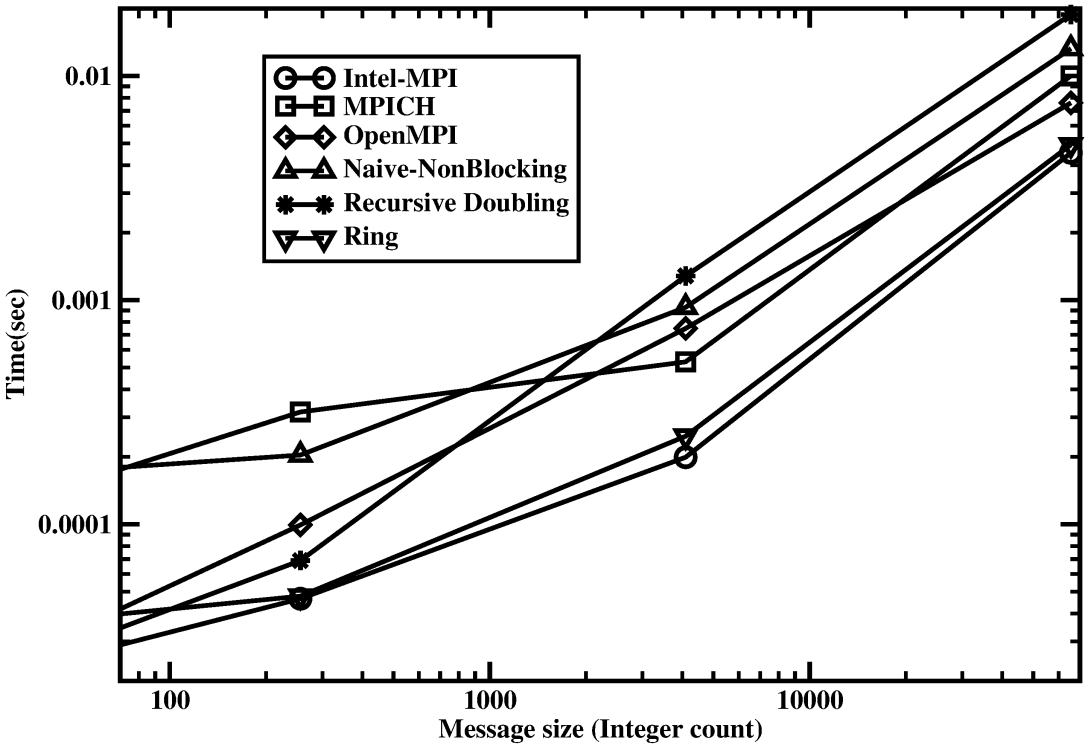


Figure-2: time vs message size for 32 processors.

In the figure we present the effect of message size on 32 processors. Clearly Vendor implantation (Intel-MPI) is the best one for all message size, as it knows the underling architecture better and it implemented optimized algorithms for their known architecture. The best implantation algorithms of this report is ring algorithms, recursive doubling shown poor performance for large message, which little surprising. Interestingly, Naïve algorithms didn't that bad, that general

audience would expect. For short message size MPICH implementation has shown poor performance.

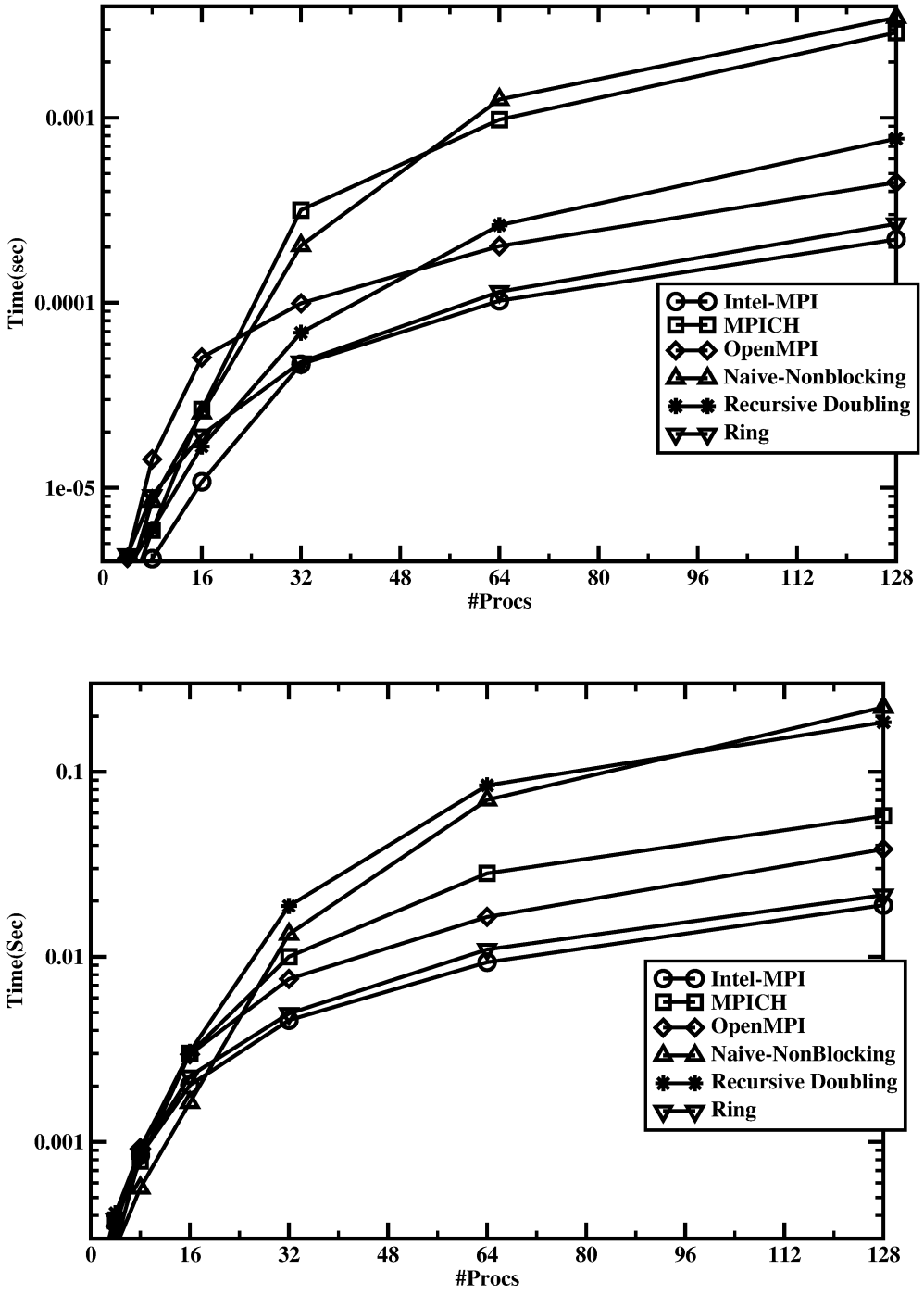


Figure-3: a) time vs. #processor(256 integer message) b)) time vs. #processor(65536 integer message)

At figure-3, we have shown performance for variable processors at fixed message size (e.g: small and large message) size. As expected Collective communication take

longer time for large processors count. For all cases , Intel-MPI has shown better performance. Ring algorithms performance is also promising for both case. MPICH is poor among the available implementations.

3 All-to-all personalized.

3.1 Naïve NonBlocking algorithm:

MPICH has adopted that algorithms for their early implementations.

Pseudo-code is as follow:

```
for (i=0;i<p;++i){
    int partner=i;
    MPI_Irecv();
    MPI_Isend();
}
MPI_Waitall();
```

3.2 Wrap around algorithm:

Each processor receives from i-1 and sends to i+1, where i=1,p-1 steps. This algorithm is good for long message.

3.3 E-cube routing:

It is based on store and forwarding communication, where congestion is assumed. Total cost is, (ts+twm)(p-1). It take p-1 steps, at each step node communicate with id XOR step number.

3.4 Hypercube based algorithm extended from mesh:

It takes logp steps, at every step each processor keep busy itself. In figure-4, flow chart is shown, which exactly identical to Matrix transpose after the all communications. This implementation in this report has some bugs,

M / P	0	1	2	3	4	5	6	7
0	0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7
1	1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7
2	2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7
3	3,0	3,1	3,2	3, 3	3,4	3,5	3,6	3,7
4	4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7
5	5,0	5,1	5,2	5,3	5,4	5, 5	5,6	5,7
6	6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7
7	7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7

M / P	0	1	2	3	4	5	6	7
0	0,0	1,0	0,2	1,2	0,4	1,4	0,6	1,6
1	0,1	1,1	0,3	1,3	0,5	1,5	0,7	1,7
2	2,0	3,0	2,2	3,2	2,4	3,4	2,6	3,6
3	2,1	3,1	2,3	3, 3	2,5	3,5	2,7	3,7
4	4,0	5,0	4,2	5,2	4,4	5,4	4,6	5,6
5	4,1	5,1	4,3	5,3	4,5	5, 5	4,7	5,7
6	6,0	7,0	6,2	7,2	6,4	7,4	6,6	7,6
7	6,1	7,1	6,3	7,3	6,5	7,5	6,7	7,7

M / P	0	1	2	3	4	5	6	7
0	0,0	1,0	2,0	3,0	0,4	1,4	2,4	3,4
1	0,1	1,1	2,1	3,1	0,5	1,5	2,5	3,5
2	0,2	1,2	2,2	3,2	0,6	1,6	2,6	3,6
3	0,3	1,3	2,3	3,3	0,7	1,7	2,7	3,7
4	4,0	5,0	6,0	7,0	4,4	5,4	6,4	7,4
5	4,1	5,1	6,1	7,1	4,5	5, 5	4,7	7,5
6	4,2	5,2	6,2	7,2	4,6	5,6	6,6	7,6
7	4,3	5,3	6,3	7,3	4,7	5,7	6,7	7,7

M / P	0	1	2	3	4	5	6	7
0	0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
1	0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
2	0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
3	0,3	1,3	2,3	3, 3	4,3	5,3	6,3	7,3
4	0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
5	0,5	1,5	2,5	3,5	4,5	5, 5	6,5	7,5
6	0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
7	0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figure-4: Hyper cube all to all personalized communications.

3.5: Results

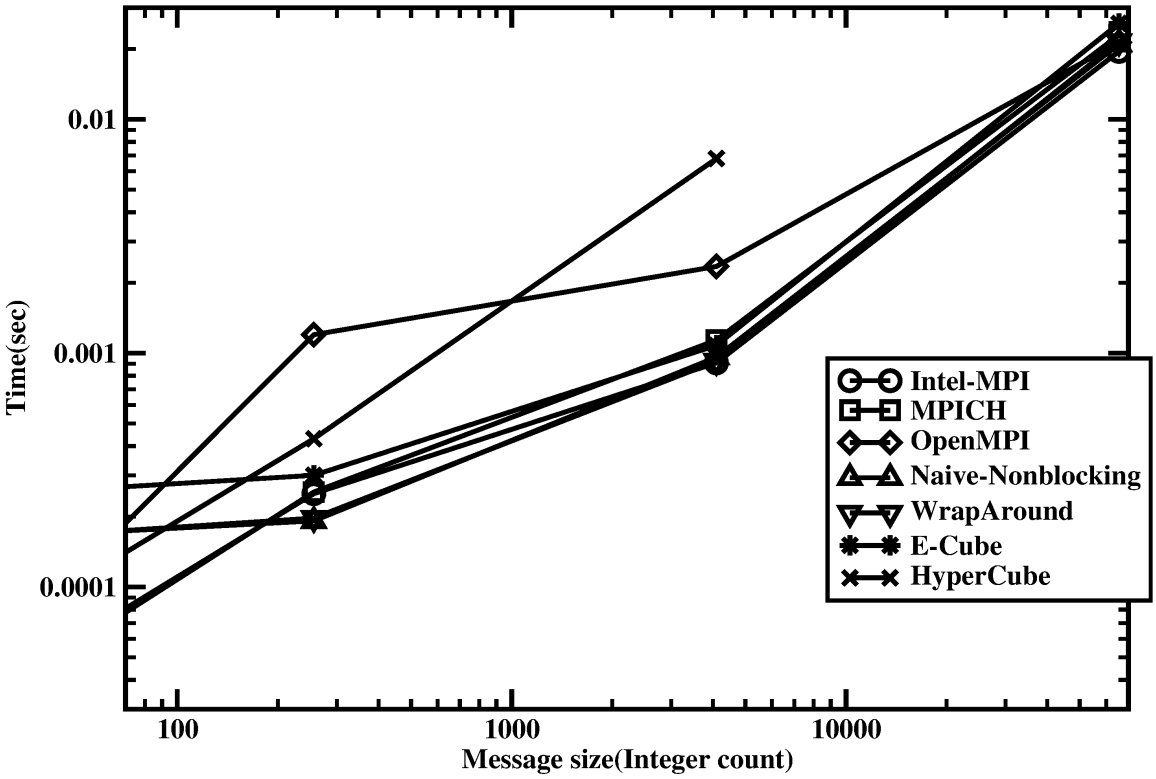
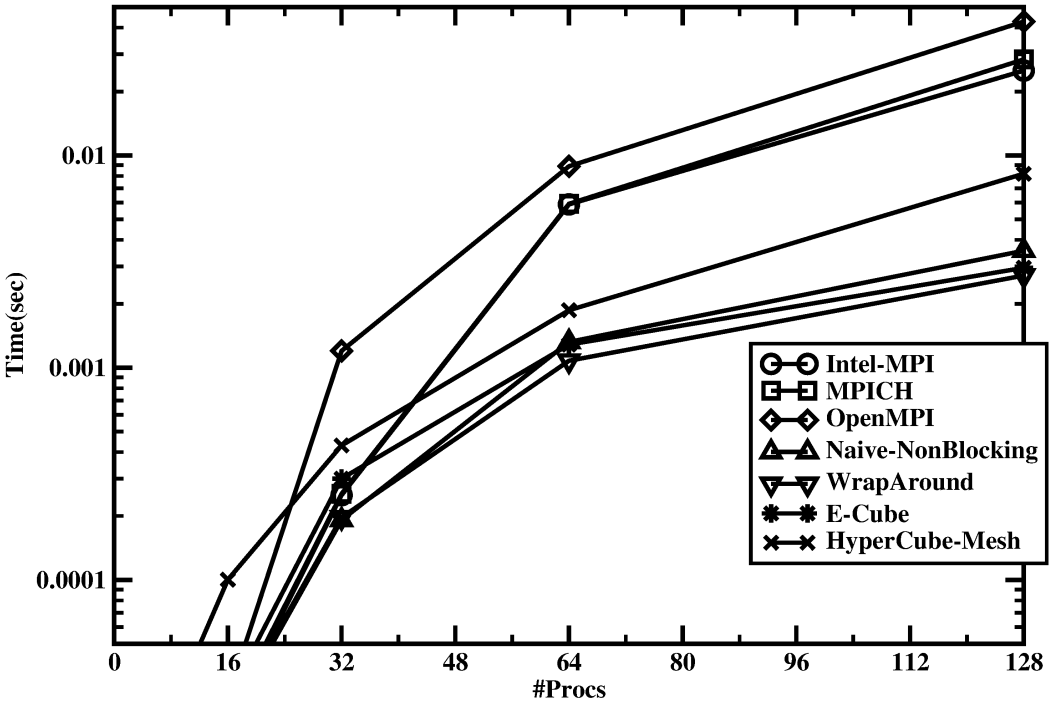


Figure-5: time vs message size for 32 processors.



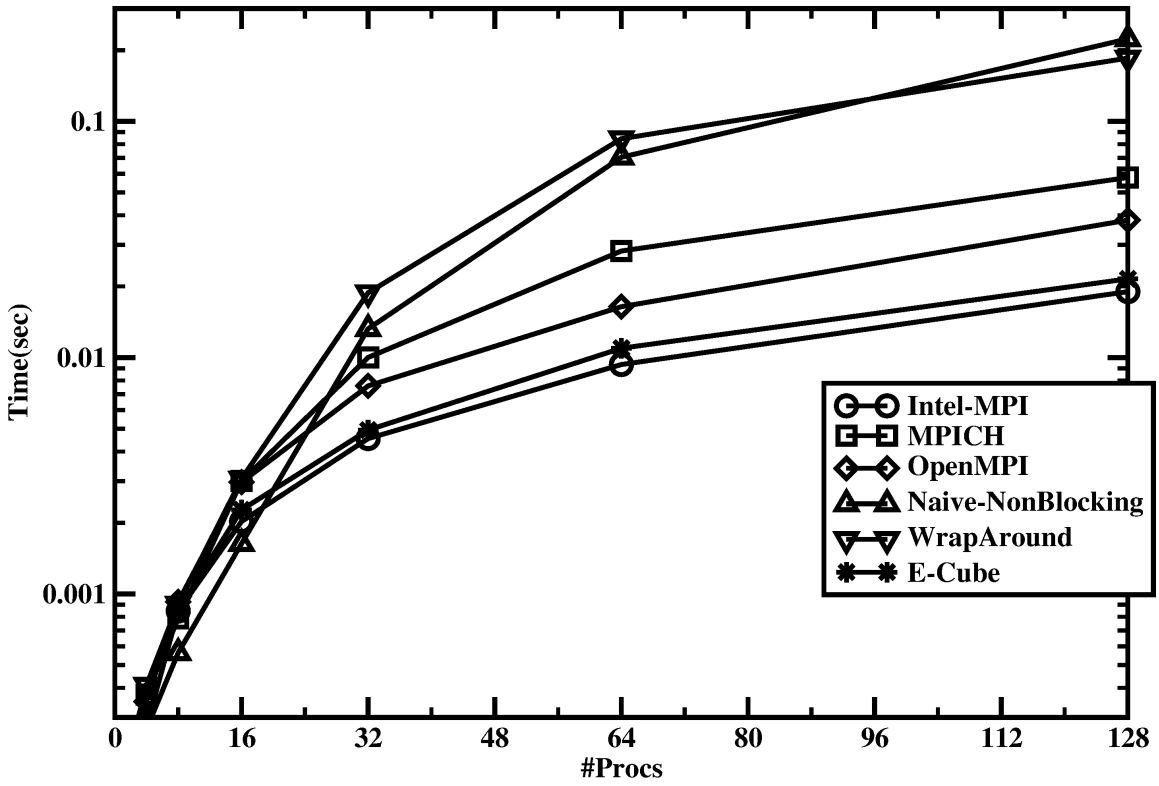


Figure-6: a) time vs. #processor(256 integer message) b) time vs. #processor(65536 integer message)

Conclusion:

In conclusions vendor implementation is always good. Author of this report is also interested to implement other available popular algorithms, so that future user of Shadow super computer can get a detailed benchmark study.

References:

[1] https://www.hpc.msstate.edu/computing/hpc.php

[2] R. Thakur, *Int. J. High Perform. Comput. Appl.*, 2005, **19**, 49–66.

[3] J. Bruck, C. T. Ho, S. Kipnis, E. Upfal and D. Weathersby, *IEEE Trans. Parallel Distrib. Syst.*, 1997, **8**, 1143–1156.

[4] S. Bokhari, 1991.,NASA Contractor report