Message Passing for Linux Clusters with Gigabit Ethernet Mesh Connections

Jie Chen, William Watson III
HPC Group
Jefferson Lab
Newport News, VA 23606
{chen,watson}@jlab.org

Robert Edwards
Theory Group
Jefferson Lab
Newport News, VA 23606
edwards@jlab.org

Weizhen Mao
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795
wm@cs.wm.edu

Abstract

Multiple copper-based commodity Gigabit Ethernet (GigE) interconnects (adapters) on a single host can lead to Linux clusters with mesh/torus connections without using expensive switches and high speed network interconnects (NICs). However traditional message passing systems based on TCP for GigE will not perform well for this type of clusters because of the overhead of TCP for multiple GigE links. In this paper, we present two os-bypass message passing systems that are based on a modified M-VIA (an implementation of VIA specification) for two production GigE mesh clusters: one is constructed as a 4x8x8 (256 nodes) torus and has been in production use for a year; the other is constructed as a 6x8x8 (384 nodes) torus and was deployed recently. One of the message passing systems targets to a specific application domain and is called QMP and the other is an implementation of MPI specification 1.1. The GigE mesh clusters using these two message passing systems achieve about 18.5 μs half-way round trip latency and 400MB/s total bandwidth, which compare reasonably well to systems using specialized high speed adapters in a switched architecture at much lower costs.

1. Introduction

During the past decade clusters running the Linux operating system (OS) using Intel X86 CPUs offer attractive platforms for those who seek parallel computing systems with high performance-to-cost ratios. Like traditional massive parallel processors (MPP), these clusters have to be organized with network interconnects (NICs) to allow message passing among all nodes. The choice of the network topology or architecture for a cluster has some influence on the scalability and performance of the cluster. There are several commonly used network topologies such as switched networks, meshes (tori), trees and hypercubes [16], but a majority of clusters prefer the switched network topology which offers a fully connected network. Deploying the switched topology requires expensive network switches in addition to specialized NICs that provide high bandwidth and low latency. However, for certain parallel applications that require communications occurring mostly between nearest neighbors, the mesh (torus) topology may offer a better performance-to-cost ratio in comparison to a switched architecture because of the removal of expensive switches. A single Gigabit Ethernet (GigE) [23] link may not provide as much bandwidth as a high performance NIC such as Myrinet [5], but multiple GigE links in a mesh could provide comparable total bandwidth and adequate latency. Fortunately, recent progress in performance coupled with a decline in price for copper-based GigE interconnects (adapters) makes them ideal candidates for constructing Linux clusters with mesh connections.

Currently, there are a few TCP based message passing systems[12] for Linux clusters with GigE mesh network topology. The portable MPI [31] implementation MPICH [14] over TCP (MPICH-P4) actually can work for a mesh topology with careful setups of routing tables, host names and host ranks even though it is designed and implemented for a switched network environment.

TCP based message passing systems rarely approach the GigE raw hardware performance because of the overhead of kernel involvement and multiple copies during data transfers [18]. To reduce this type of overhead, an industry standard called Virtual Interface Architecture (VIA) [21] stemmed from a technique dubbed "user-level networking" or ULN [27], which removes the kernel from the critical paths of sending and receiving messages, has been in place for several years. There are a few VIA implementations among which the M-VIA [32] is the only one that is appropriate for GigE adapters. There is indeed an implementation of MPI over M-VIA called MVICH [33] but it does not work for mesh topology.

There are several message passing systems that are based on the ULN approach for GigE adapters, but they are ei-

ther not working for mesh topology or not implemented as hardware independent. For instance, there are EMP [24], GAMMA [7], and PM/Ethernet-kRMA [25]. The first one is a truly zero-copy OS-bypass message passing system, but it has been implemented only for a set of GigE adapters which are not inexpensive. The second one uses customized linux network device drivers to deliver low latency and high bandwidth, but it is designed explicitly for a switched topology. The last one actually uses multiple GigE adapters along with even more GigE switches to achieve high bandwidth.

The primary mission of two production Linux clusters with GigE mesh connections deployed at Jefferson Lab (Jlab), which is a national laboratory, is to carry out Lattice Quantum Chromodynamics (LQCD) [28] calculations which describes the strong interaction among quarks. An LQCD calculation is carried out in a 4-dimensional (4-D) box of points, which approximates a portion of the space-time continuum. Specifically, each node in a cluster operates on a regular 4-D sub-lattice, calculating determinants and inverses of 3x3 complex matrices and communicating 3-dimensional (3-D) hyper-surface data to adjacent nodes, i.e., utilizing nearest-neighbor communication, in each iterative step after which a global reduction, one type of collective communications, is carried out. However, these clusters may also be used for other scientific calculations requiring more complex communication patterns. Therefore two message passing systems have been implemented for the clusters: one is QMP (QCD message passing) [34] focusing on the LQCD applications; the other one is an implementation of MPI specification 1.1 offering wider capabilities to other applications. Both systems are derived from a common core that is based on a modified M-VIA communication software. Consequently two systems should perform similarly.

In this paper, we first give a brief overview of VIA and describe the hardware and software environment for our work. We then point out what modifications we have made to the original M-VIA, and evaluate and compare the performance of TCP and the modified M-VIA for Linux clusters of mesh topology using GigE adapters. Subsequently we present our design and implementation of our message passing systems with emphasis on both point-to-point and collective communications. Furthermore, an application benchmark results for a GigE mesh and a Myrinet switched cluster are given to demonstrate that our GigE mesh clusters are indeed cost effective platforms at least for LQCD calculations. Finally we conclude our work and discuss future research.

2. Virtual Interface Architecture (VIA)

The Virtual Interface (VI) architecture eliminates the operating system overhead by providing each process with a protected, directly accessible interface to the network hardware - a Virtual Interface. Each VI represents a communication endpoint, and pairs of such VIs can be connected to form a communication channel for bi-directional point-to-point data transfers. The operating system is only involved in setting up and tearing down VI communication channels and pinning data buffers from which interconnects can safely DMA data, but it is no longer in the way of the critical paths of data transfers. Figure 1 shows an organization view of the VI architecture that consists of several components such as VI, Completion Queues, Send/Recv Queues and so on.

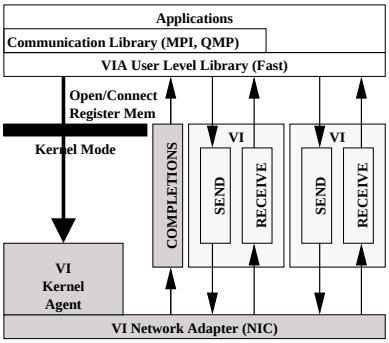

Figure 1: The VI Architectural Model

Currently VIA supports three levels of communication reliability at the NIC level: Unreliable Delivery, Reliable Delivery and Reliable Reception. In addition, VIA also supports two different styles of communications: send/receive, and remote memory access (RMA).

3. Hardware and Software Environment

There are two production GigE mesh Linux clusters at Jlab. One that has been in production use for a year containing 256 nodes is arranged as either a $4 \times 8 \times 8$ or a $4 \times 4 \times 16$ 3-D mesh with wraparound links (torus). Each node has a single Pentium 4 Xeon 2.67 GHz processor with 256 MB of memory and is connected to its six neighbors by three dual port Intel Pro/1000MT GigE adapters [30] situated on three PCI-X slots (133/100MHz 64bit). The other cluster that was deployed recently containing 384 nodes is configured as a $6 \times 8 \times 2^3$ 5-D mesh that can be projected to different 3-D mesh configurations with wraparound links (torus). Each node has a single Pentium 4 Xeon 3.00 GHz processor with 512 MB of memory and is connected to its six neighbors by three same type of adapters along with an additional on-board GigE adapter of the same type. (From now on we use mesh to refer to a mesh with wraparound links unless otherwise specified.) All performance data presented in this paper are collected on the first cluster. Performance data in Section 6 for Myrinet networks are collected on a 128-node cluster of 2.0GHz Pentium 4 Xeon connected through Myrinet LaNai9 adapters and a Myrinet 2000 switch that

has a full-bisection Clos [8] topology. Each GigE adapter costs $140, with a total expenditure of $420 for networking components on a single node, which is much less expensive than the price of single port (∼$1000) for either Myrinet or infiniband networks. All nodes are running RedHat Linux 9 with kernel version 2.4.26. The modified M-VIA version is based on M-VIA 1.2 and the M-VIA driver for Intel Pro/1000MT adapters is developed locally [6] and is based on the Intel e1000 driver version 5.2. The M-VIA GigE driver is loaded with options of 2048 transmission descriptors and 2048 receiving descriptors to enhance the capability of pipelining and to reduce possible network contentions. Furthermore, the GigE driver is tuned to utilize interrupt coalescing of the Intel adapters by selecting appropriate values for some driver parameters such as interrupt delay.

4. M-VIA and TCP

In order to understand the benefits of using M-VIA as a low-level communication software on which MPI and QMP are based, several key performance benchmarks for M-VIA and TCP are presented. The benchmark results illustrate the point that M-VIA is far superior in terms of performance to that of TCP, and delivers high bandwidth and adequate latency for multiple GigE adapters in mesh topology.

The modified M-VIA has two major changes made to the existing M-VIA 1.2 in addition to a few bug fixes. The first is to allow hardware/software checksum to be performed on each Ethernet packet. The second is to enable packet switching so that non-nearest neighbor communications are possible. The Intel e1000 M-VIA driver developed at Jlab takes advantage of Intel Pro/1000MT hardware checksum capability to checksum each packet without degrading performance. The mesh geometry information is injected into the M-VIA driver so that packet routing is possible.

4.1. Point-to-Point Latency and Bandwidth

Latency and bandwidth are the two most important benchmarks to describe a communication system [10]. A latency value describes how long it takes a single packet to travel from one node to another node and it comprises several components such as host sending/receiving overhead and NIC processing time [9]. By studying this benchmark closely, improvements may be made to reduce the host overhead of a communication system. The latency performance data are obtained as usual by taking half the average round-trip time for various message sizes.

Network bandwidth describes how much data can be pushed through the network in a unit time and thus is critical to parallel program performance because higher bandwidth decreases occupancy and the likelihood of contention. In contrast to the relative simple way to obtain latency values, bandwidth values can be obtained in several different ways which characterize different communication patterns. Two types of bandwidth values are of interest: bidirectional ping-pong bandwidth in which data travel in both directions alternatively; bidirectional simultaneous bandwidth that simulates data transfers in both direction simultaneously. The first type of bandwidth reveals an upper bound of the performance because of no loaded CPUs are involved. On the other hand, the second type of bandwidth can reveal the effect of CPU overhead on sending and receiving.

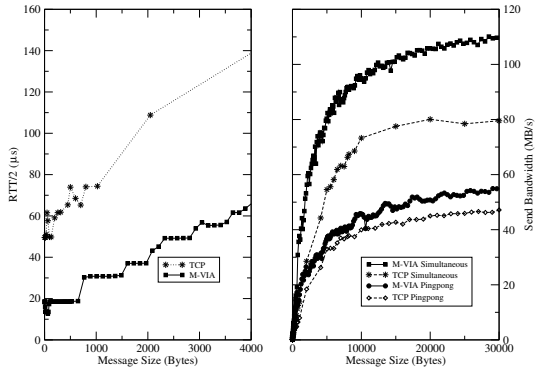

Figure 2: M-VIA Point-to-Point Latency and Bandwidth Values

Figure 2 summarizes both latency and bandwidth results for M-VIA and TCP. The bandwidth results are sending bandwidth alone not counting receiving data. The latency and bandwidth results of M-VIA are consistently better than those of TCP for all the test data sizes. The latency of TCP is at least 30% higher than the M-VIA latency which is around 18.5μs for messages of size smaller than 400 bytes. This illustrates that M-VIA, which has around 6μs of sending and receiving host overhead [6] by eliminating memory copies on sending and by using only one memory copy on receiving, can indeed deliver adequate latency for parallel applications. However, the M-VIA does not provide latency values approaching the sub 10μs range because of one memory copy on receiving and expensive kernel interrupts generated by GigE adapters. The simultaneous send bandwidth of M-VIA is approaching 110MB/s for not very large message sizes. It is 37% better than that of TCP in comparison to the marginally better results for the other type of bandwidth. This is not surprising since this type of test really reveals host overhead and any reduction in sending and receiving overhead produces larger differences in the results.

4.2. Aggregated Bandwidth

Single link bandwidth values presented in the previous subsection cannot address the issues of aggregated or usable bandwidth, which is the sum of the simultaneous bandwidth

of each GigE link within a single user process of a node in a mesh network topology. There are 6 GigE links in a 3-D mesh meanwhile there are 4 GigE links in a 2-D mesh. Figure 3 presents aggregated send bandwidth values of a node for a 2-D and a 3-D mesh.

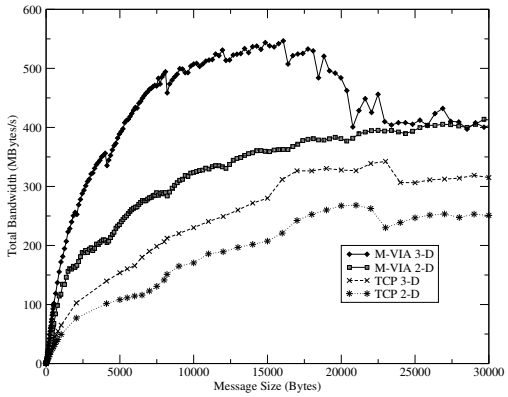

Figure 3: M-VIA Multi-dimensional Aggregated Bandwidth Values

The M-VIA aggregated bandwidth for a 2-D mesh increases smoothly and flattens off around 400 MB/s, which is similar to the result obtained by PM/Ethernet-kRMA [25], corresponding to roughly 100MB/s along each link. In contrast, the aggregated bandwidth for a 3-D mesh peaks around 550 MB/s and eventually drops to 400 MB/s. The fall off of the M-VIA 3-D bandwidth values in the end of the tested region is due to the overhead of one memory copy in M-VIA upon receiving data in addition to difficulties of fully pipelining the 6 GigE links in a single process.

From the above latency and bandwidth results, M-VIA indeed provides excellent bandwidth and adequate latency for parallel computing. In particular, the aggregated bandwidth values for multiple connections in a 3-D mesh are on par with bandwidth values of switched Myrinet networks[29] at a fraction of the cost.

5. Message Passing Systems: MPI/QMP

To facilitate parallel message passing on the Jlab mesh clusters, two message passing systems are implemented on top of common components that are based on the customized M-VIA system. One is QMP focusing on LQCD calculations with a subset of functionalities of MPI. The other is an implementation of MPI 1.1 specification. These two systems perform the same on key benchmarks, therefore from now on we refer these two systems as MPI/QMP.

The MPI/QMP implementation over M-VIA in mesh topology aims to maximize performance by utilizing M-VIA communication support to eliminate overhead and to overlap communication and computation. It provides a low-cost point-to-point and collective communication system. The basic design principles of MPI/QMP are listed below:

- high performance exploitation of M-VIA communication support such as remote memory access (RMA);
- low-latency and high-bandwidth point-to-point communication with receiver-side message matching and zero-copy communication using RMA; and
- efficient collective communication algorithms for mesh network topology.

5.1. Point-to-Point Communication

The design of MPI/QMP point-to-point communication is strongly influenced by capabilities of the underlying M-VIA software. To achieve high bandwidth and low latency, several important design choices have to be made. The following highlights our decisions.

First, each node creates and maintains 6 VIA connections to its nearest neighbors. Each connection has pre-posted VIA sending and receiving descriptors along with memory buffers used for copying applications' small messages during data transfers.

Second, each connection maintains a list of tokens to regulate data flow on the connection, since M-VIA has no built-in flow control mechanism. The number of tokens represents the number of VIA receiving buffers currently posted on the receiving end of a connection. This number is constantly updated to the sender by either a piggybacked application message or an explicit control message.

Third, different communication schemes are used for small and large messages. The RMA capability of M-VIA enables MPI/QMP to implement zero-copy data transfers. However, using the RMA capability of M-VIA in sending and receiving implies synchronous or rendezvous communication semantics that means a send call is not completed unless the corresponding receive call is issued. Therefore RMA technique cannot be used for messages of small sizes since synchronous communications increase application latency. For messages of small sizes ($<$16K bytes), a so-called *eager* protocol is used such that a sending messages are copied into pre-posted memory buffers which then are DMAed into GigE adapters and received messages are DMAed from GigE adapters to pre-posted buffers which are copied to user receiving buffers when ready. For messages of large sizes, bandwidth and reducing host overhead are more important. A remote memory write and sender-side matching technique [26] is therefore used.

Finally, communication to non-nearest neighbors is enabled by kernel-level packet switching provided by the modified M-VIA. The routing algorithm is a simple Shortest-Direction-First algorithm which chooses the direction that has the smallest number of remaining steps for a packet.

Figure 4 presents some of point-to-point communication benchmark results. The small insert shows the usual

point-to-point half-way round trip latency values that are around 18.5μs for small messages, which illustrates small implementation overhead of MPI/QMP. The 2-D and 3-D aggregated bandwidth results of MPI/QMP are clearly less than that of M-VIA, which is partly due to flow control and synchronous RMA control messages. Nonetheless MPI/QMP still exhibits around 400MB/s total bandwidth for 3-D mesh. The sudden jump in bandwidth values around 16000 bytes is a direct result of switching from using memory copies to RMA in MPI/QMP implementation.

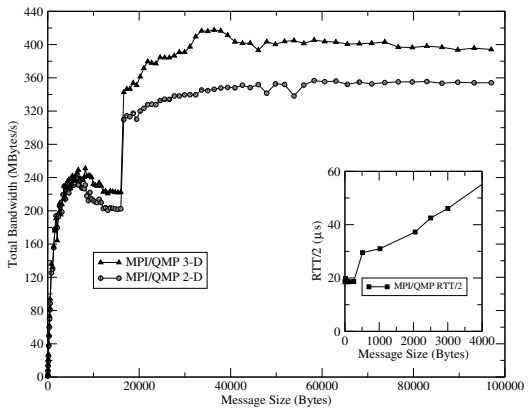

Figure 4: MPI/QMP Point-to-point Performance

The packet switching for non-nearest neighbors is handled at kernel interrupt level without copying data to and from user space. This effectively removes some of the M-VIA sending and receiving overheads and leads to a $\sim$12.5μs node-to-node routing latency. Therefore the point-to-point latency between any non-nearest-neighbor is $12.5\mu s \times (n-1) + 18.5\mu s$, where n is the number of steps between the nodes.The switching throughput is dictated by the aggregated bandwidth, which is around 500MB/s, of 6 GigE adapters. However the point-to-point bandwidth of two non-nearest neighbors under no network contention has the similar results of two nearest neighbors.

5.2. Collective Communication

Collective communication [11] involves global data movement and global control among all nodes in a cluster and has received a lot of attention in parallel community in the past few decades [22], but most algorithms have been designed for the MPPs with mesh topology using dedicated routers or switches on each node [1][4]. Fortunately, some of the fundamental algorithms can still be applied to a Linux cluster with 3-D GigE mesh connections since each Linux node can behave as a store-and-forward switching node with six full-duplex communication channels, and has multi-port capability which means at any time each node can communicate with some of its neighbors simultaneously. To simplify our discussions, we will name a node i in the mesh with its coordinates, e.g., (x_i, y_i, z_i) in the 3-D case, and use $xdim$, $ydim$, and $zdim$ as sizes of the mesh in dimensions x, y, and z respectively.

A broadcast is implemented via a simple algorithm that a broadcast message travels along a x axis first, then cross an xy plane and finally through all yz planes. A reduction behaves very much like a reverse of a broadcast except that each node carries out some reduction operations, such as sum, before forwarding the reduced value to its neighbors. The number of communication steps of the broadcast and reduction algorithms are roughly $xdim/2 + ydim/2 + zdim/2$.

A basic scheme of global combining algorithm [20] is based on first reducing all messages to a node which then broadcasts the reduced value to all the other nodes. This algorithm takes roughly twice as many communication steps as the broadcast algorithm does. A barrier synchronization is implemented as global combining with a null reduction.

Figure 5 presents timing results of broadcast and global sum of integers for different message sizes on a $4 \times 8 \times 8$ mesh. The broadcast for messages of small sizes takes about 200μs for 10 communication steps, i.e., 20μs per step, which is in line with the MPI/QMP latency value of 18.5μs. The linear increase of the broadcast timing results is direct related to the linear increase of point-to-point latency results. The timing results of global sum are roughly twice as large as those of broadcasts, which confirms what the global sum algorithm suggests in the previous paragraph.

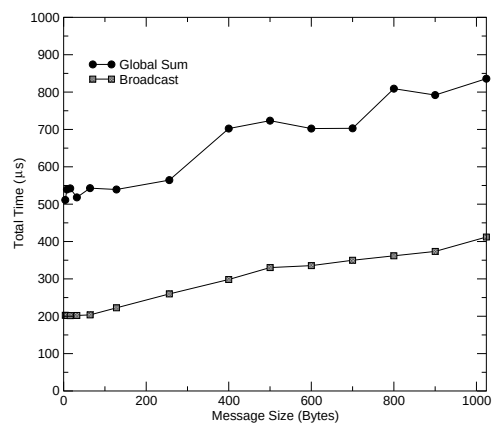

Figure 5: Broadcast and global sum performance results

A typical LQCD calculation needs it own input data which are initially stored at the root node. Before the parallel calculation begins at each node, a major preparation step is to dispatch all input data from the root to the nodes where they belong to. This is actually the one-to-all personalized communication (scatter) problem [11][2][3][17]. On average, a typical LQCD calculation involves sending

messages (small-size data) from the root to the messages' corresponding destinations at least ~250,000 times. This makes it necessary and beneficial to design optimal or near-optimal routing algorithms to handle the scatter communication problem. Moreover the algorithm for all-to-one personalized (gather) communication is simply the reverse of the scatter algorithm.

The goal of the scatter problem is to dispatch, in the store-and-forward (packet-switching) manner, all messages from the root to their corresponding destinations in the minimum number of steps (hops). In general, an algorithm for the problem must contain two components. The first component is the selection of messages to send for each time step. When the number of messages residing at a node is more than the number of ports of the node, a message or a set of messages has to be selected to be sent in the next time step. The second component is the actual routing. After a message is selected, a direction (among four in the 2-D case and six in the 3-D case) needs to be chosen to send the message in the next time step.

Our first algorithm is called Shortest-Direction-First (SDF). At any time step when there are nodes in the mesh holding messages that have not reached their destinations, each of these nodes selects a message to send using the First-Come-First-Serve principle and then send it along a direction chosen by the Shortest-Direction-First principle, i.e., the direction in which the message has the smallest number of remaining steps will be chosen. The algorithm does not dispatch messages in the shortest amount of time, thus is not optimal. But its simple design makes the implementation relatively easy.

In the scatter problem, the root is the bottleneck for delays since all messages are initially at the root and have to be sent one by one (in the single-port mode) or group by group (in the multi-port mode) toward their respective destinations. Our second algorithm, OPT, achieves optimality in several ways. First, each message will travel the shortest distance possible. For example, in the 3-D case, the distance that a message i with destination (x_i, y_i, z_i) will travel is $distance(i) = \min\{x_i, xdim - x_i\} + \min\{y_i, ydim - y_i\} + \min\{z_i, zdim - z_i\}$, assuming that the root node is the origin of the coordination system with zero coordinates in all dimensions. Second, messages are selected using the Furthest-Distance-First principle [17]. That is, a message that will travel the furthest distance to reach its destination will be among the first to be sent from each node. Third, and most importantly, the algorithm minimizes potential network contentions by a partitioning scheme. The mesh is first partitioned into roughly equal-size regions, with one region corresponding to one link leaving the root, and all nodes in the region are accessible from the link in the smallest number of steps possible. Consequently, all messages are also divided into groups, with one group corresponding to one region and with all messages of destinations falling into the region being put in the corresponding group. The partition establishes the 1-1-1 correspondence among a link leaving the root, a region of nodes, and a group of messages. To spread out the congestion at the root, messages in group i will be sent via link i to leave the root to enter region i. And from then on, the message will stay within region i and travel to its destination without any delay in the smallest number of steps.

Let k be the number of ports at a node. Let p be the number of nodes in the mesh. Then the root will need at least $(p-1)/k$ steps to send out all $p-1$ messages. Our algorithm OPT uses exactly $(p-1)/k$ steps to send out all messages from the root and guarantees that once a message leaves the root, it travels within its region without any delay in the fastest way until reaching the destination. Furthermore, the movement of messages in different region is parallel while the movement of messages in the same region is sequential in a streamline fashion without collisions among messages. Therefore, OPT is optimal. The time (measured by the number of steps) needed for OPT to dispatch all messages to their destinations is $\max\{T_1, T_2\}$, where $T_1 = (p-1)/k$ is the time for the root to send out all messages to their regions and $T_2 = max_{1 \le i \le p-1}\{distance(i)\} + c$ is the time to dispatch the message with the furthest distance to travel plus c, which is the number of additional messages with the same distance to travel in the same region. The term c is a very small constant (usually 0 and sometimes 1), thus may be ignored.

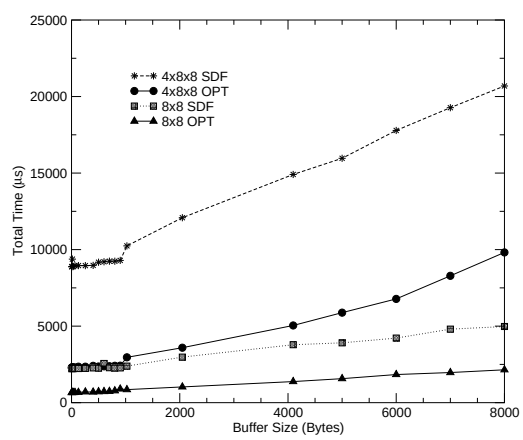

Figure 6: Personalized one-to-all communications

We have implemented both SDF and OPT algorithms for an 8×8 configuration and a $4 \times 8 \times 8$ configuration of the mesh cluster currently deployed at Jlab. Figure 6 presents the timing results of these two algorithms. SDF is easier to implement and OPT requires more overhead cost in obtaining the partition beforehand. However, on average, OPT dispatches messages to their destinations almost four times faster than SDF does for either cluster configuration across all tested message sizes. In addition our OPT algorithm

scales very well from the 8×8 configuration to the $4 \times 8 \times 8$ configuration for most part of tested message sizes except for the large sizes where simultaneous sends from the root node using six links become difficult.

Finally an all-to-all personalized communication is implemented as a parallel execution of every one-to-all personalized communication from all nodes.

6. Performance-to-cost Ratio

The Jlab Linux clusters with GigE mesh connections are indeed less expensive than clusters that are using switched architecture with expensive NICs due to much less expensive adapters and absence of switches. However, whether the clusters offer better performance-to-cost ratios needs to be verified by real applications in addition to the bandwidth and latency benchmarks. Therefore an LQCD benchmark application is carried out on a cluster with Myrinet switched connections and the $4 \times 8 \times 8$ GigE mesh cluster. The benchmark application is compiled on the GigE cluster using QMP/MPI and on the Myrinet cluster using vendor supplied MPI. The benchmark results are normalized to a single node for a fair comparison. The performance-to-cost ratio values are calculated based on the costs at the time of the GigE cluster installation. The results are summarized in table 1.

Table 1: Normalized LQCD benchmark results and estimated dollars per Mflop values

Lattice size	4^4	6^4	8^4
Myrinet(Gflops)	0.91	1.3	1.3
Myrinet($/Mflops)	3.29	2.3	2.3
GigE(Gflops)	0.73	0.95	1.12
GigE($/Mflops)	2.74	2.10	1.78

The LQCD benchmark code performs a little better in the switched Myrinet cluster than it does in the GigE mesh cluster since multiple GigE adapters post more host CPU overheads and QMP/MPI has higher latency values. The gradual increase of GigE performance with respect to the lattice size is clearly an indication of decreasing in surface-to-volume effect [13]. Most of all, the estimated dollar per mega-flops values of the GigE mesh cluster are certainly smaller than those of the Myrinet cluster, which means better performance-to-cost ratios for the GigE mesh cluster.

7. Conclusions

In this paper we present MPI and QMP message passing systems for Jlab production Linux clusters with mesh connections using GigE adapters. They are based on a modified M-VIA, which provides excellent bandwidth and adequate latency for parallel computing. Careful design and implementation of MPI/QMP yields a high performance point-to-point and collective communication system. Specifically, MPI/QMP delivers an excellent aggregated bandwidth of 400MB/s for multiple GigE links in mesh topology, which is on par with what a single Myrinet link offers in a switched network architecture. In addition MPI/QMP provides adequate latency of 18.5μs. Furthermore, MPI/QMP utilizes efficient broadcast, reduction, and global combining algorithms and an optimal scatter/gather algorithm to deliver low collective communication latency. Most of all, LQCD calculations using MPI/QMP in the Linux clusters with GigE mesh connections achieve better performance-to-cost ratios than the same applications can do using vendor-supplied MPI implementation with a Myrinet switched network architecture.

To further improve the performance of collective communication, we have been investigating the possibility of applying a technique called NIC-based or NIC-assisted global reduction [19]. Since the Intel GigE adapters have no on-board programmable processor and not enough RAM, we are working on a scheme of interrupt-level based collective communication, in which intermediate collective communications are carried out in the kernel space. This method eliminates the overhead of copying data to user space for the intermediate steps, therefore reduces the overall latency. In addition, we are investigating a possible new M-VIA feature, that is similar to the NAPI [15] appeared in Linux kernel 2.6, to reduce the cost of OS-interrupts generated by multiple GigE cards. Currently we are actively integrating new features into the future QMP/MPI releases and are porting M-VIA to Linux kernel 2.6. The current MPI/QMP implementation can be found at ftp://ftp.jlab.org/pub/hpc/QMP/QMP-mvia-mesh.tar.gz.

Acknowledgment

This work is supported by the Department of Energy, Contract DE-AC05-84ER40150.

References

[1] M. Barnett, R. Littlefield, D. Payne, and R. van de Geijn, Global Combine on Mesh Architecture with Wormhole Routing, *Proceedings of the 7th International Parallel Processing Symposium*, 13-16, 1993.

[2] D. Barth and P. Fraigniaud, Approximation algorithms for structures communication problems, *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*, 180-188, 1997.

[3] S. Bhatt, G. Pucci, A. Ranade, and A. Rosenberg, Scattering and gathering messages in networks of proces-

sor, *IEEE Transactions on Computers*, **42**(8), 938-948, 1993.

[4] S. Bokhari and H. Berryman, Complete Exchange on a Circuit Switched Mesh, *Proceedings of the Scalable High Performance Computing Conference*, 300-306, 1992.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, Myrinet: A gigabit-per-second local area network, *IEEE Micro*, **15**(1), 29-36, 1995.

[6] Jie Chen and William Watson III, Communication Software Performance for Linux Clusters with Mesh Connections, *Proceedings of the 7th Joint Conference on Information Sciences*, 381-384, 2003.

[7] G. Chiola and G. Ciaccio, GAMMA: a Low-cost Network of Workstations Based on Active Messages, *Proceedings of 5th EUROMICRO workshop on Parallel and Distributed Processing*, 1997.

[8] C. Clos, A Study of Non-Blocking Switching Networks, *Bell System Technical Journal*, 406-424, 1953.

[9] D. Culler, L. Liu, R. P. Martin, and C. Yoshikawa, LogP performance assessment of fast network interfaces, *IEEE Micro*, 1996.

[10] David E. Culler and Jaswinder Pal Singh, *Parallel Computer Architecture; A Hardware/Software Approach*, Morgan Kaufmann Publishers, 1999.

[11] J. Duato, S, Yalmanchili, and L. M. Ni, *Interconnection networks; An engineering approach*, IEEE Computer Society Press, 1997.

[12] Z. Fodor, S. D. Katz and G. Papp, Better than $1/Mflops sustained: scalable PC-based parallel computer for lattice QCD, *Computational Physics Communication*, **152**, 121, 2003.

[13] Ian Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, 1995.

[14] W. Gropp, E. Lusk, N. Doss and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, **22**(6), 789-828, 1996.

[15] J. H. Salim, R. Olsson and A. Kuznetsov, Beyond Softnet, *Proceedings of the 5th Annual Linux Showcase and Conference*, 165-172, 2001.

[16] Frank T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, 1991.

[17] Y.-D. Lyuu, On the furthest-distance-first principle for data scattering with set-up, *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing*, 633-640, 1993.

[18] R. P. Martin, A. M. Vahdat, D. E. Culler, T. E. Anderson, Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture, *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997.

[19] A. Moody, J. Fernandez, F. Petrini and D. K. Panda, Scalable NIC-based Reduction on Large-scale Clusters, *Proceedings of the ACM/IEEE conference on Supercomputing*, 2003.

[20] D. K. Panda, Global reduction in wormhole k-ary n-cube networks with multidestination exchange worms, *Proceedings of the 10th International Parallel Processing Symposium*, 652-659, 1995.

[21] Greg Regnier, *Virtual Interface Architecture*, Intel Press, 2002.

[22] Y. Saad and M. H. Schultz, Data Communication in Parallel Architectures, *Parallel Computing*, **11**, 131-150, 1989.

[23] R. Seifert, *Gigabit Ethernet: technology and applications for high speed LANs*, Addision-Wesley, 141-280, 1998.

[24] P. Shivam, P. Wyckoff and D. Panda, EMP: Zero-copy OS-bypassing NIC-driven Gigabit Ethernet Message Passing, *Proceedings of the ACM/IEEE conference on Supercomputing*, 2001.

[25] S. Sumimoto and K. Kumon, PM/Ethernet-kRMA: A High Performance Remote Memory Access Facility Using Multiple Gigabit Ethernet Cards, *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 326-333, 2003.

[26] O. Tatebe, U. Nagashima, S. Sekiguchi, H. Kitabayashi, Design and implementation of FMPL: a fast message-passing library for remote memory operations, *Proceedings of ACM/IEEE conference on Supercomputing*, 2001.

[27] T. Von Eicken, A. Basu, V. Buch and W. Vogels, U-Net: A User-Level Network Interface for Parallel and Distributed Computing, *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 40-53, 1995.

[28] K.G. Wilson, *Confinement of Quarks*, Physics Review **D10**, 2445, 1974.

[29] Myrinet Performance Measurements, http://www.myri.com/myrinet/performance/index.html

[30] Intel Pro/1000 MT Dual Port Server Adapter, http://www.intel.com/network/connectivity/products/pro1000mt_dual_server_adapter.htm

[31] The MPI standard, http://www.mpi-forum.org/

[32] M-VIA: A High Performance Modula VIA for Linux, http://www.nersc.gov/research /FTG/via

[33] MVICH: MPI for Virtual Interface Architecture, http://www.nersc.gov/research/ftg/mvich /index.html

[34] LQCD Message Passing, http://www.lqcd.org/qmp/