USENIX Association

Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta

Atlanta, Georgia, USA
October 10–14, 2000

Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network. *

Hong Ong[‡] and Paul A. Farrell[†]
Department of Mathematics and Computer Science,
Kent State University, Kent, OH 44242.
[‡] hong@mcs.kent.edu [†] farrell@mcs.kent.edu

Abstract

We evaluate and compare the performance of LAM, MPICH, and MVICH on a Linux cluster connected by a Gigabit Ethernet network. Performance statistics are collected using NetPIPE which show the behavior of LAM/MPI and MPICH over a gigabit network. Since LAM and MPICH use the TCP/IP socket interface for communicating messages, it is critical to have high TCP/IP performance. Despite many efforts to improve TCP/IP performance, the performance graphs presented here indicate that the overhead incurred in protocol stack processing is still high. Recent developments such as the VIA-based MPI implementation MVICH can improve communication throughput and thus give the promise of enabling distributed applications to improve performance.

1 Introduction

Due to the increase in network hardware speed and the availability of low cost high performance workstations, cluster computing has become increasingly popular. Many research institutes, universities, and industrial sites around the world have started to purchase/build low cost clusters, such as Linux Beowulf-class clusters, for their parallel processing needs at a fraction of the price of mainframes or supercomputers.

On these cluster systems, parallel processing is usually accomplished through parallel programming libraries such as MPI, PVM [Geist], and BSP [Jonathan]. These environments provide well-defined portable mechanisms for which concurrent applications can be developed easily. In particular, MPI has been widely accepted in the scientific parallel computing area. The use of MPI has broadened over time as well. Two of the most extensively used MPI implementations are MPICH [Gropp, Gropp2] from Mississippi State University and Argonne National Laboratory and LAM [LSC] originally from Ohio Supercomputing Center. LAM is now being maintained by the University of Notre Dame. The modular design taken by MPICH and LAM has allowed research organizations and commercial vendors to port the software to a great variety of multiprocessor and multicomputer platforms and distributed environments.

Naturally, there has been great interest in the performance of LAM and MPICH for enabling high-performance computing in clusters. Large scale distributed applications using MPI (either LAM or MPICH) as communication transport on a cluster of computers impose heavy demands on communication networks. Gigabit Ethernet technology, among others high-speed networks, can in principle provide the required bandwidth to meet these demands. Moreover it also holds the promise of considerable price reductions, possibly even to commodity levels, as Gigabit over copper devices become more available and use increases. However, it has also shifted the communication bottleneck from network media to protocol processing. Since LAM and MPICH use TCP/UDP socket interfaces to communicate messages between nodes, there have been great efforts in reducing the overhead incurred in processing the TCP/IP stacks. However, the efforts have yielded only moderate improvement. Since then, many systems such as U-Net [Welsh], BIP [Geoffray], and Ac-

[*] This work was supported in part by NSF CDA 9617541, NSF ASC 9720221, by the OBR Investment Fund Ohio Communication and Computing ATM Research Network (OCARnet), and through the Ohio Board of Regents Computer Science Enhancement Initiative

tive Message [Martin] have been proposed to provide low latency and high bandwidth message-passing between clusters of workstations and I/O devices that are connected by a network. More recently, the Virtual Interface Architecture (VIA) [Compaq] has been developed to standardize these ideas. VIA defines mechanisms that will bypass layers of protocol stacks and avoid intermediate copies of data during sending and receiving messages. Elimination of this overhead not only enables significant communication performance increases but will also result in a significant decrease in processor utilization by the communication subsystem. Since the introduction of VIA, there have been several software and hardware implementations of VIA. Berkeley VIA [Buonadonna], Giganet VIA [Speight], M-VIA [MVIA], and FirmVIA [Banikazemi] are among these implementations. This has also led to the recent development of VIA-based MPI communications libraries, noticeably MVICH [MVICH].

The rest of this paper is organized as follows: In Section 2.1, we briefly overview the VIA architecture. In Section 2.2, we give a brief description of Gigabit Ethernet technology. The testing environment is given in Section 3. In Section 4, we present performance results for TCP/IP, LAM and MPICH. Preliminary performance results using VIA and MVICH on a Gigabit Ethernet network will also be presented. Finally, conclusions and future work are presented in Section 5.

2 VIA and Gigabit Ethernet

2.1 VIA Overview

The VIA [INTEL, INTEL2] interface model was developed based on the observation by researchers that the most significant overhead was the time required to communicate between processor, memory, and I/O subsystems that are directly connected to a network. In particular, communication time is not scaling to each individual component of the whole system and can possibly increase exponentially in a cluster of workstations.

This communication overhead is caused by the time accumulated when messages move through different layers of the Internet protocol suite of TCP/UDP/IP in the operating system. In the past, the overhead of end-to-end Internet protocols did not significantly contribute to the poor network performance since the latency equation was primarily dominated by the underlying network links. However, recent improvements in network technology and processor speeds has made the overhead of the Internet protocol stacks the dominant factor in the latency equation.

The VIA specification defines mechanisms to avoid this communication bottleneck by eliminating the intermediate copies of data. This effectively reduces latency and lowers the impact on bandwidth. The mechanisms also enable a process to disable interrupts under heavy workloads and enable interrupts only on wait-for-completion. This indirectly avoids context switch overhead since the mechanisms do not need to switch to the protocol stacks or to another process.

The VIA specification only requires control and setup to go through the OS kernel. Users (also known as VI Consumers) can transfer their data to/from network interfaces directly without operating system intervention via a pair of send and receive work queues. And, a process can own multiple work queues at any given time.

VIA is based on a standard software interface and a hardware interface model. The separation of hardware interface and software interface makes VI highly portable between computing platforms and network interface cards (NICs). The software interface is composed of the VI Provider library (VIPL) and the VI kernel agent. The hardware interface is the VI NIC which is media dependent. By providing a standard software interface to the network, VIA can achieve the network performance needed by communication intensive applications.

VIA supports send/receive and remote direct memory access (RDMA) read/write types of data movements. These operations describe the gather/scatter memory locations to the connected VI. To initiate these operations, a registered descriptor should be placed on the VI work queue. The current revision of the VIA specification defines the semantics of a DMA Read operation but does not require that the network interface support it.

The VI kernel agent provides synchronization by providing the scheduling semantics for blocking calls. As a privileged entity, it controls hardware interrupts from the VI architecture on a global, and per VI basis. The VI kernel agent also supports buffer registration and de-registration. The registration of buffers allows the enforcement of protection across process boundaries via page ownership. Privileged kernel processing is required to perform virtual-to-physical address translation and to wire the associated pages into memory.

2.2 Gigabit Ethernet Technology

Gigabit Ethernet [GEA], also known as the IEEE Standard 802.3z, is the latest Ethernet technology. Like Ethernet, Gigabit Ethernet is a media access control (MAC) and physical-layer (PHY) technology. It offers one gigabit per second (1 Gbps) raw bandwidth which is 10 times faster than fast Ethernet and 100 times the speed of regular Ethernet. In order to achieve 1 Gbps, Gigabit Ethernet uses a modified version of the ANSI X3T11 Fibre Channel standard physical layer (FC-0). To remain backward compatible with existing Ethernet technologies, Gigabit Ethernet uses the same IEEE 802.3 Ethernet frame format, and a compatible full or half duplex carrier sense multiple access/ collision detection (CSMA/CD) scheme scaled to gigabit speeds.

Like its predecessor, Gigabit Ethernet operates in either half-duplex or full-duplex mode. In full-duplex mode, frames travel in both directions simultaneously over two channels on the same connection for an aggregate bandwidth of twice that of half-duplex mode. Full duplex networks are very efficient since data can be sent and received simultaneously. However, full-duplex transmission can be used for point-to-point connections only. Since full-duplex connections cannot be shared, collisions are eliminated. This setup eliminates most of the need for the CSMA/CD access control mechanism because there is no need to determine whether the connection is already being used.

When Gigabit Ethernet operates in full duplex mode, it uses buffers to store incoming and outgoing data frames until the MAC layer has time to pass them higher up the legacy protocol stacks. During heavy traffic transmissions, the buffers may fill up with data faster than the MAC layer can process them. When this occurs, the MAC layer prevents the upper layers from sending until the buffer has room to store more frames; otherwise, frames would be lost due to insufficient buffer space.

In the event that the receive buffers approach their maximum capacity, a high water mark interrupts the MAC control of the receiving node and sends a signal to the sending node instructing it to halt packet transmission for a specified period of time until the buffer can catch up. The sending node stops packet transmission until the time interval is past or until it receives a new packet from the receiving node with a time interval of zero. It then resumes packet transmission. The high water mark ensures that enough buffer capacity remains to give the MAC time to inform the other devices to shut down the flow of data before the buffer capacity overflows. Similarly, there is a low water mark to notify the MAC control when there is enough open capacity in the buffer to restart the flow of incoming data.

Full-duplex transmission can be deployed between ports on two switches, a workstation and a switch port, or between two workstations. Full-duplex connections cannot be used for shared-port connections, such as a repeater or hub port that connects multiple workstations. Gigabit Ethernet is most effective when running in the full-duplex, point-to-point mode where full bandwidth is dedicated between the two end-nodes. Full-duplex operation is ideal for backbones and high-speed server or router links.

For half-duplex operation, Gigabit Ethernet will use the enhanced CSMA/CD access method. With CSMA/CD, a channel can only transmit or receive at one time. A collision results when a frame sent from one end of the network collides with another frame. Timing becomes critical if and when a collision occurs. If a collision occurs during the transmission of a frame, the MAC layer will stop transmitting and retransmit the frame when the transmission medium is clear. If the collision occurs after a packet has been sent, then the packet is lost since the MAC layer has already discarded the frame and started to prepare for the next frame for transmission. In all cases, the rest of the network must wait for the collision to dissipate before any other devices can transmit.

In half-duplex mode, Gigabit Ethernet's performance is degraded. This is because Gigabit Ethernet uses the CSMA/CD protocol which is sensitive to frame length. The standard slot time for Ethernet frames is not long enough to run a 200-meter cable when passing 64-byte frames at gigabit speed. In order to accommodate the timing problems experienced with CSMA/CD when scaling half-duplex

Ethernet to gigabit speed, slot time has been extended to guarantee at least a 512-byte slot time using a technique called *carrier extension*. The frame size is not changed; only the timing is extended.

Half-duplex operation is intended for shared multi-station LANs, where two or more end nodes share a single port. Most switches enable users to select half-duplex or full-duplex operation on a port-by-port basis, allowing users to migrate from shared links to point-to-point, full-duplex links when they are ready.

Gigabit Ethernet will eventually operate over a variety of cabling types. Initially, the Gigabit Ethernet specification supports multi-mode and single-mode optical fiber, and short haul copper cabling. Fiber is ideal for connectivity between switches and between a switch and high-speed server because it can be extended to greater length than copper before signal attenuation becomes unacceptable and it is also more reliable than copper. In June 1999, the Gigabit Ethernet standard was extended to incorporate category 5 unshielded twisted-pair (UTP) copper media. The first switches and network NICs using category 5 UTP became available at the end of 1999.

3 Testing Environment

The testing environment for collecting the performance results consists of two Pentium III PCs running at 450MHz with a 100MHz memory bus, and 256MB of PC100 SD-RAM. The PCs are connected back to back via a Gigabit Ethernet NIC installed in the 32bit/33MHz PCI slot. The PCs are also connected together through a SVEC 5 port 10/100 Mbps autosensing/autoswitching hub via 3Com PCI 10/100Mbps Ethernet Cards (3c905B). Three different types of Gigabit Ethernet NICs, the Packet Engine GNIC-II, the Alteon ACEnic, and the SysKonnect SK-NET were tested. The device drivers used are Hamachi v0.07 for GNIC-II, Acenic v0.45 for ACEnic, and Sk98lin v3.01 for SK-NET. In addition, the cluster is isolated from other network traffic to ensure the accuracy of the tests. The cluster is running Red Hat 6.1 Linux distribution with kernel version 2.2.12. In addition, M-VIA v0.01, LAM v6.3, MPICH v1.1.2, and MVICH v0.02 were installed. M-VIA is an implementation of VIA for Linux, which currently supports fast Ethernet cards with DEC Tulip chipset or the Intel i8255x chipset, and the Packet Engines GNIC-I and GNIC-II Gigabit Ethernet cards. The device driver used by M-VIA is a modified version of Donald Becker's Hamachi v0.07. MVICH is a MPICH based MPI implementation for M-VIA. M-VIA and MVICH are being developed as part of the NERSC PC Cluster Project. NetPIPE-2.3 [Snell] was used to test the TCP/IP, LAM, MPICH and MVICH performance. For M-VIA tests, we used the `vnettest.c`, a ping-pong-like C program, distributed with the software.

4 Performance Evaluation

In this section, we present and evaluate TCP/IP, M-VIA, LAM, MPICH, and MVICH point-to-point communication performance using the various Gigabit Ethernet NICs mentioned earlier. The throughput graph is plotted using throughput versus transfer block size. Throughput is reported in megabits per second (Mbps) and block size is reported in bytes since they are the common measurements used among vendors. The throughput graph clearly shows the throughput for each transfer block size and the maximum attainable throughput. The throughput graph combined with application specific requirements will help programmers to decide what block size to use for transmission in order to maximize the achievable bandwidth. The signature (latency) graph is plotted using throughput per second versus total transfer time elapsed in the test. In `NetPIPE`, the latency is determined from the signature graph. The network latency is represented by the time to transfer 1 byte and thus coincides with the time of the first data point on the graph.

4.1 Comparing TCP/IP and M-VIA Performance

Since TCP was originally engineered to provide a general transport protocol, it is not by default optimized for streams of data coming in and out of the system at high transmission rates (e.g 1Gbps). In [Farrell], it is shown that communication performance is affected by a number of factors and indicated that one can tune certain network parameters to achieve high TCP/IP performance especially for a high speed network such as a Gigabit Ethernet network. We have taken care to tune the TCP pa-

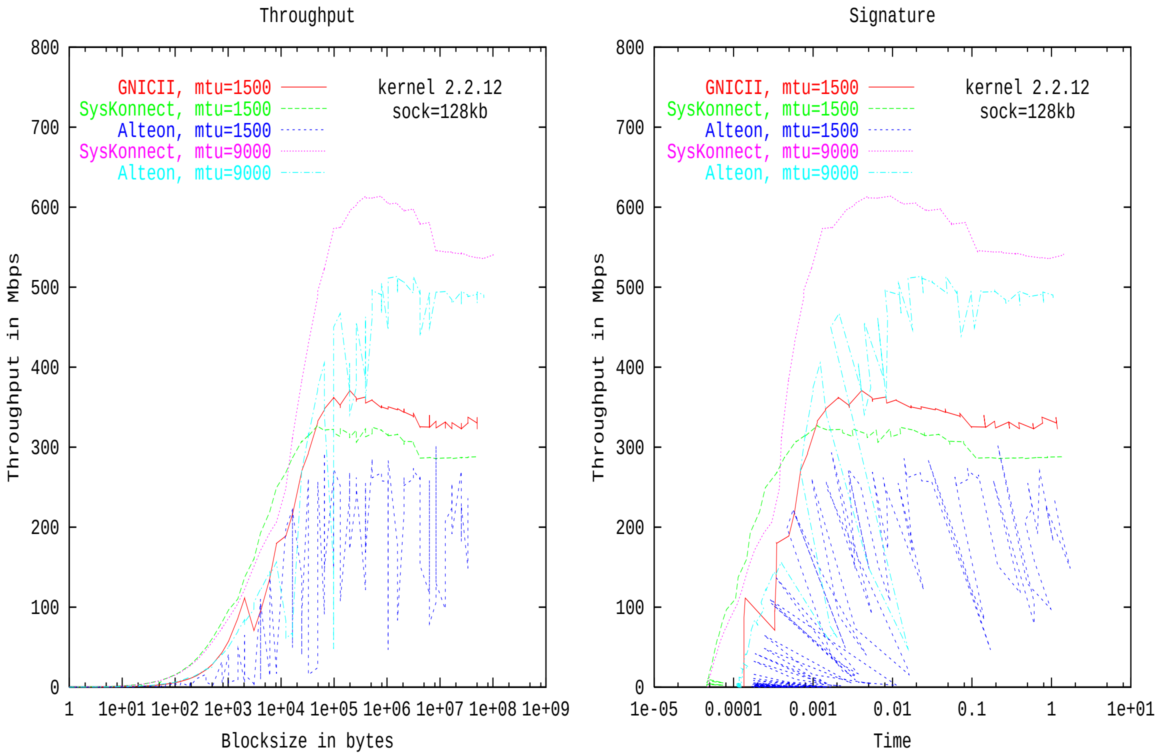

Figure 1: GNIC-II, Alteon, & SysKonnect: TCP Throughput and Latency

rameters according to RFC 1323 TCP/IP Extension for High Performance [RFC1323] in order to achieve high speed TCP/IP communication. We have also set the window size to 128KB rather than the default 64KB in the Linux 2.2.12 kernel.

Figure 1 shows the TCP/IP throughput and latency for various Gigabit Ethernet NICs. Since ACEnic and SK-NET support frame sizes larger than the default of 1500 bytes, we tested them with different MTU sizes. In the figure, we present the TCP/IP performance with MTU of 1500 bytes for all Gigabit Ethernet NICs, and also with MTU equals to 9000 for ACEnic and SK-NET which achieves the highest peak throughput.

One obvious observation from the figures is there are many severe dropouts in ACEnic TCP/IP performance. The reason for these dropouts is supposedly due to the ACEnic device driver. For instance, using ACEnic device driver v0.32, we obtained maximum TCP/IP throughput of 356Mbps using MTU equals to 1500 and 468 Mbps using an MTU of 6500 bytes as opposed to an MTU of 9000 bytes. Furthermore, the latency of ACEnic driver v0.32 is approximately 40% less than the latency of ACEnic device driver v0.45. In addition, with MTU of 1500 bytes and the ACEnic device driver v0.32, the TCP throughput performance is better than that presented here. However, the TCP/IP performance of the ACEnic using device driver v0.45 with large MTU has improved substantially. In general, the overall TCP behavior for both ACEnic device drivers v0.32 and v0.45 have not been improved since v0.28, i.e., the performance graphs have many severe dropouts. In [Farrell], the ACEnic device driver v0.28 running on the Linux 2.2.1 kernel has a smoother performance curve and achieved its maximum throughput of 470 Mbps, using an MTU of 9000.

For MTU of 1500, the maximum attainable throughput is approximately 371 Mbps, 301 Mbps, and 331 Mbps for GNIC-II, ACEnic, and SK-NET respectively. And, the latency is approximately 137 μsecs, 182 μsecs, and 45 μsecs for GNIC-II, ACEnic, and SK-NET respectively. With the lowest latency, SK-NET is able to perform much better than the ACEnic and than the GNIC-II for message sizes up

MTU	Alteon		SysKonnect	
byte	sock=64KB	sock=128KB	sock=64KB	sock=128KB
1500	294.624	301.765	326.488	331.028
4500	402.537	441.932	506.391	509.832
8500	362.068	495.932	495.813	561.456
9000	372.955	513.576	512.490	613.219

Table 1: TCP/IP Performance with Large Socket Buffer and MTU

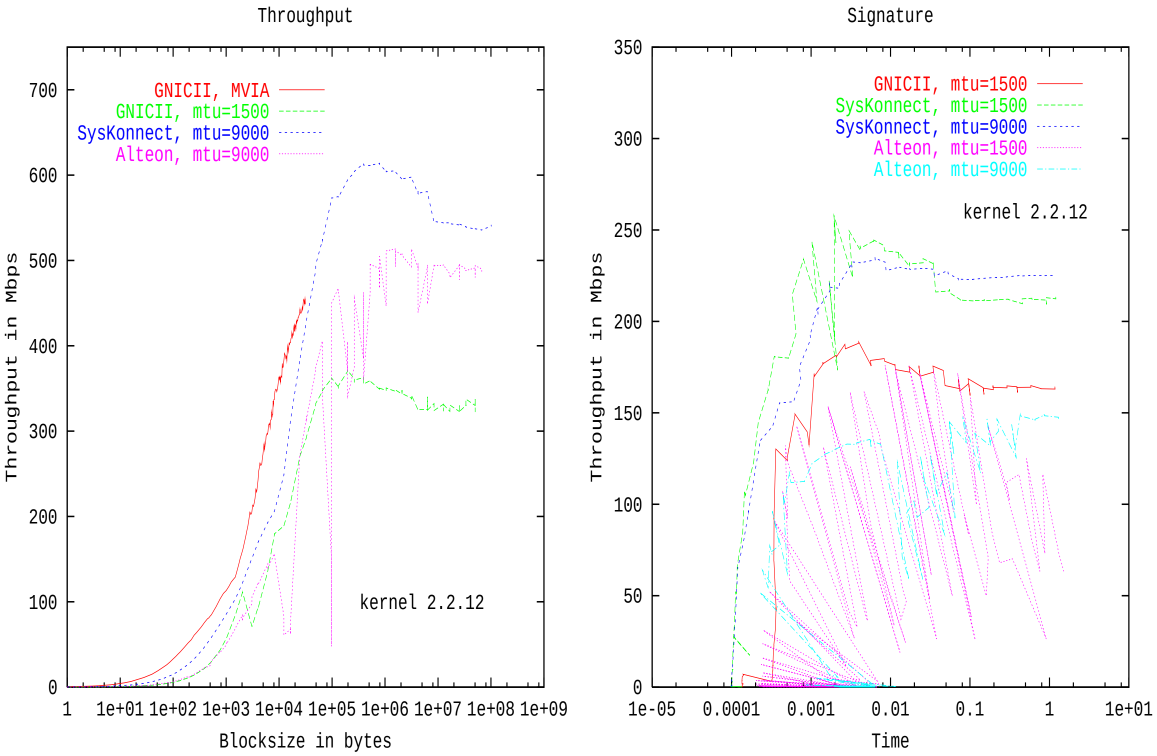

Figure 2: GNIC-II: M-VIA Throughput and Latency

to 49KB. For example, for message size of 16KB, SK-NET throughput is approximately 32% more than the GNIC-II and 82% more than the ACEnic. However, for message sizes greater than 49KB, SK-NET reaches its maximum of 331 Mbps.

Tests on networks based on FDDI, ATM [Farrell2] and Fibre Channel have shown that high speed networks perform better when the MTU is larger than 1500 bytes. Similarly, we expect Gigabit Ethernet would also perform better with an MTU greater than 1500 bytes. From Figure 1, we see that ACEnic maximum attainable throughput increases approximately 70% reaching 513 Mbps when the MTU is set to 9000; And, for SK-NET, the maximum attainable throughput has also increased to approximately 613 Mbps. The latency of ACEnic has decreased to 121 μsecs; and, the SK-NET has increased slightly to 46 μsecs. In order to benefit from the larger MTU, one must also use a larger socket buffer size rather than the default socket buffer size of 64KB. Table 1 shows this effect for various sizes of MTU and socket buffer sizes of 64KB and 128KB.

Figure 2 shows the throughput and latency of M-VIA on the GNIC-II compared with the best attainable performance for each card using TCP. The maximum attainable throughput for M-VIA remains yet to be determined. This is due to the fact that `vnettest.c` stops when message size reaches 32KB which is the maximum data buffer size supported by the M-VIA implementation. For message sizes around 30KB, the throughput reaches approximately 448 Mbps with latency of only 16 μ secs. Thus, the throughput is approximately 53%, 42% and 4% more than GNIC-II, ACEnic, and SK-NET respectively.

The VIA specification only requires VIA developers to support the minimum data buffer of 32KB. However, developers may choose to support data buffer sizes greater than 32KB. In this case, developers must provide a mechanism for the VI consumer to determine the data buffer size. Thus, we expect a larger data buffer will give higher throughput as message sizes continue to increase. On the other hand allocating larger data buffers may result in memory wastage.

4.2 Comparing LAM, MPICH, and MVICH Performance

In this section, we present and compare the performance of LAM, MPICH, and MVICH on a Gigabit Ethernet network. Before moving on to discuss the performance results of LAM and MPICH, it is useful to first briefly describe the data exchange protocol used in these two MPI implementations. The choices taken in implementing the protocol can influence the performance as we will see later in the performance graphs.

Generally, LAM and MPICH use a short/long message protocol for communication. However, the implementation is quite different. In LAM, a short message consisting of a header and the message data is sent to the destination node in one message. And, a long message is segmented into packets with the first packet consisting of a header and possibly some message data sent to the destination node. Then, the sending node waits for an acknowledgment from the receiver node before sending the rest of the data. The receiving node sends the acknowledgment when a matching receive is posted. MPICH (P4 ADI) implements three protocols for data exchange. For short messages, it uses the eager protocol to send message data to the destination node immediately with the possibility of buffering data at the receiving node when the receiving node is not expecting the data. For long messages, two protocols are implemented - the rendezvous protocol and the get protocol. In the rendezvous protocol, data is sent to the destination only when the receiving node requests the data. In the get protocol, data is read directly by the receiver. This choice requires a method to directly transfer data from one process's memory to another such as exists on parallel machines.

All the LAM tests are conducted using the LAM client to client (C2C) protocol which bypasses the LAM daemon. In LAM and MPICH, the maximum length of a short message can be configured at compile time by setting the appropriate constant. We configured the LAM short/long messages switchover point to 128KB instead of the default 64KB. For MPICH, we used all the default settings. Figure 3 shows LAM throughput and latency graphs. And, Figure 4 shows MPICH throughput and latency graphs.

For LAM using MTU size of 1500 bytes, the maximum attainable throughput is about 216 Mbps, 188 Mbps, and 210 Mbps with latency of 140 μsecs, 194 μsecs, and 66 μsecs for the GNIC-II, ACEnic, and SK-NET respectively. For MPICH using MTU size of 1500, the maximum attainable throughput is about 188 Mbps, 176 Mbps, and 249 Mbps with latency of 142 μsecs, 239 μsecs and 99 μsecs for the GNIC-II, ACEnic, and SK-NET respectively. Since LAM and MPICH are layered above TCP/IP stacks, one would expect only a small decrease in performance. However, the amount of performance degradation in LAM and MPICH as compared to the TCP/IP performance is considerable. For LAM, the performance drop of approximately 42%, 38% and 41% for GNIC-II, ACEnic, and SK-NET respectively. And, the performance drop for MPICH is approximately 49%, 42%, and 25% for GNIC-II, ACEnic, and SK-NET respectively.

Changing MTU to a larger size improves LAM performance somewhat. For LAM, the maximum attainable throughput is increased by approximately 42% for SK-NET and by approximately 36% for the ACEnic with MTU of 9000 respectively. However, changing MTU to a bigger size decreases MPICH performance. For MPICH, the maximum attainable throughput drops by approximately 7% for an SK-NET and by approximately 15% for an ACEnic with MTU of 9000.

In all cases, increasing the size of the MTU also increases the latency slightly except in the case of the test on MPICH using the ACEnic. In particular, the latency of LAM is approximately 69 μsecs for SK-NET and 141 μsecs for ACEnic. And, the latency of MPICH is approximately 100 μsecs for SK-NET and 2330 μsecs for ACEnic.

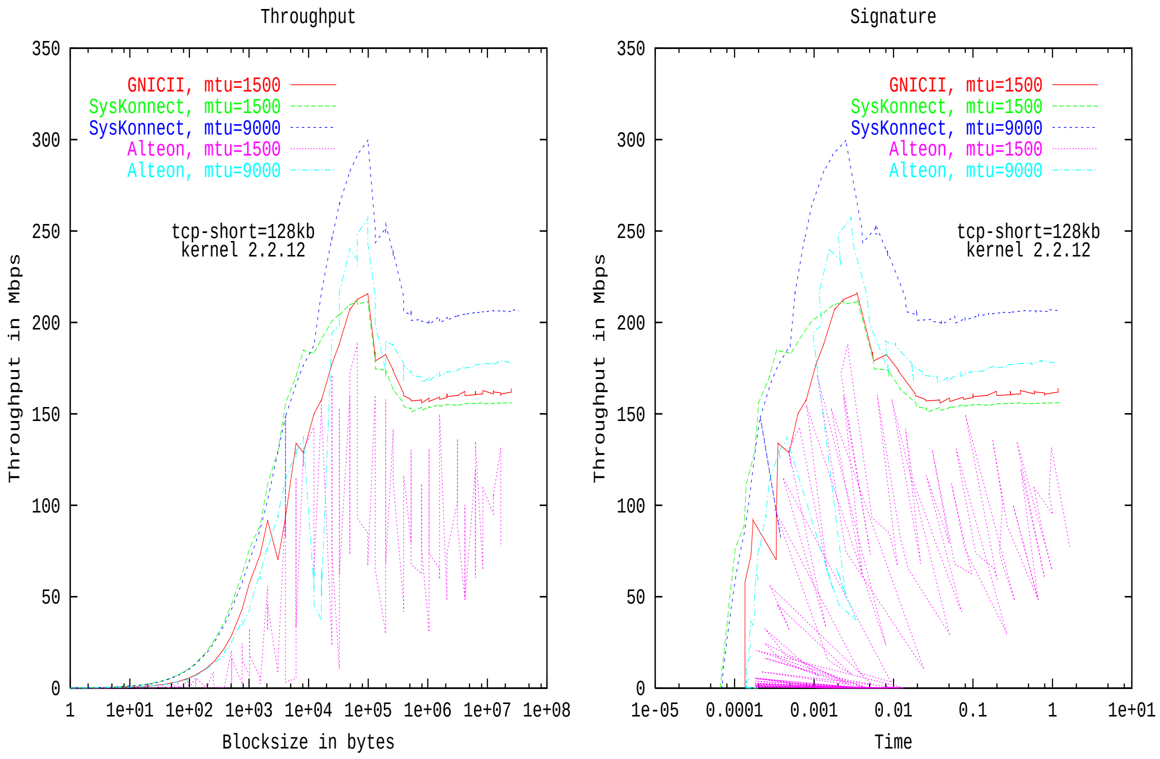

Figure 3: GNIC-II, Alteon, & SysKonnect: LAM Throughput and Latency

Again, we see that there are many severe dropouts for both LAM and MPICH using the ACEnic card.

Several things can be said regarding these performance results.

- As noted from Table 1, TCP/IP performs better on a Gigabit Ethernet network for large MTU and socket buffer size. During initialization, LAM sets send and receive socket buffers, SOCK_SNDBUF and SOCK_RCVBUF, to a size equal to the switch-over point plus the size of the C2C envelope data structure. This explains why, when we made the MTU greater than 1500 bytes, LAM performance improved. However, MPICH initializes SOCK_SNDBUF and SOCK_RCVBUF size equal to 4096 bytes. Hence, a larger MTU does not help to improve MPICH performance much.

- In both LAM and MPICH, a drop in performance, more noticeably for LAM at 128KB, is caused by the switch from the short to long message protocol described above. In particular, we specified that messages of 128KB or longer be treated as long messages in LAM. For MPICH, the default switch-over point to long message handling is 128000 bytes.

From the figures, it is evident that an MPI implementation layered on top of a TCP/IP protocol depends highly on the underlying TCP/IP performance.

Figure 5 shows MVICH performance. MVICH attains a maximum throughput of 280 Mbps with latency of only 26 μsecs for message sizes as low as 32KB. Again, we were unable to run message sizes greater than 32KB. From the figure, it is evident that, as hoped, MVICH performance is much superior to that of LAM or MPICH using TCP/UDP as communication transport.

5 Conclusion

In this paper, we have given an overview of VIA and Gigabit Ethernet technology. The performance of TCP, M-VIA, LAM, MPICH, and MVICH using three types of Gigabit Ethernet NICs, the GNIC-

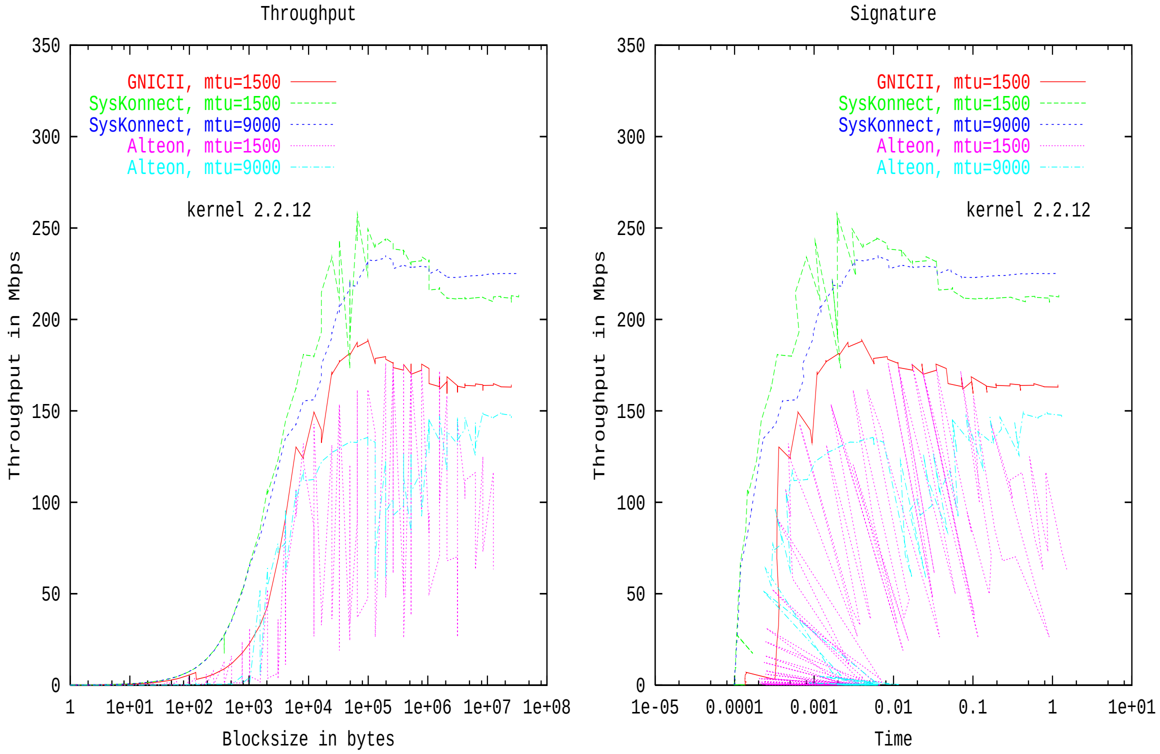

Figure 4: GNIC-II, Alteon & SysKonnect: MPICH Throughput and Latency

II, the ACEnic, and the SK-NET on a PC cluster were presented. We evaluated and compared the performance of TCP, LAM, MPICH, and MVICH. In order to achieve high TCP/IP performance on a high speed network, we indicated that one has to tune certain network parameters such as RFC1323, socket buffer size, and MTU. We attempted to explain the poor performance of LAM and MPICH and show MVICH as a promising communication library for MPI based applications running on a high speed network. We remark that we tested M-VIA v0.01 and MVICH v0.02 which are very early implementations and the performance is likely to improve further with further development.

Further investigation of the effects of tuning the MPICH implementation is also warranted. In addition, ANL is currently in the process of modifying MPICH to provide a device independent layer which will permit easy addition of device driver modules for various networks. This may also lead to further improvements in performance. SysKonnect is currently writing a VIA driver for the SK-NET card. It will be of some interest to compare the throughput and, in particular, latency achievable with this implementation.

References

[Banikazemi] M. Banikazemi, V. Moorthy, L. Hereger, D. K. panda, and B. Abali. *Efficient Virtual Interface Architecture Support for IBM SP Switch-Connected NT Clusters.* International Parallel and Distributed Processing Symposium. (2000)

[Buonadonna] P. Buonadonna, J. Coates, S. Low, and D. E. Culler., *Millennium Sort: A Cluster-Based Application for Windows NT using DCOM, River Primitives and the Virtual Interface Architecture.* In Proc. of 3rd USENIX Windows NT Symposium. (1999)

[Compaq] Compaq Computer Corp., Intel Corporation, Microsoft Corporation, *Virtual Interface Architecture Specification version 1.0.* `http://www.viarch.org`

[Farrell] Paul Farrell and Hong Ong, *Communication Performance over a Gigabit Ethernet Network*, IEEE Proc. of 19th IPCCC (2000).

[Farrell2] Paul Farrell, Hong Ong, and Arden Ruttan, *Modeling liquid crystal structures using*

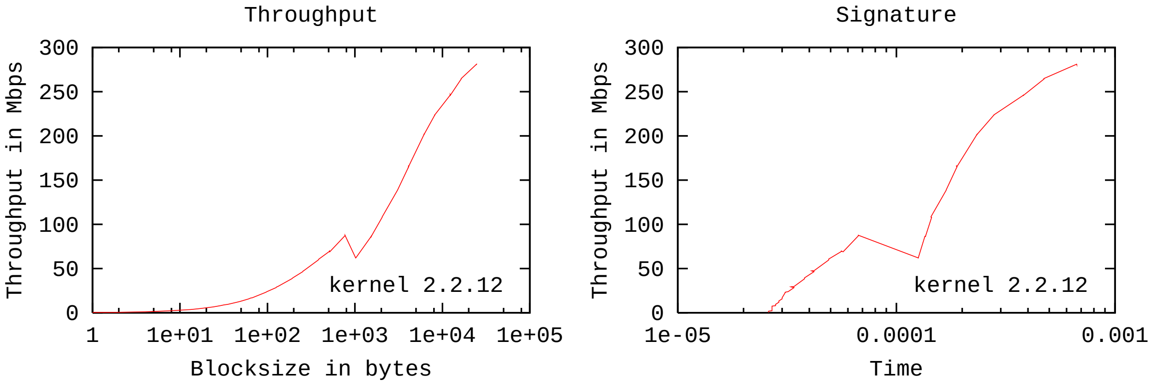

Figure 5: GNIC-II: MVICH Throughput and Latency

MPI on a workstation cluster, to appear in Proc. of MWPP 1999.

[GEA] Gigabit Ethernet Alliance, *Gigabit Ethernet Overview.* (1997) `http://www.gigabit-ethernet.org/`

[Geist] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R .Manchek, V. Sunderam. *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Network Parallel Computing.* MIT Press (1994).

[Geoffray] P. Geoffray, L. Lefevre, C. D. Pham, L. Prylli, O. Reymann, B. Tourancheau, and R. Westrelin. *High-speed LANs: New environments for parallel and distributed applications.* In EuroPar'99, France. Springer-Verlag (1999).

[Gropp] W. Gropp and E. Lusk and N. Doss and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parall. Comp. 22 (1996).

[Gropp2] William D. Gropp and Ewing Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI*, Argonne National Laboratory (1996), ANL-96/6.

[Jonathan] Jonathan M. D. Hill, Stephen R. Donaldson, and David B. Skillicorn, *Portability of performance with the BSPlib communications library*, Massively Parallel Programming Models Workshop (1997).

[INTEL] Intel Corporation, *Virtual Interface (VI) Architecture: Defining the Path to Low Cost High Performance Scalable Clusters.* (1997)

[INTEL2] Intel Corporation, *Intel Virtual Interface (VI) Architecture Developer's Guide revision 1.0.* (1998).

[LSC] Laboratory for Scientific Computing at the University of Notre Dame. `http://www.mpi.nd.edu/lam`

[Martin] Richard P. Martin, Amin M. Vahdat, David E. Culler, Thomas E. Anderson. *Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture.* ISCA 24 (1997).

[MVIA] M-VIA: A High Performance Modular VIA for Linux. `http://` `www.nersc.gov/research/FTG/via/`

[MVICH] MPI for Virtual Interface Architecture. `http://` `www.nersc.gov/research/FTG/mvich/`

[RFC1323] Jacobson, Braden, & Borman, *TCP Extensions for High Performance*, RFC 1323 (1992).

[Speight] E. Speight, H. Abdel-Shafi, and J. K. Bennett. *Realizing the Performance Potential of the Virtual Interface Architecture.* In Proc. of the International Conf. on Supercomputing. (1999)

[Snell] Q. O. Snell, A. R. Mikler, and J. L Gustafson, *NetPIPE: Network Protocol Independent Performance Evaluator.*, Ames Laboratory/ Scalable Computing Lab, Iowa State. (1997)

[Welsh] Matt Welsh, Anindya Basu, and Thorsten von Eicken. *Incorporating Memory Management into User-Level Network Interfaces.* Proc. of Hot Interconnects V, Stanford (1997).