Protocol-Dependent Message-Passing Performance on Linux Clusters

Dave Turner and Xuehua Chen
Ames Laboratory – Iowa State University
327 Wilhelm Hall, Ames, Iowa, 50011
Email: turner@ameslab.gov

Abstract

In a Linux cluster, as in any multi-processor system, the inter-processor communication rate is the major limiting factor to its general usefulness. This research is geared toward improving the communication performance by identifying where the inefficiencies lie and trying to understand their cause. The NetPIPE utility is being used to compare the latency and throughput of all current message-passing libraries and the native software layers they run upon for a variety of hardware configurations.

1. Introduction

Almost every modern parallel processing system is made of processors with decent performance coupled with a reasonable amount of local memory and access to local or global disk space. The main limiting factor in most systems is the inter-processor communication rate. This limits the efficient use of the processing power available, and the ability of applications to scale to large numbers of processors.

PC/workstation clusters use processors of the same caliber as those in traditional massively parallel systems, but the communication rate between nodes is much lower. Research is being done to improve the communication performance between machines using both commodity and proprietary network hardware. Work is also being done to overcome the limitations on the software side, allowing applications to take full advantage of the performance that the network offers.

Inefficiencies can occur at many levels between the application and hardware layers. Up to 50% of the raw performance can be lost in the message-passing layer alone. The operating system and driver often add to the message latency and decrease the maximum bandwidth by doing many memory-to-memory copies of the data as each message is packetized for transmission. This extra data movement results in the saturation of the main memory bus, which typically occurs well before the PCI bus gets saturated.

The first step in improving the overall performance of the message-passing system is to identify where the performance is being lost [1,2] and determine why. The research reported in this paper concentrates on evaluating the performance of the message-passing libraries on a variety of hardware configurations. In order to do that, it was also necessary to measure the performance of the lower-level communication systems that each runs on. The goal of this work is to test a broad enough set of hardware configurations to provide the basis for good generalized conclusions to be drawn as to the effectiveness of each message-passing library.

2. The testing methodology

The NetPIPE [3,4] utility performs simple ping-pong tests, bouncing messages of increasing size between two processors. Message sizes are chosen at regular intervals, and also with slight perturbations, to provide a complete test of the system. Each data point involves many ping-pong tests to provide an accurate timing. All latencies discussed in this paper are small message latencies representative of the round trip time divided by two for messages smaller than 64 bytes.

NetPIPE modules have been developed to directly test MPI [5-7], PVM [8,9], TCGMSG [10], TCP, GM, SHMEM, and LAPI (no VIA interface yet). This allows for a direct comparison between the existing message-passing libraries, and to the native communication layers they run upon. These measurements are not affected by the choice of a compiler or compiler options.

NetPIPE measures the point-to-point communication performance between idle nodes. This provides an upper bound to the performance that an application could achieve, since there is no measurement of the effect that a loaded CPU would have on the communication system. Several of the message-passing libraries tested allow message traffic to progress independent of the main application thread, and should therefore provide better performance for real applications. Testing the performance within real applications would therefore be useful in determining the effect of these two factors beyond the information that NetPIPE provides.

Most of the graphs shown in this paper are from two 1.8 GHz Pentium 4 PCs with 768 MB of PC133 memory and 32-bit 33 MHz PCI slots. These are taken as typical PCs for building clusters, costing around $1000 each. All tests were done back-to-back with no intervening switch, except for the Giganet VIA tests. The machines ran RedHat 7.2 with the Linux 2.4.7-10 kernel, except for the M-VIA tests that needed the 2.4.2-2 kernel and some tests with the older 2.2.19 kernel to determine the difference in performance.

Two dual-processor Compaq DS20 computers with 500 MHz Alpha 21264 processors were also used to provide a basis for comparison with the PC results. These machines have 64-bit 33 MHz PCI slots, offering greater PCI performance than the 32-bit PCI bus of the Pentium 4 PCs.

Four sets of Gigabit Ethernet cards were tested. The TrendNet TEG-PCITX copper GigE cards (32-bit 33 MHz, ns83820 driver) cost $55 each, representing the new wave of low cost GigE NICs. The Netgear GA622 copper NICs cost $90 and are identical to the TrendNet cards except that they can take advantage of the 64-bit PCI bus. The older Netgear GA620 fiber GigE cards (32/64-bit 33/66 MHz, AceNIC driver) cost more at $220. The SysKonnect SK-9843 GigE cards (32/64-bit 33/66 MHz, sk98lin driver) are expensive at $565, but have a very low latency and provide a high bandwidth when jumbo frames of 9000 byte MTU size are enabled. The performance of proprietary Myrinet and Giganet network hardware was also tested.

While all these Gigabit Ethernet cards were tested on both the PCs and the Compaq DS20s, only a few graphs are presented here. The Netgear GA620 cards on the PCs represent mature hardware and drivers at a modest price. The TrendNet cards on the PCs represent the new low-cost generation of GigE cards. The SysKonnect cards with jumbo frames enabled on the 64-bit PCI bus of the DS20s demonstrate what each library can do in a faster environment.

3. The message-passing libraries

The most recent versions of each message-passing library were tested on the most current kernel (RedHat 7.2 Linux 2.4.7-10) and driver supported. A few omissions include CHIMP/MPI [11], which apparently has not been supported since 1997, EMP [12], a NIC-driven message-passing system, and GPSHMEM [13], a general-purpose implementation of the SHMEM library that allows one-sided communications on clusters.

3.1. The MPICH library

MPICH 1.2.3 [14,15] is a portable version of the MPI standard freely distributed by Argonne National Laboratory. It is also the base implementation for many vendor and research libraries. MPICH-GM tests on Myrinet NICs used a slightly older version 1.2.1..7 of MPICH with GM 1.5, while MVICH 1.0 started with MPICH 1.2.1 and used cLAN 2.0.1 VIA drivers.

MPICH is a source code distribution, so it takes some effort to configure and compile while many other message-passing systems now come as easy-to-install Linux RPMs. On Unix systems, MPICH runs on top of its p4 library [16,17] implemented as a blocking channel device. Progress on data transfers is only made during MPI library calls.

The primary optimization parameter is the *P4_SOCKBUFSIZE* environment variable that is vital to maximizing the performance. Other optimization parameters include the *–p4sctrl* run-time flag and the *P4_WINSHIFT* variable. The thresholds are not designed to be user tunable, but can always be modified in the source code. The rendezvous cutoff, where MPICH changes to a handshake protocol, can be increased this way by changing *128000* in mpid/ch2/chinit.c and mpid/ch_p4/chcancel.c to the desired value. The handshake requires the exchange of small messages before data is sent, adding twice the latency time to the transmission time. This can result in a dip in performance for systems with large latencies. For older versions of MPICH, you may need to use the *–use_rndv* configuration flag in order to use the rendezvous method for large messages.

3.2. The LAM/MPI library

LAM/MPI 6.5.4-1 [18] (Local Area Multicomputer) is a full MPI implementation freely available from Indiana University. It is available as an easy-to-install RPM for Linux, currently included in the RedHat distribution.

The *lamd* daemons allow LAM/MPI to operate in a heterogeneous environment and provide advanced monitoring and debugging features. The *lamboot* command easily starts the daemons, and then codes are run using the familiar *mpirun* command.

Using the *–O* flag in homogeneous environments greatly improves performance. The *–lamd* argument to *mpirun* directs all messages through the daemons, providing access to the monitoring and debugging facilities, but greatly reducing the performance.

3.3. The MPI/Pro library

MPI/Pro 1.6.3 [19] is a commercial version of the full MPI standard available from MPI Software Technology for around $100 per node with an 8 node minimum. It is distributed as an easy-to-install RPM for Linux that runs directly on TCP, and has interfaces for VIA and GM. MPI/Pro uses a separate thread to actively manage the progress of all messages.

Increasing the *–tcp_long* or *–via_long* parameters diminishes the effects from the extra handshaking at the rendezvous threshold. The *–tcp_buffers* run-time parameter did not help in the NetPIPE tests.

3.4. The MP_Lite library

MP_Lite 2.3 [20] is a lightweight message-passing library developed by the authors. It supports a restricted set of the MPI commands, including blocking and asynchronous send and receive functions, and many common global operations. It does not support many of the advanced capabilities of MPI such as the use of communicators, derived data types, parallel I/O, and the one-sided communications of MPI-2.

MP_Lite is freely distributed by Ames Laboratory, compiles in a few seconds, and works well for running small MPI programs. It is being used as a research tool for investigating methods to improve message-passing performance. The results presented here are from the SIGIO interrupt driven module that keeps data flowing through the TCP buffers by trapping SIGIO interrupts sent when data enters or leaves a TCP socket buffer. Message progress is therefore maintained at all times.

MP_Lite increases the TCP socket buffer sizes up to the maximum level allowed. The only tuning available is to increase the maximum socket buffer sizes allowed by the system. For Linux, this can be done by assigning values like *net.core.rmem_max = 524288* and *net.core.wmem_max = 524288* in /etc/sysctl.conf.

3.5. The PVM library

PVM 3.4.3 is the Parallel Virtual Machine message-passing system developed at Oak Ridge National Laboratory. It has a completely different syntax from MPI as well as some different functionality.

PVM now comes as an easy-to-install RPM for Linux in the RedHat distribution. The *pvm* utility provides an easy way to start the *pvmd* daemons, but is a bit tricky if communicating on an interface other than the primary one (xpvm was not tried, and may be easier to use for this case).

There are many optimizations to sift through, several of which make a very large difference. The default configuration will send all messages through the *pvmd* daemons, which limits the performance greatly. All applications should use the *pvm_setopt*() function to choose direct communications when possible. Using *PvmDataInPlace* in *pvm_initsend*() prevents copying of the data before and after transmission.

3.6. The TCGMSG library

TCGMSG 4.04 is the Theoretical Chemistry Group Message-Passing Toolkit distributed by Pacific Northwest National Laboratory with the Global Arrays 3.1.8 package. It has a very limited number of functions, and is designed simply to provide an interface between applications like the NWChem quantum chemistry code and either TCP or an underlying message-passing library like MPI. In TCGMSG, the *SND* function blocks until the matching *RCV* has been completed.

4. Performance on Gigabit Ethernet

These message-passing libraries run directly on top of TCP sockets, so it is vital to understand the raw performance and tuning characteristics of TCP across the various hardware configurations. You cannot just slap in a Gigabit Ethernet card and expect to achieve decent performance like you can with more established Fast Ethernet technology. The default OS tuning levels have not kept pace with what is needed to communicate at higher speeds.

As an example, the performance of the TrendNet GigE cards flattens out at 290 Mbps when the default TCP socket buffer sizes are used. Increasing these to 512 kB eliminates the socket buffer size as a limiting factor, doubling the raw throughput. Each node opens 2 socket buffers for each machine in a run, so 512 kB socket buffer sizes should not place too high of a burden on the memory in most moderately sized clusters.

The heavy black lines at the top of figures 1 and 2 show that the raw TCP performance reaches a maximum of 550 Mbps on both the Netgear GA620 fiber NICs and the cheaper TrendNet cards. The latencies are poor under the new Linux 2.4.x kernel, at 120 µs and 200 µs respectively. Figure 3 shows that the 9000 Byte MTU jumbo frames on the SysKonnect cards plus the 64-bit PCI bus of the Compaq DS20s provides a raw TCP performance up to 900 Mbps with a low 48 µs latency. On the PCs, the 32-bit PCI bus limits the bandwidth of these SysKonnect cards to a maximum of 710 Mbps.

These TCP curves provide the maximum performance that each message-passing library strives for. Except where noted, the small-message latencies for the message-passing libraries are pretty close to that of raw TCP and therefore will not be dwelled upon.

4.1. MPICH performance

Increasing *P4_SOCKBUFSIZE* from its default 32 kB up to 256 kB is vital. This raised the maximum throughput from 75 Mbps up to 400 Mbps for a 5-fold increase in performance. The *–p4sctrl* and *P4_WINSHIFT* parameters did not help in these tests.

Once *P4_SOCKBUFSIZE* is optimized, MPICH performs respectably on all the Gigabit Ethernet cards. Figures 1-3 show that MPICH does suffer a 25% - 30% loss in each case for large messages transfers. The most noticeable feature is the sharp dip at 128 kB in figure 1 where MPICH starts using a large-message rendezvous mode that requires a handshake before data is sent. This is not a user tunable parameter, but it is possible to change it in the source code.

4.2. LAM/MPI performance

On the Netgear GigE cards, LAM/MPI tops out at 350 Mbps when no optimizations are used. For homogeneous systems, using *-O* brings the performance nearly to raw TCP levels. The convenience of running with the *lamd* daemons comes at a large price, cutting the performance down to 260 Mbps and doubling the latency to 245 µs.

Figure 1 shows that the only real deficiency is from a slight dip in performance at the rendezvous threshold, which is apparently not user-tunable. On the 64-bit PCI bus and jumbo frames of the SysKonnect cards on the DS20s, figure 3 shows that LAM/MPI loses about 25% of the performance that TCP offers for large messages. Figure 2 shows even more severe problems on the cheaper TrendNet cards, where LAM/MPI and many of the other message-passing libraries suffer from a 50% loss in performance.

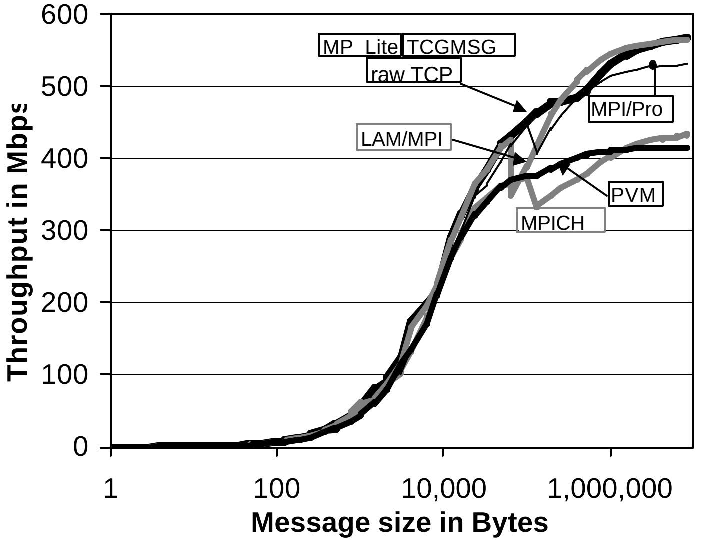

Figure 1. The message-passing performance across the Netgear GA620 fiber Gigabit Ethernet cards between PCs. The MP_Lite and TCGMSG curves were left off since they fell nearly on top of the TCP curve and had no interesting features.

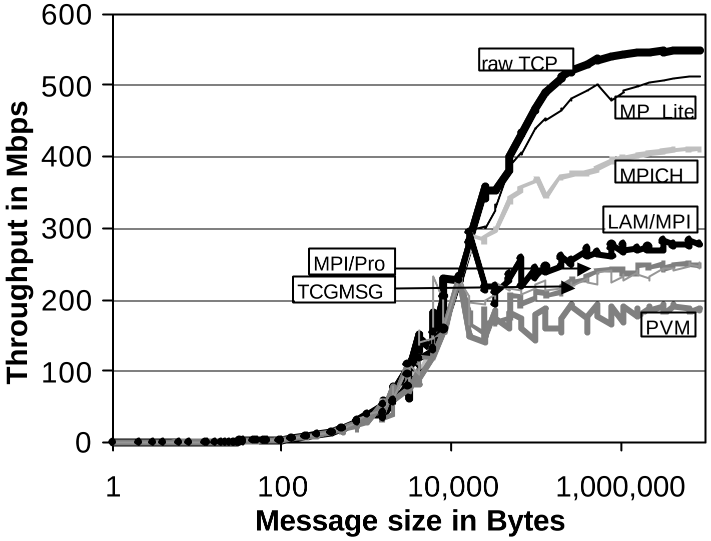

Figure 2. The message-passing performance across the TrendNet TEG-PCITX copper Gigabit Ethernet cards between two PCs.

4.3. MPI/Pro performance

MPI/Pro comes out of the box fairly well tuned. Increasing the *–tcp_long* parameter from the default 32 kB to 128 kB removes much of a dip in performance at the rendezvous threshold.

With the *–tcp_long* parameter optimized, figure 1 shows that MPI/Pro performs exceedingly well on the Netgear cards, getting to within 5% of the raw TCP results. However, MPI/Pro also has severe problems with the cheaper TrendNet cards, flattening out at 250 Mbps. MPI/Pro performs very well on the SysKonnect cards between PCs. Support for the Alpha Linux environment has just recently become available, but not in time to fully test the performance and include the results in this paper.

4.4. MP_Lite performance

Figures 1-3 show that MP_Lite matches the raw TCP performance to within a few percent on all GigE cards (the MP_Lite curve was left off figure 1 because it laid almost on top of the raw TCP curve). The only tuning needed was to increase the maximum socket buffer sizes on the system. Preliminary results on an MPICH-MP_Lite implementation at the channel interface layer show that this performance can be passed along to the full MPI implementation of MPICH.

4.5. PVM performance

The default configuration for PVM sends all messages through the *pvmd* daemons, which limits performance to around 90 Mbps across the Gigabit Ethernet cards. Bypassing the daemons using the *pvm_setopt*() function to choose direct communications produces a 4-fold increase to a maximum of 330 Mbps. Using *pvm_initsend(PvmDataInPlace)* prevents PVM from copying the data before and after transmission, further increasing the maximum transfer rate to 415 Mbps.

With these optimizations, PVM provides performance similar to MPICH, losing 25%-30% of what TCP offers. However, figure 2 shows that PVM has trouble with the TrendNet cards where it is limited to only 190 Mbps. PVM does not take much advantage of the greater bandwidth offered on the SysKonnect cards on the Compaq DS20s, providing a maximum of only 500 Mbps compared to the 900 Mbps that raw TCP offers.

4.6. TCGMSG performance

Even though little development has been done on TCGMSG since 1994, it passes on nearly all the performance that TCP offers. The TCGMSG curve falls to within a few percent of the raw TCP curve in figure 1, and was therefore left off. This is not surprising since it is only a thin layer on top of TCP. Curiously, it still suffers on the TrendNet cards, where performance is limited to 250 Mbps, and on the SysKonnect cards using jumbo frames on the DS20s where the throughput tops out at 500 Mbps.

TCGMSG can also run on top of an MPI library. NetPIPE measurements showed that there is no performance lost by running TCGMSG-MPICH compared to MPICH alone, though the fact that a TCGMSG *SND* blocks until the matching *RCV* is completed may affect real applications more.

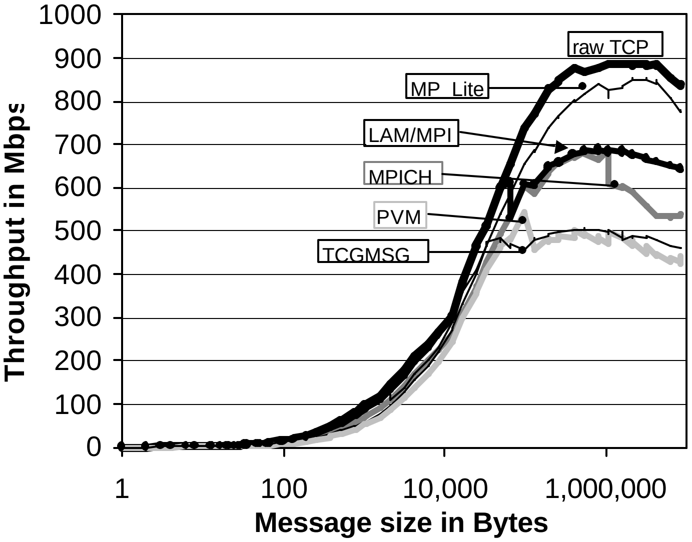

Figure 3. The message-passing performance using a 9000 Byte MTU size across SysKonnect SK-9843 Gigabit Ethernet cards between two Compaq DS20s.

5. Performance on Myrinet cards

The proprietary Myrinet PCI64A-2 cards from Myricom [21,22] have 66 MHz RISC processors that are slower than the 133 MHz PCI64B and 200 MHz PCI64C versions currently available. All are 64/32-bit 66/33 MHz cards starting at $1200 each with switches running $500 per port.

Myricom provides a native GM library and a full MPI implementation through MPICH-GM. The system can also be configured to run IP over GM directly. MPI/Pro also has a Linux RPM that runs directly on GM.

There are several tunable parameters that affect GM, and therefore MPICH-GM. The *--gm-recv* flag sets the mode to the default *Polling*, or to *Blocking* or *Hybrid*. All produce approximately the same results, except that the *Blocking* mode has a latency of 36 µs compared to 16 µs for the others. In general, the *Hybrid* mode should be used as it provides the same results as the *Polling* mode but should not burden the CPU as much. The default Eager/Rendezvous threshold of 16 kB is already optimal.

Figure 4 shows that the raw GM performance reaches a maximum of 800 Mbps with a 16 µs latency. MPICH-GM and MPI/Pro-GM results are nearly identical, losing only a few percent off the raw GM performance in the intermediate range. The NetPIPE TCP module was used to show that IP-GM has a latency of 48 µs compared to 32-200 µs for TCP over the Gigabit Ethernet cards, but otherwise offers similar performance.

6. VIA hardware and software

VIA is the Virtual Interface Architecture [23,24], an API and software layer designed to facilitate faster communication between computers. Its goal is to provide a more streamlined path between the application and the network, allowing zero- or one-copy transmissions that bypass the overhead of the operating system.

M-VIA 1.2b2 [25] is a modular VIA package developed by researchers at Lawrence Berkeley National Laboratory. This is a software implementation of the VIA API that runs on some Fast Ethernet and Gigabit Ethernet devices on Linux systems, including the sk98lin device driver for the SysKonnect cards. This is a research project in progress with only beta releases at this time.

The Giganet CL1000 cards tested cost around $650 each. These are hardware VIA implementations, tested with the cLAN 2.0.1 device driver. The Giganet tests were performed using an 8-port CL5000 switch, costing around $500 per port.

6.1. The VIA libraries

MVICH 1.0 [26] is an MPICH 1.2.1 ADI2 implementation that runs on top of VIA. It has been developed and tested on M-VIA supported devices, plus ServerNet and Giganet hardware. MP_Lite 2.3 also has a VIA module tested on M-VIA and Giganet. MPI/Pro has a VIA module tested on Giganet hardware.

MVICH has many user-tunable parameters. It is vital to configure MVICH using *–DVIADEV_RPUT_SUPPORT* to get good performance. Setting *–via_long* to 64 kB gets rid of a dip due to the rendezvous threshold, but increasing it higher caused the system to freeze up. Increasing *VIADEV_SPIN_COUNT* to 1000 and raising the rendezvous threshold to 128 kB helped the performance of MVICH in the intermediate range. No benefit was seen from the other optimization parameters.

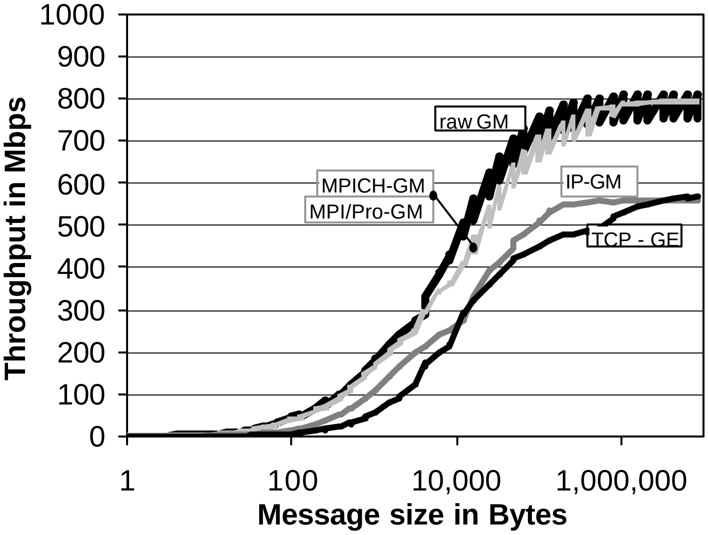

Figure 4. The message-passing performance across Myrinet PCI64A-2 cards between two PCs.

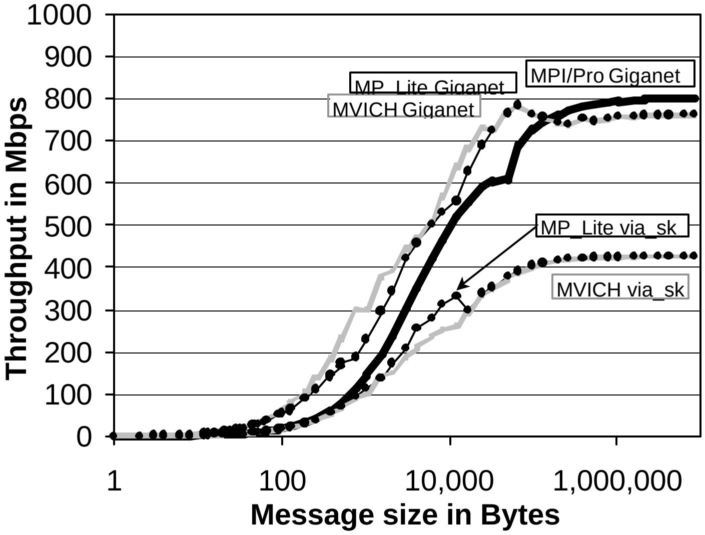

Figure 5. The message - passing performance across Giganet CL1000 cards and using M-VIA across SysKonnect SK-9843 Gigabit Ethernet cards between PCs (MVICH is gray, MP_Lite has dots).

6.2. Performance on VIA hardware and M-VIA

Figure 5 shows the performance comparison of all these systems fully tuned. MPI/Pro, MVICH, and MP_Lite all produce maximum communication rates around 800 Mbps on the Giganet hardware. MVICH and MP_Lite have latencies of 10 μs, while MPI/Pro has a greater overhead at 42 μs.

MVICH and MP_Lite/M-VIA using the via_sk98lin device across the SysKonnect cards reached a maximum of 425 Mbps with a 42 μs latency. The small dip at 16 kB is at the RDMA threshold. Unfortunately, this is approximately the same performance that raw TCP offers for this hardware configuration.

7. Discussion

The NetPIPE results presented here show that most message-passing libraries perform reasonably well under the right conditions and when properly tuned. It should be stressed that it does take some effort to measure and tune the operating system and message-passing libraries for optimal performance. The default parameters are often not set appropriately for modern high-speed hardware, but tuning a few simple parameters can increase the communication performance by as much as a factor of 5.

Figure 1 shows that most message-passing libraries can deliver performance close to raw TCP levels on the more expensive Gigabit Ethernet cards. MPICH and PVM currently suffer about a 25% loss in performance for large messages. Similar results were gotten using the SysKonnect cards with a 9000 Byte MTU on the PCs, where MPICH and PVM still suffer 25-30% losses but the other libraries delivered within 10% of the TCP communication rate of 710 Mbps.

MPICH uses the p4 library to interface with the TCP layer on Unix systems. p4 receives all messages to a buffer rather than directing them to the application memory when a receive has been pre-posted. MPICH therefore must use a *memcpy* to move all incoming data out of the p4 buffer, causing the loss in performance for large messages.

The cheaper TrendNet cards gave most of the libraries more of a problem. Only MP_Lite and MPICH worked well, with many message-passing libraries reaching only 250-300 Mbps. The raw TCP measurements on these cards showed that they needed larger socket buffer sizes in order to achieve peak performance. MP_Lite and MPICH both have user-tunable parameters for the socket buffer size, and therefore appropriate tuning produces good performance.

For those libraries where source code is provided, it is possible to change the socket buffer sizes manually and recompile the code. For TCGMSG, increasing *SR_SOCK_BUF_SIZE* in sndrcvP.h from 32 kB to 256 kB brought the performance up to raw TCP levels. The poor performance of the other message-passing libraries on the TrendNet cards should likewise be remedied by simply increasing the socket buffer sizes. Unfortunately, this currently involves finding the parameters in the code and recompiling.

The Compaq DS20 tests using the Netgear GA622 copper cards, which are identical to the TrendNet cards except for being able to take advantage of the 64-bit PCI bus, showed poor performance even for raw TCP. Great care must be taken in evaluating these new GigE cards. Newer ns83820 drivers appearing in the 2.4.19-pre kernels and the Netgear *gam* drivers both show improved performance and stability. More tuning of these drivers is still needed, and is likely to reduce their need for large socket buffers.

The SysKonnect jumbo frames tests on the Compaq DS20s illustrated in figure 3 show a great difference in performance between the message-passing libraries. Once again, the socket buffer size plays a major role in the ultimate performance of each library. To demonstrate this, the socket buffer size hardwired into the TCGMSG source code was increased from 32 kB to 128 kB, resulting in the performance increasing from 500 Mbps to 900 Mbps, matching raw TCP. Similar results may occur if the socket buffer sizes were increased in the other libraries.

Custom hardware, while expensive, does provide better performance than Gigabit Ethernet. Figure 4 illustrates that Myrinet cards can provide both low latency and high bandwidth, with a 16 µs latency and 800 Mbps maximum rate. MPICH-GM and MPI/Pro-GM pass nearly all this performance on to the application layer. IP over GM offers little more than TCP over Gigabit Ethernet on these systems, but at a greater cost.

VIA hardware such as the Giganet CL1000 cards likewise provides impressive performance, but again at a high price. MVICH, MPI/Pro and MP_Lite all deliver around 800 Mbps to the application layer. MP_Lite and MVICH provide applications with around 10 µs latencies, while MPI/Pro is significantly higher at 42 µs.

The M-VIA project is designed to provide greater performance through bypassing the inefficiencies of the operating system, allowing better performance on existing Ethernet hardware. Unfortunately, these limited tests on the SysKonnect cards between PCs show that MVICH/M-VIA and MP_Lite/M-VIA provide about the same performance as raw TCP. More tests are needed to fully explore the capabilities of M-VIA.

It should be remembered that all these NetPIPE tests represent an upper bound for the ultimate performance. The libraries are internally very different, and therefore will react differently within real applications. A message-passing library like MPI/Pro that has a message progress thread, or MP_Lite that is SIGIO interrupt driven, will keep data flowing more readily.

8. Conclusions

Overall, the message-passing libraries pass on most or all of the performance that the underlying communication layer offers. Most of the deficiencies could be easily corrected by simply increasing the socket buffer sizes. This was not the case just a few years ago when small-message latencies were 2-3 times the raw TCP levels and up to 30-40% of the throughput could be lost in the message-passing layer. This represents the progress that the developers of each library have made in optimizing their code, and also the fact that the processing speed in computers has increased faster than the networking capability.

All graphs presented here were after optimization of the available parameters. A graph of the performance before optimization would show drastically different results. It is vital to take the time to measure and optimize the performance of the OS and message-passing system when dealing with gigabit speed hardware.

The developers of the message-passing libraries need to update some of the default optimization parameters to reflect today's faster networks. They all need to provide user-tunable parameters for the most important optimizations, such as the socket buffer sizes and the rendezvous threshold. If this is done, most would be able to deliver a majority of the performance that TCP offers

9. Acknowledgements

The authors would like to thank Ricky Kendall, Shoba Selvarajan, and Weiyi Chen for their contributions to this project, and to MPI Software Technology for providing evaluation copies of their MPI/Pro software. This work is supported by the Applied Mathematical Sciences Program of Ames Laboratory – U.S. Department of Energy under contract number W-7405-ENG-82.

10. References

[1] Hong Ong and Paul A. Farrell, "Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network," *Proceedings of the 4th Annual Linux Showcase & Conference*, Atlanta, October 10-14, 2000.

[2] R. Dimitrov and A. Skjellum, "An Efficient MPI Implementation for Virtual Interface (VI) Architecture-Enabled Cluster Computing," *Proceedings of the Third MPI Developer's Conference*, March 1999.

[3] NetPIPE: http://www.scl.ameslab.gov/netpipe/

[4] Quinn O. Snell, Armin R. Mikler, and John L. Gustafson, "NetPIPE: A Network Protocol Independent Performance Evaluator," *IASTED Conference Paper*.

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

[6] MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8 (3/4) 165-416, 1994.

[7] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongara, *MPI: The Complete Standard*, MIT Press, Cambridge, Massachusetts, 1996.

[8] PVM: http://www.csm.ornl.gov/pvm/

[9] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, *PVM: Parallel Virtual Machine*, MIT press, 1994.

[10] TCGMSG: http://www.emsl.pnl.gov:2080/docs/parasoft/tcgmsg/tcgmsg.html

[11] CHIMP: ftp://ftp.epcc.ed.ac.uk/pub/chimp/release/

[12] P. Shivam, P. Wyckoff, D. Panda, "EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing," *Proceedings of SC2001*, Denver, CO, November 2001.

[13] K. Parzyszek, J. Nieplocha and R.A. Kendall, "A Generalized Portable SHMEM Library for High Performance Computing," *Proceedings of the IASTED Parallel and Distributed Computing and Systems 2000*, Las Vegas, Nevada, November 2000, (M. Guizani and X. Shen, Eds.), pp. 401-406. IASTED, Calgary (2000).

[14] MPICH: http://www-unix.mcs.anl.gov/mpi/mpich/

[15] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "High-performance, portable implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, 22(6):789-828, September 1996.

[16] p4: http://www.mcs.anl.gov/~lusk/p4/

[17] J. Boyle, R. Butler, T. Disz, B. Glickfeld, E. Lusk, R. Overbeek, J. Patterson, and R. Stevens, *Portable Programs for Parallel Processors*, published by Holt, Rinehart & Winston, 1987.

[18] LAM/MPI: http://www.lam-mpi.org/

[19] MPI/Pro: http://www.mpi-softtech.com/

[20] MP_Lite: http://www.scl.ameslab.gov/Projects/MP_Lite/

[21] Myricom: http://www.myri.com/

[22] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.K. Su, "Myrinet: A Gigabit-per-second Local Area Network," *IEEE Mirco*, Vol. 15 No. 1, pgs 29-36, February 1995.

[23] VI Developers Forum: http://www.vidf.org/

[24] Compaq, Intel, and Microsoft. *Virtual Interface Architecture Specification, Version 1.0*. December 1997, http://www.viarch.org/.

[25] M-VIA: http://www.nersc.gov/research/ftg/mvia/

[26] MVICH: http://www.nersc.gov/research/ftg/mvich/