An Empirical Study of MPI over PC Clusters

Fazal Noor, Majed Alhaisoni*, Antonio Liotta+*
*Computer Science and Software Engineering Department
University of Hail, Saudi Arabia
+Department of Electrical Engineering and
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
The Netherlands
f.noor@uoh.edu.sa, majed.alhaisoni@gmail.com, a.liotta@tue.nl

***Abstract* — Message Passing Interface (MPI) is an important mechanism in P2P. Herein we assess how different types of MPI collective communication functions perform on a Gigabit Ethernet Homogeneous Beowulf PC cluster. In this way we provide an insight on the factors that affect P2P applications over an enterprise context such as the emerging Cloud-based services. By contrast to the literature, which includes mostly theoretical studies, we carry out an empirical study. We show that the bahaviour of scatter and gather are most unpredictable in comparison with other collective functions.**

Keywords - MPI benchmark, Homogeneous PC Cluster, Ethernet, Collective Communications, Latency, Bandwidth, All-to-all, Gather, Scatter, Broadcast.

I. INTRODUCTION

In recent years Peer-to-Peer (P2P) networks have become an active area of research [1]-[4]. Traditional networks use the client/server paradigm where dedicated servers offer clients services. P2P networks are characterized by all peers having the capability of both being a client and a server. P2P networks can support many applications such as sharing of resources, e.g. communication services, file sharing, query search, distributed computing, etc.

In P2P topology, MPI (Message Passing Interface) is considered as a common communication protocol for various P2P systems. Therefore, it is considered as a good mechanism with its goals are to have high performance, scalability, and portability. Having low delay with reasonable throughput is important for computing clusters, due to a lack of shared memory implies large amounts of network data transfer. However, portablility is very important for MPI. The scalability of MPI is mainly due to MPI being the real standard in distributed computing.

In this paper, a simulation of P2P is done using MPI collective communications routines on a PC cluster to measure and evaluate the performance of P2P system. A homogeneous PC cluster is defined as one having identical hardware (including network hardware such as switches) and operating system on all the machines in the network. It is considered heterogeneous if PC hardware and/or software is different from each other in a cluster. One of the reasons to study homogeneous PC cluster is to gain insight into the behaviour of collective communication models used as models usually are made under the assumption of homogeneity. The objective of this paper is twofold, first to study how a variety of MPI communication models perform over PC clusters. Second, we define and measure in practical settings the execution time of Ethernet networks. We pinpoint the overheads and how these affect link efficiency.

Among all technologies, including Infiniband [5], Quadrics [6], and Myrinet [7], we have decided to focus on Ethernet which is readily available for experimentation.

The paper is organized as follows, in Section II some related work is presented. In Section III experimental methods are presented and in Section IV the results are presented. Section V contains discussion of the results presented. Finally in Section VI conclusion and future work is presented.

II. RELATED WORK

There has been a lot of work on MPI communications performance of PC cluster. Most of work is performed on a heterogeneous Beowulf PC clusters. In our work we focus on studying the performance of MPI collective communications on homogeneous Beowulf PC cluster consisting of 20 identical machines. In [9] the authors have used MPIBench a software for benchmarking the performance of MPI functions using a highly accurate, globally synchronized clock. In [10] the authors have developed a MagPIe library which optimizes MPI's collective communication and have used a LogP model for short messages and LogGP model for long messages (Table 1). In [11] a nice comparison is made among the common

 ISBN: 978-1-61208-173-1

parallel communicaton models appearing in the vast literature, such as Hockney, LogP, LogGP, and PlogP. All these models appearing in the literature make an assumption of a homogeneous environment and are listed in Table 1 [11].

Table 1: Common Communication Models

Model	***Time***	***Parameters***
Hockney	$T = l + m/b$	l : is latency of network b: is bandwidth
Log P	$T= o + (k/w -1)*max (g,o) + L + o$	o: overhead time to transmit or receive g: gap min time interval between message p: number of processors l: upper bound on latency k: number of bytes in a message w: size of the network package in bytes
Log GP	$T= o + (k-1)G + L + o$	G: gap per byte for long messages
P Log P	Same as Log P but with each parameter being dependent on message size.	Same as Log P but dependant on message length m L: end to end latency

However analytical models can not truly replace actual performance measurements. In our work we present empirical results of the MPI collective communication routines. Such functions provide insight to communicating on both wireless and wired Peer-to-Peer systems.

III. EXPERIMENTAL METHODS

A. Performance Measurement

The factors which affect performance are many and may be listed as:

1. Hardware related: CPU clock speed and number of CPUs on motherboard, memory, and network adapters.
2. Network related: Type of hardware, cable, fast-ethernet, switches, and routers. Protocols used TCP/IP, UDP/IP and others.
3. Software related: Operating system type, user buffering, kernel buffering of data, types of MPI routines used for example collective communications. MPI eager and rendezvous protocols, efficiency of algorithms, and polling or interrupts.

From the above we can define execution time as a function of both hardware and software. Execution Time will be a function of topology, number of nodes, message size, switch, router, network adapter, type of algorithm, link type, overhead computer and operating systems, physical medium, MPI related, TCP/IP related, and Ethernet related. This is indeed a very complex function in which latencies of both hardware and software have to be considered. Hard disk, RAM, and cache access times with network interface card, PCI, and PCI Express transportation times need to be considered in calculation of execution time. Also the latencies of network devices should be considered such as the switches which are in the 2 to 20 µsec range [8]. In routers the processing delay due to software processes would be considerably higher. DSL or Cable internet connections have less than 100 milliseconds (ms) delays but less than 25 ms are desired. Satellite Internet connections have an average of 500 ms or higher latency. The peak theoretical bandwidth of a network connection is fixed by the technology used but the actual throughput (bandwidth) varies with time and is affected by high latencies. Excessive latencies on the network causes bottlenecks that hinder flow of data therefore decreasing effective bandwidth.

The total time of sending a message from one peer to another peer computer can be represented in terms of execution times and communication time as,

$$Time = t_{execA} + t_{comm} + t_{execB} \ , \qquad (1)$$

where t_{exec} time can be defined as,

$$t_{exec} = K \cdot C / f \ . \qquad (2)$$

where K is instructions per program, C is clock cycles per instruction and f is CPU frequency. The time involves the message's journey from the transmitting computer's memory, user space to kernel space to the network interface, through the physical medium, to the switch, and then to the receiver computer's network interface, and up to the application.

In Peer-to-Peer applications collective operations are rampant. Broadcast, scatter, and gather routines are common and their communication time depends on the size of each message, number of messages, interconnection structure, and network contention. The communication time can be written as

$$t_{comm} = t_s + m \cdot t_t + t_c \ . \qquad (3)$$

In the above equation t_s is the message latency assumed constant and includes the overhead time at the source and destination; t_t is the transmission time computed as $1/B$ where B is link bandwidth given in Q bits/sec; and m represents message to send. The contention time t_c is burst dependent and usually removed for simplicity. The time complexity is $O(m)$ for m data items. Usually one sends messages from one computer to multiple destinations. The 1-to-N fan-out broadcast is when the same message is sent to N destinations sequentially, then the communication time is

 ISBN: 978-1-61208-173-1

$$t_{comm} = N(t_s + m \cdot t_t), \quad (4)$$

and for the *scatter* and *gather* communication models is,

$$t_{comm} = N(t_s + m \cdot t_t / n). \quad (5)$$

In scatter a unique message is sent from source to every other destination and in gather a unique message is received from every other nodes.

The time complexity is $O(Nm)$ for one source connecting to N destinations. In (5), n is the total number of nodes and $N = n$-1. For a tree type structure the time complexity of 1-to-N fan-out will depend on number of nodes at each level and the number of levels. The disadvantage of a binary tree implementation is if one node fails then all the nodes below it will not receive the message.

For an Ethernet LAN network, communication time can be defined as:

$$t_{comm} = t_m + m_E \cdot t_t + t_q + t_p + t_E \quad (6)$$

where all terms depend on message size and message time at Media Access Control (MAC); t_m is dependant on message size and would be same for a homogeneous network and would be some factor multiplied by message size, $\alpha \cdot m_{size}$; m_E represents the number of bits in an Ethernet packet; t_q is the queuing delay; t_p is the propagation delay defined as d/c, where d is the length of the link and c is speed of light in medium having a value less than $3x10^8$ m/sec (e.g. Copper wire .77c); t_E is the Ethernet interframe gap which is 12 bytes (96 bits).
The traffic intensity can be defined as

$$T_i = (m_E \cdot p) / B \quad (7)$$

where p is the average packet arrival rate. Therefore T_i approaching close to 0 will indicate small delay; T_i approaching to 1 is an indication of large delay.

The communication overhead can affect transmission efficiency. The transmitted Ethernet packet has a payload which is TCP/IP encapsulated in addition to the application header. TCP header consists of 20 bytes and the IP header consists of at least 20 bytes. The transmitted Ethernet frame has a preamble of 8 bytes and an interframe gap of 12 bytes. The number of Ethernet packets per second on the link will be,

EthernetPacketsPerSecond = *B* / [8 *(*FrameSize* + *Interframe gap* 12 bytes + *Preamble* 8 bytes)] (8)

The Ethernet protocol efficiency is defined as,

Efficiency = Payload size/ Frame size (9)

and the throughput is

Throughput = Efficiency x B (10)

Therefore for every Ethernet packet on the link, a 96 bit interframe gap and 64 bits of preamble would be overhead. If the link has the capacity of 1 Gbps then for a minimum Ethernet frame size of 64 bytes transmitted, the link will consist of 7.62 x 10^8 bits/sec due to Ethernet frame and an overhead of 2.38 x 10^8 bits/sec due to interframe gap and preamble combined. For a Gigabit Ethernet the minimum frame size would be 512 bytes when operating in half-duplex mode. The Ethernet protocol efficiency is low for small packets (e.g. 54.76% for 64 bytes) and high (e.g. 97.53% for 1518 bytes) for large packets and hence the throughput is low for small packets and high for large packets.

Latency can have detrimental affects lasting few seconds or can be persistent depending on source of delays. Both bandwidth and latency are two main entities to measure network performance. Since software related latencies are hard to measure and define, one resorts to empirical methods as done in the next section.

B. Beowulf PC cluster Specifications

Our testbed consists of a PC cluster including 20 Lenovo machines with the following specifications: Intel Core™ 2 Duo CPU, E4400 2.00GHz, 1.00 GB of RAM. Network Card: Broadcom Netlink, Gigabit Ethernet, Driver date 8/28/2006 version 9.81.0.0. The PC are connected to a Gigabit D-Link Ethernet switch. Each machine has RedHat Enterprise AS Linux operating system installed, and use LAM 7.0.6/MPI 2.

C. MPI Benchmarks

The Message Passing Interface (MPI) is a standard interface that is broadly used with distributed computing applications [12]-[15]. The following MPI-based benchmarks are used to test the communication performance of the nodes:

a) *All-to-all*: every node sends a message to every other node.
b) *Broadcast*: one node sends one message to every other node.
c) *Gather*: all nodes send a different message to a single node.
d) *Scatter*: a single node sends a different message to every other node.
e) *Point-to-Point*: a single message is sent/received between 2 specific nodes.

The implementation details of the above collective communications are usually unknown to the programmer.

 ISBN: 978-1-61208-173-1

The MP1-1 standard specified the "blocking collective communications" only while the MPI-2 standard defines "non-blocking" routines which perform better with some applications. MPI related software performance will depend on the type of message passing protocols, *eager* (asynchronous communication) or *rendezvous* (synchronous), type of message buffering (user and system), sender-receiver synchronization (e.g. polling or interrupt), and efficiency of the algorithms used to implement the collective communication routines.

There are many benchmarking software available on the internet such as MPIBench or mpptest. However, most benchmarks available for collective communication basically use the following procedure to measure the execution time.

1. All processes arrive at Barrier
2. Start time
3. MPI_collective_fn
4. All processes arrive at Barrier
5. End time
6. PTime = End_t ime – Start_time

where MPI_collective_fn is one of the MPI functions all-to-all, gather, scatter, broadcast, and Point-to-Point.

D. Analysis of Execution Time (PTime)

In the benchmark procedure above, PTime consists of time to execute the MPI_collective_fn function and twice the time of Barrier.

First, the time spent at the transmitting computer would involve sending data by the kernel of system to the network interface card (NIC) and the time NIC takes to pack bytes in an Ethernet frame to send the frame on the physical wire. Depending on type of NIC architecture, a typical Ethernet NIC would have specifications as given in Table 2.

Table 2. Gigabit Ethernet Network Interface Specifications.

Speed	Interface	Data Width	Clock	Time for 1 byte transfer
1 Gbps	GMII	8 bits	125 MHz	8 ns
10 Gbps	XGMII	32 bits	156.25 MHz	.8 ns

To transmit an Ethernet frame the time to transmit from Medium Access Control (MAC) to Physical (PHY) for a 1 Gbps link would be

$$T_t = 8 \text{ ns} * Ethernet\_Frame\_size \quad (10)$$

One thing about Gigabit Ethernet is its clock rate at 125 MHz but more data is transmitted per time. The transfer rate is higher since 125 MHz x 2 bits per signal (i.e. per wire pair in Cat 5E cable) x 4 signals per time = 1 Gbps. Therefore, on the motherboard if PCI Express is available then with a maximum transfer rate of up to 250 MB/s then full speed of Gigabit Ethernet is achievable.

Next, the switch receives the Ethernet frame, and processes the frame with a typical delay of 2 to 20 µsecs [8]. It is then sent out to the destination computer where again the NIC on receiving computer processes it (PHY taking anywhere from 200 to 300 nanoseconds depending on technology) and sends it to the MAC layer, then onto the TCP/IP layers up to the application. The process of transmittance, reception, and acknowledgement is repeated according to TCP/IP, Ethernet framing, and depending upon the application's instructions, i.e. MPI collective_fn function and time of Barrier. The Barrier is used to explicitly control the flow of execution. There are at least 3 types of Barrier implementations [9], namely,

a) Counter implementation (linear barrier)
b) Tree implementation
c) Butterfly barrier

The time complexity of barrier with counter implementation is $O(n)$. For both the tree and the butterfly implementations the time complexity is $O(\log n)$, where n is the number of nodes. From the above, PTime depends on how MPI collective_fn and Barrier are implemented in LAM 7.0.6/MPI 2, as summarized in Table 3 (for large number of nodes).

Table 3. Time Complexity

Linear Model	PTime	
	Barrier Implementation	
	a) Counter	*b) Tree and c) Butterfly*
MPI_alltoall	O(nNm) + O(n)	O(nNm) + O(log n)
MPI_Bcast	O(Nm) + O(n)	O(Nm) + O(log n)
MPI_Gather	O(Nm) + O(n)	O(Nm) + O(log n)
MPI_Scatter	O(Nm) + O(n)	O(Nm) + O(log n)

IV. RESULTS

The MPI benchmarks are run on a PC cluster by first fixing the number of nodes in a communication group to 2 and varying the size of messages from x Kbytes to y Mbytes (2^n where n = 0,1,2,3,4…,). Then the number of nodes in communication group is iteratively incremented up to 20 nodes.

The figures show Minimum Round Trip (MRT) time measured in granularity of µsecs for messages of sizes ranging from 256 KB to 2 MB.

In Figure 1 all-to-all minimum round trip time is plotted versus number of nodes in a PC cluster. Fig. 1 show MRT is almost constant (with little variation) for a communication group consisting of anywhere from 2 to 9 nodes in comparison with 10 to 20 nodes which shows MRT linearly increasing.

In Figure 2, broadcast MRT time is plotted versus number of nodes. From the figure we observe MRT increases with increase in the number of nodes again with a steeper slope for large message sizes within each group. Note, from the figure it shows somewhat a step wise increment in MRT values.

 ISBN: 978-1-61208-173-1

In Figures 3 and 4, gather and scatter MRT time is plotted versus number of nodes, respectively. Both gather and scatter have a similar shape for MRT. The shape is more pronounced for large message sizes within each group. For gather and scatter communications it is seen from the figures MRT for small number of nodes e.g. 2, 3 and for large number of nodes, e.g. 16-20 takes on values which are much larger than the MRT of the number of nodes in between. MRT takes on a minimum for 8, 9 nodes and slightly higher for 10 nodes. It is also interesting to note that skewed shape flattens out as the message size is reduced. The figures of scatter and gather show unexpected behavior in MRT for nodes up to 9. The reason for this is presented under the discussion section.

In Figure 5, Point-to-Point benchmark is plotted with logarithmic scale for MRT and messages sizes ranging from 4 bytes to 8 Mbytes.

Comparing all-to-all communication with the others it is seen all-to-all has the highest MRT as expected.

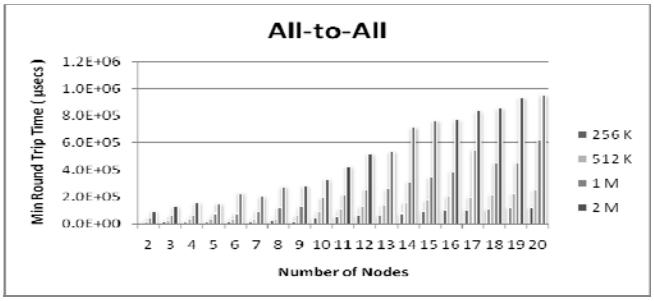

Figure 1. All-to-all benchmark showing MRT for message sizes of 256K B, 512KB, 1MB, 2MBand for different size of PC cluster.

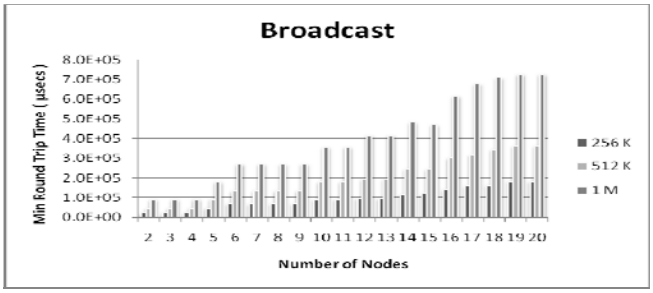

Figure 2. Broadcast benchmark showing MRT for message sizes of 256KB, 512KB, 1MB and for different size of PC cluster.

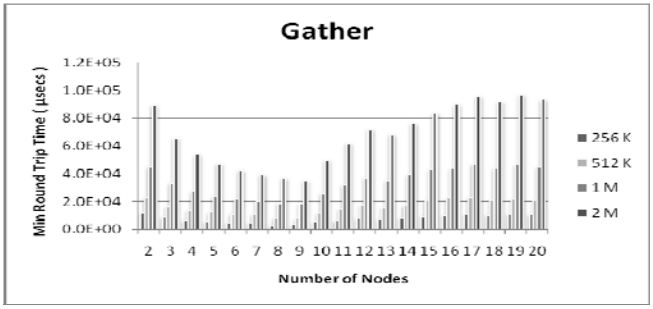

Figure 3. Gather benchmark showing MRT for message sizes of 256KB, 512KB, 1MB, 2MB and for different size of PC cluster.

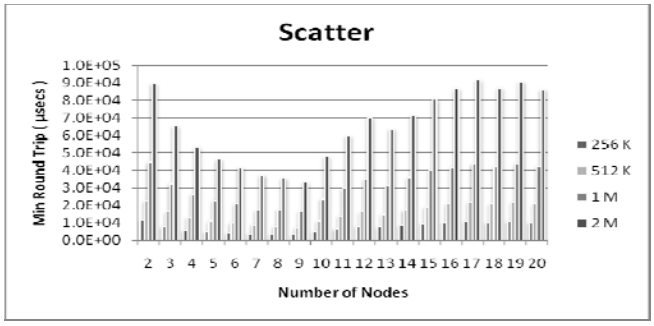

Figure 4. Scatter benchmark showing MRT for message sizes of 256KB, 512KB, 1MB, 2MB and for different size of PC cluster

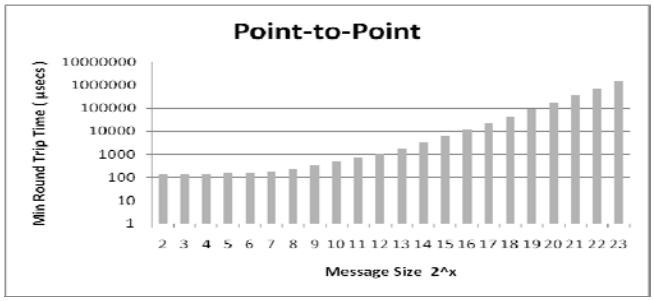

Figure 5. Point-to point benchmark showing MRT log scale versus message sizes of 4 bytes to 8 Mbytes.

V. DISCUSSION

In this section we discuss the results of all-to-all, broadcast, gather, scatter, and Point-to-Point functions used in the benchmarks. First note, the message size being transmitted is fixed in the case of broadcast and Point-to-Point. In the second case of all-to-all, gather, and scatter, the message size is divided equally within the communication group and depends on the number of participating nodes. Let m denote the message size and n denote the number of nodes,

$$s_i = m / n_i \quad \text{for } i = 2,3,...,20 \tag{11}$$

where s_i is the actual message size being transmitted or received by each communicating node. Therefore, $s_1 > s_2 > s_3 \cdots > s_{20}$ since $n_i < n_{i+1}$. As the number of nodes are increased the message size being sent or received goes down. From (5) one knows that latency depends on message size (and of course is a function of time): as the message size s_{i+1} is less than s_i therefore the per message latency of s_i is larger than s_{i+1}. However this is not always the case as seen in the scatter and gather routines. When a host application transmits to its destination, non-blocking sends are posted by MPI,

reducing latency for certain nodes (e.g. 9 in the scatter figure) and the TCP protocol is used for reliability. It must wait for a period of time to receive an acknowledgment. If the reply does not arrive within the expected period the data is retransmitted. On Ethernet LAN the wait time is not more than a few μsecs. Thus, the overhead time has a major affect on latency.

A number of factors together are involved in having a major affect on MRT as shown in the scatter and gather plots. First, varying the size of the message has affect. Second, the implementation of LAM-MPI routines are not known to the programmer, which model is being used whether linear or tree type. Third, which protocol MPI is using either eager or rendezvous protocols. The LAM-MPI constructs the message and sends it through the network to the destination computer which must accept and act upon the message contents. LAM-MPI uses either the eager protocol or the rendezvous protocol depending on message size. With the eager protocol, as soon as a message is posted both the envelope and data are sent to the destination. If on the destination the receive operation is not posted then buffering has to be done; this buffering involves an additional data duplication. In the rendezvous protocol, when a message is posted the envelope is sent to the destination and buffered. As soon as a receive is posted the destination sends an acknowledgement to the sender which then only will be the data send by the sender In this case buffering of the data is avoided and used for large messages.

Fourth, TCP/IP protocol is being used by LAM-MPI protocols on top of TCP/IP protocols which cause higher latencies in communicating a message from sender to destination. The actual application bytes packed in an Ethernet frame are much less due to MPI application, TCP/IP, and Ethernet headers. In some cases, as depicted in the figures of scatter and gather, when the message size decreases among the communication nodes it can happen that MRT decrease down to a certain value and, as the message size is further decreased beyond the minimum MRT point, the MRT will start to increase again. The explanation of such a behavior is due to the factors mentioned above (i.e. the implementation of the scatter and gather algorithms in MPI, eager and rendezvous protocol switching depending on message size, plus TCP overhead and the increase of overhead ratio as the Ethernet frame size decreases).

VI. CONCLUSION AND FUTURE WORK

In this paper we have studied the performance of MPI collective communications routines on a gigabit Ethernet LAN. The experiments were performed to represent a Peer-to-Peer scenario in which one has different sizes of nodes in a group and variations in message size. All the benchmark routines presented closely represent typical communications of a Peer-to-Peer system i.e. broadcasting, gathering, scattering, all-to-all, and point-to-point. The results show in the case of gather and scatter that as the message size is decreased among the increasing nodes there is no set predictable pattern MRT takes. We have seen that many factors affect the performance of collective communications in a wireline (Ethernet) environment. Our next target is to extend our study to the area of Peer-to-Peer over wireless networks.

ACKNOWLEDGMENT

The authors would like to thank CSSE Research Center for carrying out experiments on the PC cluster. The authors would also thank the anonymous reviewers for their valuable and insightful comments.

REFERENCES

[1] S. Lin, A. Pan, R. Guo, and Z. Zhang., “Simulating Large-Scale Peer-TO-Peer Systems with WiDS Toolkit”, White Paper, Microsoft, Jan. 2008.

[2] M. Li, W. Lee, and A. Sivasubramaniam, “Efficient Peer-to-Peer Information Sharing over Mobile Ad Hoc Networks”, In MobEA, 2004.

[3] X. M. Huang, C.Y. Chang, and M.S. Chen, “PeerCluster: A Cluster Based Peer-to-Peer Sytem”, IEEE Transactions on Parallel and Distributed Systems”, vol. 17, No. 10, Oct. 2006, pp. 1110-1123.

[4] B. Parviz and K. Miremadi, “Building a Peer to Peer Message Passing Environment by Utilizing Reflection in .NET.”, In Proceedings of PDPTA'2006. pp.1096~1102.

[5] InfiniBand Trade Organization., http://www.infinibandta.org/

[6] http://en.wikipedia.org/wiki/Quadrics, Sept. 3, 2011.

[7] Myricom Inc., http://www.myri.com, Sept 3, 2011.

[8] CISCO Inc., http://www.cisco.com, Sept 3, 2011.

[9] F. A. Vaughan, D. A. Grove, and P. D. Coddington, “Communiation Issues for Two Cluster Computers,” ACSC ’03 Proceedings of the 26th Austrailasian computer science conference, vol 16, 2003.

[10] T. Kielmann and H. E. Bal, “Fast Measurment of LogP Parameters for Message Passing Platforms”, 4th Workshop on Runtime Systems for Parallel Programming (RTSPP), pp. 1176-1183, held in conjunction with IPDPS 2000, Cancun, Mexico, May 1-5, 2000. Lecture Notes in Computer Science, Vol. 1800.

[11] J. P. Grbovic, et al, “Performance Analysis of MPI Collective Operations”, Journal Cluster Computing, Vol 10, Issue 2, June 2007, pp. 127-143.

[12] R. Riesen, “Communicaton Patterns”, Parallel and Distributed Processing Symposium, 25-29 April 2006.

[13] A. Leko, et al, “Practical Experiences with Modern Parallel Performance Analysis Tools : An Evaluation”, Parallel and Distributed Processing, IPDPS 2008 IEEE Symposium 14-18 April 2008, Miami, Fl, pp. 1-8.

[14] B. Wilkinson and M. Allen, Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers, Second Edition, Pearson Prentice Hall, 2005.

[15] F. Noor and S. Misbahuddin, “Using MPI on PC Cluster to Compute Eigenvalues of Hermitian Toeplitz Matrices”, Lecture Notes in Computer Science, 2010, vol 6081, pp 313-323.

 ISBN: 978-1-61208-173-1