Electronic Transactions on Numerical Analysis.
Volume 15, pp. 211-224, 2003.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

FACTORS INVOLVED IN THE PERFORMANCE OF COMPUTATIONS ON BEOWULF CLUSTERS[*]

PAUL A. FARRELL [†] AND HONG ONG [‡]

Abstract. Beowulf (PC) clusters represent ve platform for large scale scientific computations. In this paper, we discuss the effects of some possible configuration, hardware, and software choices on the communications latency and throughput attainable, and the consequent impact on scalability and performance of codes. We compare performance currently attainable using Gigabit Ethernet with that of Fast Ethernet. We discuss the effects of various versions of the Linux kernel, and approaches to tuning it to improve TCP/IP performance.

We comment on the relative performance of LAM, MPICH, and MVICH on a Linux cluster connected by a Gigabit Ethernet network. Since LAM and MPICH use the TCP/IP socket interface for communicating messages, it is critical to have high TCP/IP performance for these to give satisfactory results. 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. We discuss the Virtual Interface Architecture (VIA) which is intended to provide low latency, high bandwidth message-passing between user processes. 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. Finally we present some examples of how these various choices can impact the performance of an example multigrid code.

Key words. cluster computation, gigabit ethernet, TCP/IP performance, virtual interface architecture (VIA), MPI, MPICH, LAM, MVICH, NAS multigrid benchmarks

AMS subject classifications. 65Y05, 68M14, 68M12, 68M20, 68N99, 65Y20

1. Introduction.

Beowulf (PC) clusters represent a cost-effective platform for large scale scientific computations. In this paper, we discuss the process of building such a cluster based on commodity hardware, such as PCs and general purpose network equipment. By general purpose network equipment, we mean network interface cards (NICs) and switches which have been developed for use in general local area networks (LANs) as opposed to those which are designed specifically for use in clusters or parallel machines, such as Myrinet or Giganet, although some of the comments also apply to the latter. We consider the effects of some possible configuration, hardware, and software choices on the computational performance, communications latency and throughput attainable. In particular, we discuss the effects of various versions of the Linux kernel, and approaches to tuning it to improve TCP/IP performance.

We also consider the performance of common implementations of the Message Passing Interface (MPI) software, which is commonly used to achieve distributed (message passing) parallelism in computational science. We comment on the relative performance of LAM[15], MPICH[8, 9], and MVICH[17] on a Linux cluster connected by a Gigabit Ethernet network[6]. Since LAM and MPICH use the TCP/IP socket interface for communicating messages, it is critical to have high TCP/IP performance for these to give satisfactory results. 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. We discuss the Virtual Interface Architecture (VIA[4]) which is intended to provide low latency, high bandwidth message-passing between user processes. Emerging developments such as the VIA-based MPI implementation MVICH can improve latency and hence communica-

[*] Received June 27, 2001. Accepted for publication October 19, 2001. Recommended by Ulrich Ruede.

[†] Department of Computer Science, Kent State University, Kent, Ohio 44242, U.S.A.

[‡] Department of Computer Science, Kent State University, Kent, Ohio 44242, U.S.A.

This work was supported in part by NSF CDA 9617541, NSF ASC 9720221, NSF ITR 0081324, by the OBR Investment Fund Ohio Communication and Computing ATM Research Network (OCARnet), and through the Ohio Board of Regents Computer Science Enhancement Initiative

tion throughput and thus give the promise of enabling distributed applications to achieve improved performance. Finally we present some examples of the performance achieved for the NAS[1, 2] multigrid benchmark code.

2. Testing Environment. The testing environment for collecting the performance results were those utilized in evaluating hardware and software for the creation of the Kent State Computer Science Beowulf cluster, called `fianna`. This consists of dual processor Pentium III PCs running at 450MHz with a 100 MHz bus, and 256MB of PC100 SD-RAM. The cluster operating system was the Red Hat 6.2 Linux distribution with kernel version 2.2.12. Each node has a single 10/100 Mbps Ethernet card based on the DEC Tulip chip set (3c905B) and a SysKonnect SK-NET Gigabit Ethernet card. The latter are connected by a Foundry FastIron II Plus GC Gigabit Ethernet level 2 switch using a mix of enhanced category 5 unshielded twisted pair copper (UTP), and multi-mode fiber. These were some of the first switches and network NICs using category 5 UTP and became available at the end of 1999. The 10/100 Mbps network utilizes two 3Com 3300 10/100 Mbps switches. In addition, the cluster is isolated from other network traffic to ensure the accuracy of the tests.

Three different types of Gigabit Ethernet NICs, the Packet Engine GNIC-II, the Alteon ACEnic, and the SysKonnect SK-NET, all installed in the 33MHz PCI slot, were tested. For the tests involving the Packet Engines GNIC-II and the Alteon ACEnic, and all the tests involving MTU greater than 1500, the PCs were connected back to back using a crossover cable, rather than through the Foundry switch. In the cases of the initial tests, in Sections 3 and 4, details are given about the network device drivers and Linux kernel versions used. The tests, in Sections 5 and 6, used the 2.2.12 version of the Linux kernel, running on Pentium III/450s with 256 MB of PC100 SD-RAM, and the device drivers used were Hamachi v0.07 for GNIC-II, Acenic v0.32 for ACEnic, and Sk98lin v3.01 for SK-NET respectively. In addition, M-VIA v0.01, LAM v6.3, MPICH v1.1.2, and MVICH v0.02 were installed. M-VIA[16] is an implementation of VIA for Linux, which initially supported fast Ethernet cards with the 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. The current version, M-VIA 1.2b1, supports DEC Tulip, Intel Pro/100 and 3Com "Boomerang" fast ethernet cards, and PacketEngines GNIC-I and GNIC-II, SysKonnect SK-98XX and Intel PRO/1000 gigabit ethernet cards. MVICH is a MPICH based MPI implementation, using M-VIA as a network transport layer instead of TCP/IP. M-VIA and MVICH are being developed as part of the NERSC PC Cluster Project. NetPipe-2.3 [20] 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. 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 [5], it is shown that communication performance is effected 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 parameters according to RFC 1323 TCP/IP Extension for High Performance [19] in order to achieve high speed TCP/IP communication.

3. CPU Performance. Although this is not our primary interest in this paper, we wish to indicate that many of the same issues which arise in the case of implementation of computational science codes on sequential or purpose built parallel machines also arise for PC clusters. In particular, it is important to ensure that the computation remains in core and does not use virtual memory, or significant degradation in performance will occur. This was historically the reason for explicit out-of-core implementations for large problems. One reason to use networks of workstations or clusters is to ensure that the code will run in core on each

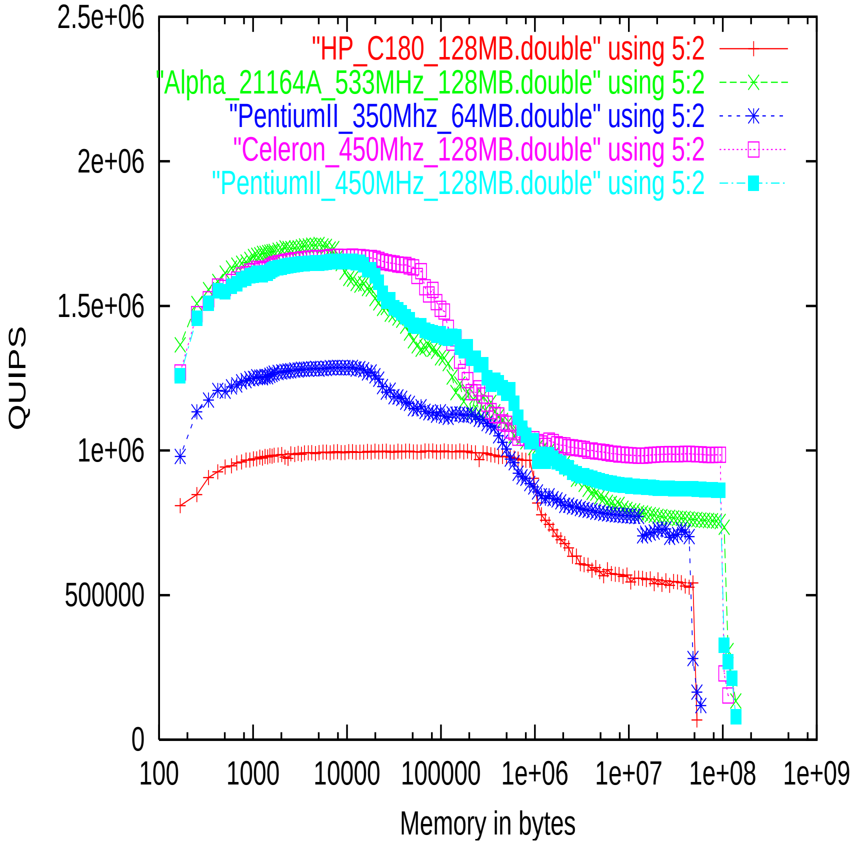

FIG. 3.1. *Hint Benchmarks for Some Processors*

machine. Even in cases where the parallelism of the algorithm is less then ideal this may lead to significant performance improvements over the case where virtual memory is used (see [21]). Also well known, particularly in multi-grid circles, is the performance enhancement which occurs if one can optimize the implementation of the algorithm to permit most of the computation to proceed in cache. This added performance has become even more important as the disparity between processor/cache speed and memory speed has grown.

To illustrate some of these issues, the performance realizable by various processors for the synthetic HINT benchmark is shown in Figure 3.1. HINT [12] or Hierarchical INTegration is a performance benchmarking tool developed at the Scalable Computing Laboratory (SCL) of the Ames Laboratory of the U.S. Department of Energy (DOE). Unlike traditional benchmarks, HINT neither fixes the size of the problem nor the calculation time and instead uses a measure called QUIPS (QUality Improvement Per Second) [10]. This enables HINT to display the speed for a given machine specification and problem size. This speed typically decreases as the calculation moves from cache access to main memory access and finally to virtual memory disk access. Such changes are easily visible with HINT generated data. Thus various points on the HINT graph correspond to various memory regimes and might be viewed as corresponding to some degree to other more conventional benchmarks such as Dhrystones, Whetstones. Linpack, SPECint, and SPECfp [11].

In Figure 3.1, it is clear that the initial plateau represents the speed while in primary cache. In some cases there is an additional plateau corresponding to secondary cache. The remaining relatively flat portion corresponds to the performance of computations using main memory and the final drop off corresponds to using virtual memory and hence accessing disk.

4. Network Performance.

In this section we address the application level performance attainable by Gigabit Ethernet using the TCP/IP protocol as implemented in the Linux socket call. First we summarize a few of the issues which arose in creating the Gigabit Ethernet standard.

4.1. Gigabit Ethernet Fundamentals.

Gigabit Ethernet [6], 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.Full duplex mode is very efficient since data can be sent and received simultaneously. However, it can only be used for point-to-point connections, such as between ports on two switches, a workstation and a switch port, or between two workstations. 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 layer, when there is enough open capacity in the buffer to restart the flow of incoming data.

Gigabit Ethernet now supports a variety of cabling types, including multi-mode and single-mode optical fiber, and enhanced category 5 unshielded twisted-pair (UTP) copper media. For half-duplex operation, Gigabit Ethernet uses the enhanced CSMA/CD access method, and in order to permit reasonable cable runs uses *carrier extension* (see [5] for more details).

Building on the success of Gigabit Ethernet as a high-performance network interconnect for local area networks, the next generation 10 Gigabit Ethernet Standard[7] is currently under development and expected to be adopted in mid-2002. It will support a data rate of 10 Gbps. Apart from the raw speed improvement, the 10 Gigabit Ethernet is significantly different in some aspects from its predecessors, for example it will only operate in full-duplex mode and only support optical fiber. A detailed discussion of 10 Gigabit Ethernet is outside the scope of

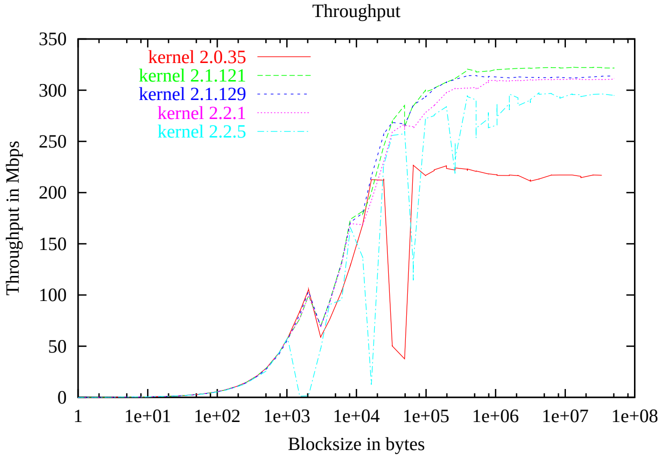

FIG. 4.1. *Kernel Progression: Throughput*

this paper. However, this upcoming technology will most probably meet the ever increasing bandwidth needs of large scale computations, such as those involving computational steering.

4.2. Linux Kernel Versions and Tuning. Although the discussions in this section are for specific Gigabit Ethernet NICs, CPU speeds, and Linux kernel versions, regardless of the specific hardware involved or the speed of the processors or network interconnects, the following discussion on network parameter tuning and driver issues are still relevant for obtaining optimum performance from Beowulf clusters.

The graphs in Figure 4.1 show the performance for the Packet Engine GNIC-II Gigabit Ethernet adaptor for different versions of Linux kernel, on a Pentium II/350 with 128MB of PC100 SD-RAM memory. In these tests, we used the Hamachi v0.07 driver and the socket size is set to 64KB. This is because the maximum socket size in kernel 2.0.35 is only 64KB and the Hamachi v0.07 was the only driver available at that time. We modified the Hamachi v0.07's descriptor length to 32 for the send list and 64 for the receive list instead the default size of 16 and 32.

In kernel 2.0.35, the maximum attainable throughput was approximately 226Mbps for message size of 191KB. However, from the figure, we also see that there is a 80% drop in performance for transfer block size ranges from 32KB to 64KB. This is due to the maximum default TCP socket buffer size of 32KB. We see that this problem is eliminated when the maximum default TCP socket buffer size is increased to 64KB in Linux kernel 2.1.x. Using kernel 2.1.x and kernel 2.2.1, the curves are free of severe drops out. The maximum attainable throughput for these graphs is approximately 310Mbps-322Mbps for transfer block size of greater than 1MB.

The sudden drop in performance (roughly 20-30Mbps) at 3KB is consistent across kernel 2.0.x, kernel 2.1.x, and kernel 2.2.1. This performance anomaly is related to the size of the Maximum Transmission Unit (MTU). This is the maximum size of a physical frame that can be sent. For an Ethernet standard compliant network the maximum this can be set to is 1500 bytes. As we will see later, this performance drop will be shifted to the right as we increase the MTU size beyond this limit.

In Figure 4.2, we see that faster processors can attain marginally higher throughput for large transfer block sizes (greater than 1MB). The maximum attainable throughput is approximately 348Mbps for Celeron and Pentium II 450MHz processors and 320Mbps for the

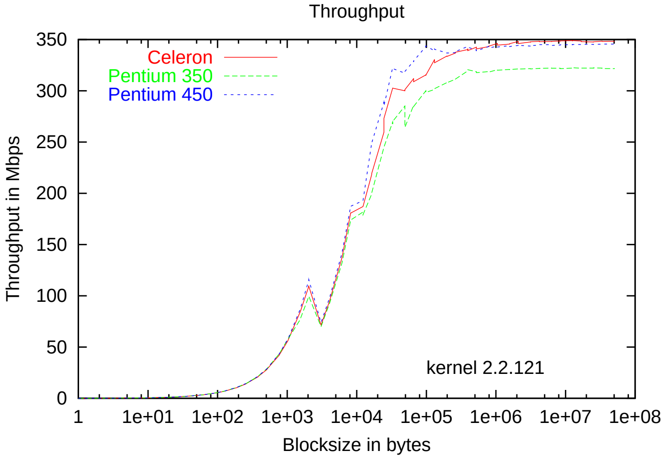

FIG. 4.2. *Processors Comparison: Throughput*

Pentium II 350MHz processor. This is largely due to the fact that faster processors can process the protocol stacks faster than the slower processors. However note that an increase in processor speed of 29% resulted in only a 9% increase in throughput. This suggests that the primary bottleneck lies elsewhere.

4.3. Socket Buffer Size and MTU Comparison. In this section, we present the throughput graphs and latency results for Gigabit Ethernet using different sizes of MTU and different socket buffer sizes. The socket buffer size, which can be set by root in Linux, determines the size of the sliding window in TCP and thus the number of packets which can be sent without an acknowledgment (ACK) being received from the receiver. Setting the socket buffer larger means that additional memory is used for buffering in the socket software implementation of TCP. It is well understood that MTU can be a limiting factor in throughput. Therefore there has been some discussion by vendors about modifying the Ethernet standard to permit MTUs larger than 1500. These are sometimes called Jumbo frames. The first vendor to provide

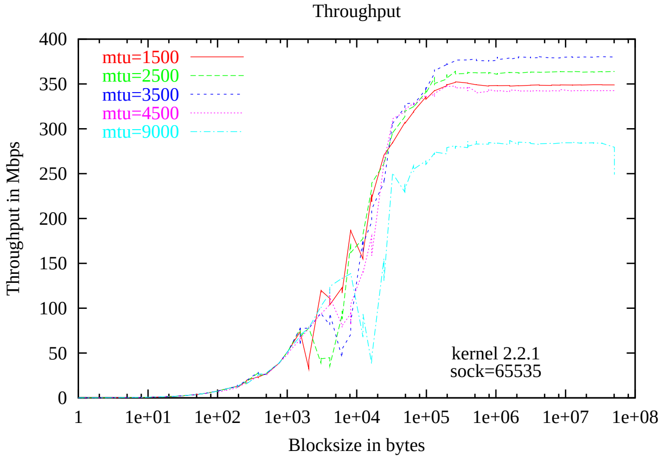

FIG. 4.3. *ACEnic Pentium: Throughput, socket=64KB*

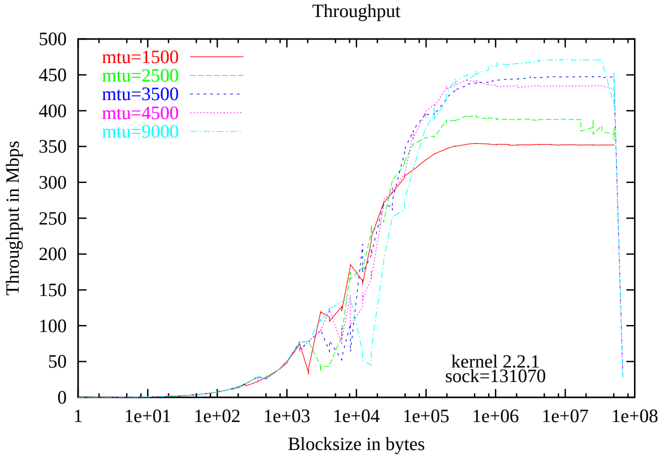

FIG. 4.4. *ACEnic Pentium: Throughput, socket=128KB*

these as a non-standard feature was Alteon in their ACEnic gigabit NIC and switches. These permitted setting the MTU to 9000 bytes. Note that to use MTU greater than 1500 both the NIC and switch must handle the larger MTU.

Figure 4.3 and Figure 4.4 show the throughput, between Pentium II/350s with 128MB of PC100 SD-RAM memory, using socket buffer sizes of 64KB and 128KB, respectively. There are a number of interesting observations from these figures. For socket buffer size of 64KB, increasing the MTU to 2500 and 3500 bytes results in increases in maximum throughput attainable. However continuing to increase the MTU to 4500 or 9000 bytes results in decreases in throughput. Thus of the MTU's tried 3500 bytes is the optimum for this socket buffer setting, and the corresponding maximum achievable throughput is approximately 380 Mbps. By contrast for 128KB socket buffer size the maximum achievable throughput continues to increase until it attains a 470Mbps at MTU of 9000 bytes. Note also that for any fixed MTU, using socket buffer size of 128KB increases the maximum achievable throughput but the effect is more noticeable for larger MTUs.

The second observation is the effect of MTU on the anomaly mentioned earlier. We observed that the drop at 3KB for 1500 bytes MTU starts to shift as we increase the MTU size regardless of socket buffer size. We believe that this is the effect of packet bursting.

The third observation is the performance drop for block size greater than 64MB. This is not a surprise at all, since our machines are only equipped with 64MB of memory, larger messages involve virtual memory. This just confirms the well known fact that using virtual memory leads to severe performance degradation.

MTU	Alteon		SysKonnect	
bytes	sock=64KB	sock=128KB	sock=64KB	sock=128KB
1500	353	356	326	331
4500	314	453	506	510
8500	240	446	496	561
9000	248	424	512	613

TABLE 4.1
TCP/IP Performance with Large Socket Buffer and MTU

Figure 4.5 shows the TCP/IP throughput and Table 4.1 gives the peak throughput, between Pentium III/450s with 256MB of PC100 SD-RAM, for various Gigabit Ethernet NICs

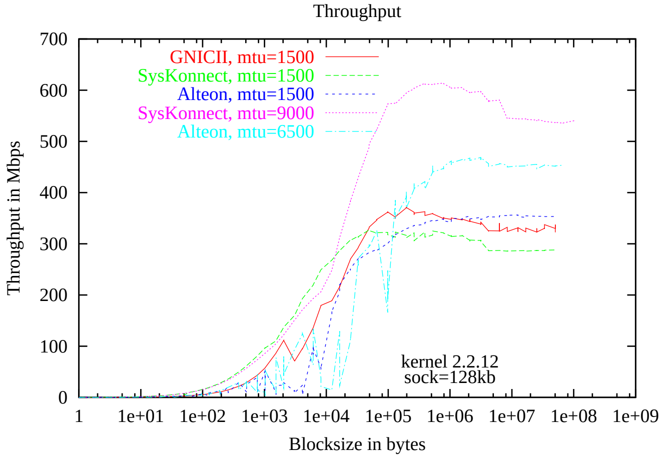

FIG. 4.5. *GNIC-II, Alteon, & SysKonnect: TCP Throughput and Latency*

on a later Linux kernel 2.2.12. In the figure, we present the TCP/IP performance with MTU of 1500 bytes for all Gigabit Ethernet NICs and the MTU for ACEnic (MTU=6500) and SK-NET (MTU=9000) that achieves the highest peak throughput. One obvious observation from the figure is there are many severe dropouts in ACEnic TCP/IP performance. The reason for these dropouts is due to the default tuning of the ACEnic device driver. These parameters were set to optimize disk throughput speeds at the expense of latency. This results from the practice of deferring transmission of small packets, thus increasing the latency dramatically and decreasing the overall throughput of messages which involve a final small packet. This could be changed by changing parameters of the driver before compilation. However the original results are included to emphasize the importance of testing the performance of a new driver version and tuning parameters appropriately for one's application.

For MTU of 1500, the maximum attainable throughput is approximately 371 Mbps, 356 Mbps, and 331 Mbps for GNIC-II, ACEnic, and SK-NET respectively. And, the latency is approximately 137 μsecs, 100 μsecs, and 45 μsecs for GNIC-II, ACEnic, and SK-NET respectively. Because of its lower latency, SK-NET is able to perform much better than the GNIC-II for message sizes up to 49KB and than the ACEnic for message sizes up to 128KB. For example, for message size of 16KB, SK-NET throughput is approximately 32% more than the GNIC-II and 30% more than the ACEnic. This emphasizes that latency is the primary factor in determining the performance for small messages. Table 4.1 shows the latency and throughput for various sizes of MTU and socket buffer sizes for kernel 2.2.12.

5. Virtual Interface Architecture.

5.1. VIA Fundamentals. As we have seen in the previous sections, latency is the most significant factor in the overall throughput attainable for most message sizes. The VIA [4, 13, 14] interface model was developed to reduce the overhead caused by moving messages through different layers of the Internet protocol suite of TCP/UDP/IP in the operating system. In the past, this overhead did not significantly contribute to 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 a more significant factor in overall latency. In this section, we present a brief description of VIA and some performance results. A more detailed description and results will appear in [3].

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. It also 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. Note that implementation of VIA only requires additional software, primarily network device drivers, rather than the modifications to hardware, firmware, and standards required to change the Ethernet MTU.

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 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.

Currently, there is significant focus on the emerging InfiniBand architecture[22]. InfiniBand is a switched-fabric I/O technology that ties together servers, network devices and storage devices. In general, InfiniBand incorporates much of the VIA technology and behavior, merged with Next Generation I/O (NGIO, led by Intel) and Future I/O (led by Compaq) standards[23]. Although it has also introduced many new concepts, its main design is still largely based on the VIA primitives.

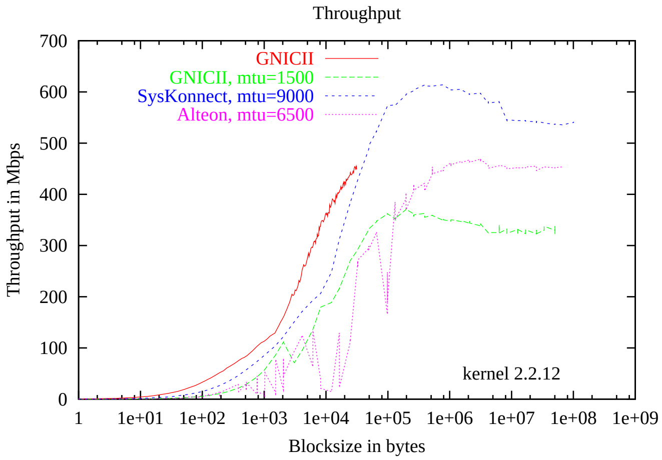

FIG. 5.1. *GNIC-II: M-VIA Throughput*

5.2. VIA Performance. Figure 5.1 shows the throughput and latency, between Pentium III/450s with 256MB of PC100 SD-RAM running the 2.2.12 version of the Linux kernel, 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 the 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 448Mbps with latency of only 16 μ secs. Thus, the throughput is approximately 53%, 74% and 4% more than GNIC-II, ACEnic, and SK-NET respectively.

6. MPI Version Performance. In this section, we present and compare the performance of LAM, MPICH, and MVICH on a Gigabit Ethernet network connecting Pentium III/450s each with 256MB of PC100 SD-RAM. A more detailed comparison of performance is contained in [18]. MPICH is a reference implementation of MPI from Argonne National Laboratories. LAM was originally developed at the Ohio Supercomputer Center (OSC) and is now being developed and maintained at Notre Dame University. It was intended as an implementation optimized for use on a network of workstations in terms of ease of use and performance. 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. A long message is segmented into packets with 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.

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 switch-over point to 128KB instead of the default 64KB. For MPICH, we used all the default settings. Figure 6.1 and Figure 6.2 show LAM and MPICH throughput graphs respectively.

NIC	LAM		MPICH	
	Throughput(Mbps)	Latency (μsecs)	Throughput(Mbps)	Latency (μsecs)
GNIC-II	216	140	188	142
ACEnic	191	111	226	164
SK-NET	210	66	249	99

TABLE 6.1
LAM and MICH Performance for MTU of 1500

Table 6.1 gives the throughput for 64KB message and the latency for MTU of 1500

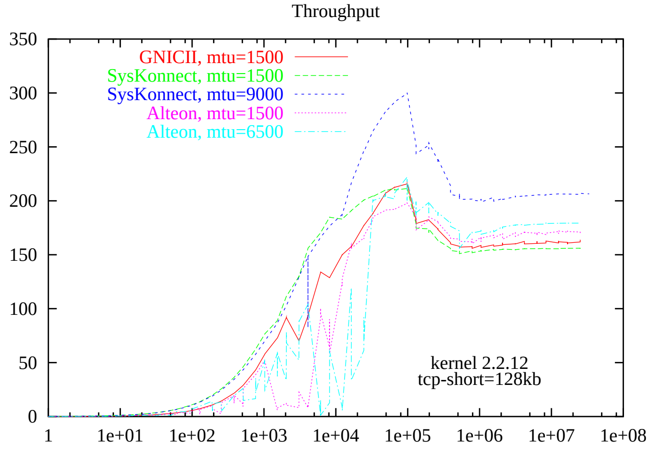

FIG. 6.1. *GNIC-II, Alteon, & SysKonnect: LAM Throughput*

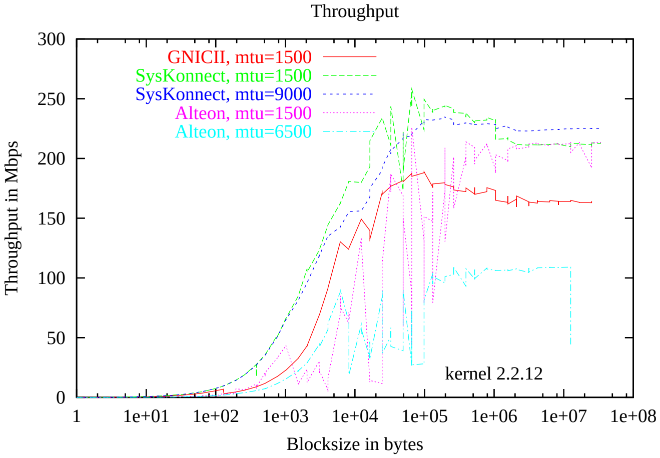

FIG. 6.2. *GNIC-II, Alteon & SysKonnect: MPICH Throughput*

bytes. Note that the amount of performance degradation in LAM and MPICH as compared to the TCP/IP performance is considerable. For LAM, the performance drops approximately 42%, 46% and 41% for GNIC-II, ACEnic, and SK-NET respectively. And, the performance drop for MPICH is approximately 49%, 37%, 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 with MTU of 9000, and by approximately 14% for the ACEnic with MTU of 6500 respectively. However, changing MTU to a larger size decreases MPICH performance. For MPICH, the maximum attainable throughput drops by approximately 7% for an SK-NET with MTU of 9000, and by approximately 52% for an ACEnic with MTU of 6500. In all cases, increasing the size of the MTU also increases the latency slightly except in the case of the test on MPICH with MTU of 6500 using the ACEnic. Again, we see that the severe dropouts in performance that we saw for TCP for the ACEnic also appear in these results.

A few remarks can be made on the causes of these effects. As noted for Table 4.1, TCP/IP performs better on a Gigabit Ethernet network for larger 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.

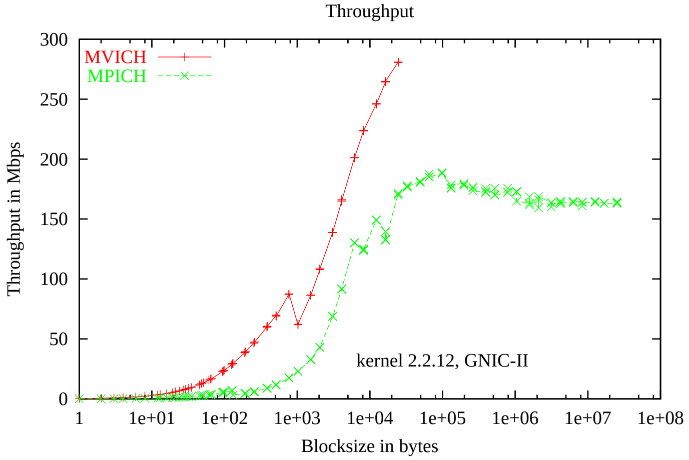

FIG. 6.3. *GNIC-II: MVICH Throughput*

Further investigation of the effects of tuning the MPICH implementation is 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.

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. Thus we might expect that an MPI implementation, that uses VIA as a transport protocol, could achieve lower latency and higher throughput than one using TCP/IP. Figure 6.3 compares MPICH and MVICH performance, and shows that MVICH attains a maximum throughput of 280Mbps with latency of only 26 μsecs for message sizes as low as 32KB. Messages greater than 32KB are segmented by MVICH but in the early version tested here we were unable to run messages 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.

7. NAS Benchmarks. In this section, we consider the performance of a sample application on the Beowulf cluster at Kent State. In particular, we compare the performance attainable using Fast and Gigabit Ethernet using LAM and MPICH. We also compare the performance of MPICH, which is implemented over TCP, with that of the beta implementation of MVICH, which is implemented over M-VIA. The tests were performed on `fianna`, the Kent State Computer Science Beowulf cluster described in Section 2.

The sample application in question is the NAS simple three dimensional multigrid benchmark [1, 2]. Results are given for the two main classes Class A and Class B, which involve 4 and 20 iterations respectively of a V-cycle multigrid algorithm used to obtain an approximate solution u to the discrete Poisson problem $\nabla^2 u = v$ on a $256 \times 256 \times 256$ grid with periodic boundary conditions. Results are also given for the smaller workstation class benchmark, called Class W, which consists of 40 V-cycle iterations on a $64 \times 64 \times 64$ grid.

	Class W		Class A		Class B	
procs	FE	GE	FE	GE	FE	GE
1	47.64	46.07				
2	75.30	83.33	77.99	81.76	83.87	87.57
4	106.71	124.83	134.99	147.02	144.39	157.81
8	166.59	212.86	295.99	321.24	317.94	348.01
16	240.44	294.28	539.97	589.31	577.80	636.91
32	263.41	346.56	830.15	1087.03	894.57	1125.53

TABLE 7.1
Total Mflops using LAM

	Class W			Class A			Class B		
procs	FE	GE	GE	FE	GE	GE	FE	GE	GE
			MVICH			MVICH			MVICH
1	47.01	47.10	46.87						
2	72.70	79.32	86.31	76.61	80.29	80.94	84.53	87.52	87.59
4	107.38	114.82	138.56	138.57	147.75	148.74	149.52	156.82	159.04
8	163.80	181.52	257.13	295.92	325.26	339.38	320.23	342.61	365.59
16	210.06	242.78	403.52	530.86	558.24	620.58	569.07	615.44	665.68
32	238.00			792.09			878.67		

TABLE 7.2
Total Mflops using MPICH/MVICH

Table 7.1 and 7.2 give the total Mflops for the NAS benchmarks using LAM, and MPICH and MVICH respectively. Table 7.3 gives the speedup for MPICH and MVICH. We remark that we were unable to run the benchmarks using MPICH or MVICH on 32 processors. It is clear from these tables that use of Gigabit Ethernet leads to substantially better performance than Fast Ethernet for Class W, A and B. Further, use of MVICH leads to substantially better performance than MPICH for all of the problem classes. In the case of the Class W problem on 16 processors, the speedup increases by nearly a factor of two.

	Class W			Class A			Class B		
procs	FE	GE	GE	FE	GE	GE	FE	GE	GE
			MVICH			MVICH			MVICH
1	1.00	1.00	1.00						
2	1.55	1.68	1.84	2.00	2.00	2.00	2.00	2.00	2.00
4	2.29	2.44	2.96	3.62	3.68	3.68	3.54	3.58	3.63
8	3.49	3.85	5.48	7.73	8.10	8.39	7.58	7.83	8.35
16	4.46	5.14	8.60	13.86	13.91	15.34	13.46	14.07	15.20
32	5.05			20.70			20.79		

TABLE 7.3
Speedup for MPICH/MVICH

8. Conclusions. In this paper, we discussed a number of issues which could impact the performance of Beowulf clusters based on general purpose networking hardware. In particular we compared the performance of point-to-point communication using TCP, and two MPI implementations (LAM and MPICH). We indicated the importance of tuning certain network parameters such as RFC1323, socket buffer size, and MTU, and testing new driver implementations to ensure that they are optimized for our applications.

We emphasized the importance of latency in determining the performance of networks and indicated how the VIA proposal could ameliorate this problem. We presented some results for a beta version of a VIA driver M-VIA and an MPI implementation MVICH, which uses it. We remark that the results included were for M-VIA v0.01 and MVICH v0.02 which are very early implementations. M-VIA version 1.2b2, released in June 2001, includes beta features support for VIPL Reliable Delivery, including conforming error handling semantics and retransmission of lost packets on Ethernet networks. This is in addition to the features of the first beta (1.2b1), such as optional VIPL peer-to-peer connection APIs and support for both 2.2.x and 2.4.x Linux kernels.

REFERENCES

[1] D. Bailey, J. Barton, T. Lasinski, and H. Simon, *The NAS Parallel Benchmarks*, Report RNR-91-002, NASA/Ames Research Center, January 1991.

[2] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga, *The NAS Parallel Benchmarks,* RNR Technical Report RNR-94-007, March 1994.

[3] Mark Baker, Paul A. Farrell, Hong Ong, Stephen L. Scott, *VIA Communication Performance on a Gigabit Ethernet Cluster*, 10 pages, (to appear in the Springer-Verlag Lecture Notes in Computer Science, Proceedings of EuroPar2001).

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

[5] Paul A. Farrell and Hong Ong, *Communication Performance over a Gigabit Ethernet Network*, 19th IEEE International Performance, Computing, and Communications Conference – IPCCC 2000, pp. 181–189, 2000.

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

[7] 10 Gigabit Ethernet Alliance, *10 Gigabit Ethernet Technology Overview White Paper.* (2001) `http://www.10gea.org/`

[8] 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).

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

[10] J. Gustafson and Q. Snell, *HINT: A New Way to Measure Computer Performance*, Proceedings of the Twenty-Eight Hawaii International Conference on System Sciences,Wailea, Maui, Hawaii, January 1995.

[11] J. Gustafson and R. Todi, *Conventional Benchmarks as a Sample of the Performance Spectrum*, `http://www.scl.ameslab.gov/Publications/HICSS98/HICSS98.html`

[12] HINT - Hierarchical INTegration benchmark web site, `http://www.scl.ameslab.gov/Projects/HINT/`

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

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

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

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

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

[18] Hong Ong, Paul A. Farrell, *Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network*, Proceedings of Linux 2000, 4th Annual Linux Showcase and Conference, Extreme Linux Track, Atlanta, 2000, pp 353–362.

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

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

[21] D. A. Wood, M. D. Hill, *Cost Effective Parallel Computing*, IEEE Computer, 28(2) (1995), pp. 69–72.

[22] InfiniBand Trade Association. `http://www.infinibandta.org`

[23] InfiniBand Trade Association. *InfiniBand Architecture Specification, release 1.0*, October 2000. `http://www.infinibandta.org`