Analyzing the Performance of MPI in a Cluster of Workstations based on Fast Ethernet [⋆]

J. Piernas, A. Flores, J. M. García

Dept. de Informática y Sistemas
University of Murcia
Campus de Espinardo, s/n 30080 Murcia (Spain)
email: {piernas, aflores, jmgarcia}@dif.um.es

Abstract. Recent improvements in LANs make network of workstations a good alternative to traditional parallel computers in some applications. MPI (Message Passing Interface) standard library implementation for network of workstation holds great promise in providing both portability and performance. However, some authors point out if this hardware improvements are not supported with a reduction of software overhead, this overhead will dominate communication cost, and many applications may no benefit from the advancements in network hardware. In this paper we present an exhaustive study of MPICH implementation of MPI standard for this environment, analyzing added overhead of each layer. Finally, we propose modifications to network interface to reduce the fixed overhead of transmission.

1 Introduction

Research in parallel computing has traditionally focused on multicomputers and shared memory multiprocessors. Currently, networks of workstations (NOWs) are being considered as a good alternative to parallel computers. That is due to there are high performance workstations with microprocessors that challenge custom-made architectures. This class of workstations is widely available at relatively low cost. Furthermore, these networks provide the wiring flexibility, scalability and incremental expansion capability required in this environment.

Nowadays, and following the principle of allowing the production of portable software, NOWs are programmed using conventional imperative languages, enhanced with communication libraries such as PVM or MPI to implement message passing and synchronization among processes. MPI [5] holds great promise in providing both portability and, through a very rich model for inter-process communication, performance efficiency for application, library and compiler developers. The active participation of numerous vendors in the standardization process and the appearance of both portable and platform-specific implementations of the library bode well for its success. In this environment, the efficiency

[⋆] This work was supported in part by the Spanish CICYT under Grant TIC94-0510-C02-02 of parallel applications is maximized when the workload is evenly distributed among workstations and the overhead introduced in the parallelization process is minimized: the cost of communication and synchronization operations must be kept as low as possible. In order to achieve this, the interconnection subsystem used to support the interchange of messages must be fast enough to avoid becoming a bottleneck.

Communication performance at the application level depends on the collaboration of all components in the communication system, especially the network interface hardware and the low-level communication software that bridges the hardware and the application. Currently, there are new layers with reduced overheads (that is, on the scale of tens of instructions per message). Active Messages [4] is one of this. Traditionally, latency and throughput are the two parameters used to indicate the performance of an interconnection network. Although these parameters are very related, it is easier to increase the peak throughput -achievable for long messages-, but it is harder to reduce the latency -because the software overhead-. Unfortunately, the performance of parallel applications is very sensitive to latency when interchanging small messages among parallel processes. In this paper, we report the experimental data obtained from running tests on a cluster of workstations based on a Fast Ethernet card. In the past, benchmark results of MPI were carried out for workstation clusters for both point-to-point and collective communication [7] [6]. Our paper has two new approaches with respect these previous reports. By one hand, we analyze the software overhead for MPICH subroutine among the different levels that it can be decomposed. In this way, we can fix the different levels of software overhead and try to reduce it. By other hand, we compare the performance between the two current implementations of Ethernet: 10 Mbps and 100 Mbps. Our investigation suggests various ways to improve the performance of NOWs. This will make NOWs a competitive approach to parallel distributed computation for an important class of applications.

The rest of the paper is structured as follows. In the next section a brief introduction to the MPI communication library is made. Then, we describe deeper the main parameters to characterize the performance of a NOW. In section 4, a description of the important MPICH layers is shown. The results of running test programs are analyzed in section 5. Finally, some conclusions and ways of future work are drawn.

2 MPI: A standard for parallel programming

In order to this paper be self-contained, we are going to introduce briefly the main concepts of MPI. Message Passing Interface (MPI) has become an emerging standard for implementing message-based parallel programs in distributed-memory computing environments. One major goal of the MPI is to provide a widely portable and ease-of-use programming library without sacrificing the performance. To establish an efficient standard for many platforms, the MPI provides several mechanisms to perform point-to-point and collective communications. The performance of these mechanisms may be varied depending on the software implementation and the underlying hardware. Several other features, such as derived datatype, persistent communication, and group concept, are also introduced to improve the ease-of-use. However, if the MPI is not carefully implemented, the communication cost can be so expensive that programs may not gain any benefit from parallel processing.

MPI is only a standard upon which many implementations exist. The one we have used is MPICH that is a portable implementation of MPI developed jointly by Argonne National Laboratory and Mississippi State University. MPICH contains an abstract device interface (ADI) upon which a high-level message passing application programmer interface such as MPI can be implemented. The ADI performs four main functions: sending and receiving, data transfer, queuing, and device-dependent functions.

In our case, we can differentiate four levels, each one provides support for the next level: sockets, p4, ADI and MPICH.

3 Characterizing the Network Interface Performance

In this section we propose a point-to-point communication model for multilayer network interface. Modeling a communication system require to measure a number of parameters enough to characterize the performance of the key resources. Several authors have proposed models in order to achieve this goal.

In [2], authors define the LogP model where a point-to-point message communication is characterized by four parameters: L, the latency experienced in each communication event; o, the total length of time that a processor is engaged in the transmission or reception of each message; g, the minimum time interval between consecutive sends or successive receives by a processor, and P, the number of processor/memory modules.

In [6], authors propose other parameters to characterize a message transmission between two processes: t_{send}, the spend time to do the packet processing, mainly message copying, packetizing, and checksum computing, t_{net}, the delay in the communication channel, and ,t_{recv}, the processing packet time in the receiver.

Both models show problems in a multilayer communication system. For low level communication systems, as active messages [4] and multiprocessor communication, an accurate parameter measurement is possible for the LogP model. In a multilayer environment, the real overhead and latency is hidden to the upper layer by the lower one. The splitting packet processing between all these layers and the extensive using of queuing in each layer make difficult an accuracy measure of L and o. In the second model, we only measure the overhead seen by the upper level, the total overhead is hidden.

More than an accurate measure of temporal cost of each communication stage, we are interesting in determine the difference between fixed temporal cost and cost per byte in each message. So we propose a simpler model where each point-to-point communication time is modeled as

$$t(n) = \alpha_{send} + \beta \times n + \alpha_{recv} \quad (1)$$

where α_{send} and α_{recv} are the fixed temporal cost in the sender and receiver, respectively, and β is the extra cost per byte. For convenience, we consider α_{send} equal to α_{recv}. In this case, equation 1 follows as

$$t(n) = 2 \times \alpha + \beta \times n \quad (2)$$

As we have noted in the experimental results, this assumption is correct due to linear behavior of temporal curves.

4 Software overhead in MPICH

As Local Area Networks improve the performance, their interfaces, resources, organization, and integration into their host computer become increasingly important. Recent papers have pointed out that announced performance for such as LANs could suffer a severe degradation due to overhead in communication software. With the announcement of Ethernet Gigabit standard for this year, bottlenecks study and alternative solutions become essential.

To make an exhaustive study software overhead, we will evaluate the communication channel performance at several layers in the protocol stack in order to identificate bottlenecks. Using this technique we can measure the added overhead of each layer, the total software overhead and the network congestion.

In order to measure the capability degradation in low latency LANs, probes we are made in both Ethernet and Fast Ethernet environments. Comparing results in both cases, we want to discover critical bottlenecks in order to achieve a better network performance.

We have centered our study in the MPICH implementation of MPI standard library. We can see an overview of MPICH software layers in figure 1.

To measure the added overhead of MPICH implementation, we decided to apply our model in the three main layers: MPICH layer, socket TCP layer, and driver layer. Modeling the MPICH layer inform us about the whole overhead seen by users programs. The socket layer parameters tell us about the optimizations provided by the operating system. Finally, a driver layer characterization shows the maximum performance achieved by the card. In order to evaluate this last level, we have used the modified version of active messages (AM) implemented by the University of Genoa [1].

5 Result evaluation

5.1 Hardware and Software platforms

We have performed our tests on a cluster of workstations with Intel Pentium 200Mhz processor, 32 Mbytes main memory, 256KBytes cache memory, 1GByte IDE hard disk and Fast Ethernet 3Com 905-network adapter.

The operating system used is Linux 2.0.27 and the MPI implementation is MPICH 1.0.13.

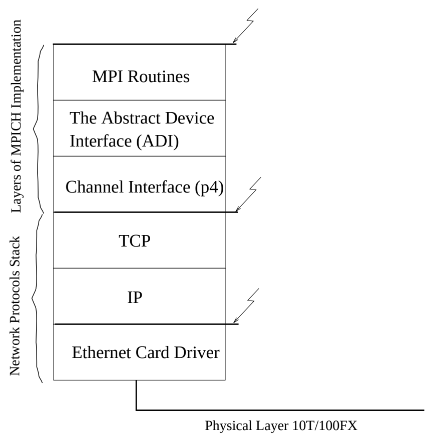


Fig. 1. MPICH Implementation Layers Overview. Broken arrows show layers chosen to evaluate the implementation.

5.2 The test program

To evaluate α and β parameters, we make use of Ping-Pong test. This test program is written in standard C. The best compiler option is always applied, unless otherwise noted. The test Ping-Pong is a simple echo between two adjacent nodes. A receiving node simply echoes back whatever it is sent, and the sending node measures round-trip time. Times are collected for some number of iterations (we have used 2000) over various messages sizes. Data rate, or bandwidth, is calculated from the number of bytes sent divided by half the round-trip time. The minimal time, the maximal time and the mean time from all processes are collected. To interpret the results, we focus on the average time, because they are more representatives of the performance the user can obtain from the machine.

5.3 Results

The results that we have obtained can be seen in the follow figures. We have differentiated between short messages (from 0 to 4192 bytes) and long messages (from 6 Kbytes to 22 Kbytes). For AM, an overrun problem arises for messages longer than 8 Kbytes. To avoid this problem we split long messages into chunks shorter than 8 Kbytes that are sent consecutively. Another important fact is that we have had to change from 3:5 to 5:3 the Rx:Tx RAM split parameter in the network adapter in order to limit overrun problem due to receiver saturation. This change has improved the network behavior.

Figure 2 shows us bandwidth achieved by MPICH for 10 and 100 Mbps Ethernet. From these results, an important different between 10Mbps and 100Mbps networks arises. In 10 Mbps Ethernet performance is limited by network bandwidth, while in Fast Ethernet software overhead becomes the bottleneck. This

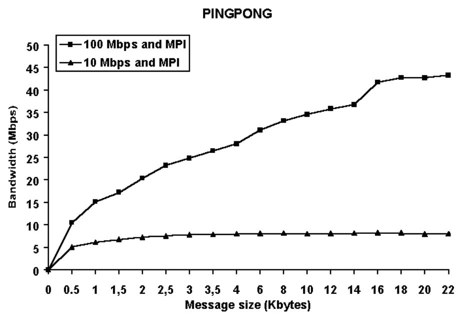
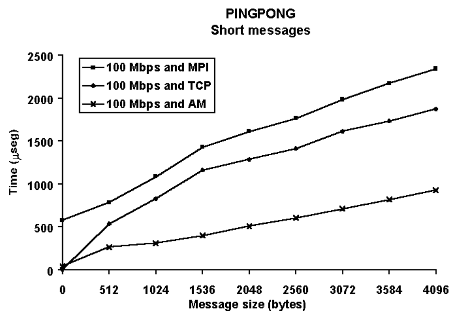


Fig. 2. Comparison between 10 and 100 Mbps MPICH results.

overhead made bandwidth obtained to be less than a half of network hardware bandwidth.



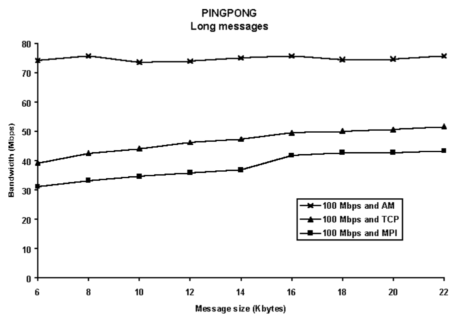


Fig. 3. Ping pong communication time. Short messages.

Fig. 4. Ping pong bandwidth. Large messages.

Figures 3 and 4 show communication time for short messages and the bandwidth achieved for large messages. As it can be noted, results with TCP and AM are better than with MPICH, because the MPI implementation overhead is removed.

5.4 Analysis of results

From figures shown in previous section several interesting conclusions follow. First of all, contrary to what it can be thought, MPICH overhead is mainly due to socket layer. TCP layer software overhead represents from 75% to 81% of MPICH overall time. Besides, the curve parallel behavior suggests this overhead remains constant regardless message size.

Another interesting fact that can be noted is there are three zones of linear behavior regard message size. Table 1 confirms this fact. In this table we show

	AM		MPICH		TCP	
Interval	α	β	α	β	α	β
0-512	10.80	0.145	144.25	0.200	1.75	0.517
512-1024	18.50	0.115	120.00	0.295	61.25	0.285
1024-1536	32.50	0.088	99.00	0.336	41.25	0.324
1536-2048	20.50	0.104	219.75	0.179	196.5	0.122
2048-2560	26.50	0.098	250.75	0.148	193.5	0.125
2560-3072	21.50	0.102	168.25	0.213	104.75	0.194
3072-3584	15.50	0.105	211.75	0.185	224.75	0.116
3584-4096	12.00	0.107	245.00	0.166	189.75	0.136
4096-6144	33.00	0.097	170.50	0.202	146.75	0.157
6144-8192	28.50	0.099	202.00	0.192	197.75	0.140
8192-10240	76.50	0.124	198.00	0.193	138.75	0.155
10240-12288	29.75	0.103	255.50	0.182	248.75	0.133
12288-14336	73.25	0.096	240.50	0.184	175.25	0.145
14336-16384	104.00	0.099	1488.25	0.010	430.75	0.109
16384-18432	148.00	0.124	330.25	0.152	142.75	0.145
18432-20480	38.75	0.104	2.75	0.188	145.00	0.144
20480-22528	111.25	0.096	287.25	0.159	365.00	0.123

Table 1. Fixed overhead and cost per byte (in μsecs) for main layers.

α and β parameters for each pair of points. First zone goes from 0 to 1536 bytes where β (slope) is relatively high. Next zone goes from 1536 to 14336 bytes where slope is flatter. Finally, third zone begins at 16Kbytes where slope is even flatter. Table 2 shows α and β parameters of each zone obtained by linear regression where r is the linear correlation coefficient.

	AM			MPICH			TCP		
Interval	α	β	r	α	β	r	α	β	r
0-1536	14.51	0.116	0.994	140.21	0.248	0.994	17.60	0.367	0.990
1536-14336	16.89	0.104	1.000	204.33	0.190	1.000	174.34	0.146	1.000
16384-22528	6.72	0.107	0.998	180.46	0.170	0.997	201.43	0.138	0.999

Table 2. Linear regression coefficients and correlation coefficient for main layers.

From both tables we note that α is usually much bigger than β. In some cases, α is more than 1400 times β, so a future optimization approach must focus on reducing communication constant factors. This problem is especially important in short messages where the start-up time dominate the communication cost.

One approach to solve this is using active messages. Active messages reduce both factors eliminating message buffering and providing a thin software layer between hardware and user applications. Another approach to reduce the software overhead is to use a complementary solution named *virtual circuit caching* (VCC) [3]. In this approach, the fixed communication cost, α, is amortized along several messages allowing a reduction effective communication time per message. The VCC model enables reduction in communication overheads by allocating communication resources once (e.g., buffers, interface channels) and re-using them for multiple messages to the same destinations. Currently, we have carried out some tests based on this approach. Preliminary experiments with this technique show hopeful results.

6 Conclusions and future work.

As we have shown in previous section, software overhead becomes the main problem as increasing raw network performance. We have modeled this overhead as a fixed cost per message, α, and a cost per transmitted byte, β. For MPICH implementation, TCP layer software overhead range from 75% to 81% overall time. We also have noted that α is usually much bigger than β, so a future optimization approach must focus on reducing communication constant factors.

Now we are working with a new approach named Virtual Circuit Caching (VCC). This technique is a complementary one to Active Messages. The VCC model enables reduction in communication overheads by allocating communication resources once and re-using them for multiple messages to the same destinations. Preliminary experiments with this technique show hopeful results.

References

1. G. Chiola and G. Ciaccio. GAMMA: a Low-cost Network of Workstations Based on Active Messages. Procc. of 5th EUROMICRO WorkShop on Parallel and Distributed Processing. London, January 1997.
2. D.E. Culler *et al.* LogP: Towards a Realistic Model of Parallel Computation. Procc. of the 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, May 1993.
3. B.V. Dao, S. Yalamanchili, and J. Duato. Architectural Support for Reducing Communication Overhead in Multiprocessor Interconnection Networks. Procc. of the Third Int. Symp. on High Performance Computer Architecture, February 1997.
4. T. Von Eicken *et al.* Active Messages: A Mechanism for Integrated Communication and Computation. Procc. of the 19th ISCA, pp. 256-266, May 1992.
5. Message Passing Interface Forum. MPI: A Message Passing Interface Standard. Int. J. of Supercomputer Applications and High Performance Computing, 8(3/4), 1994.
6. N. Nupairoj and L. Ni. Performance Evaluation of Some MPI Implementations on Workstation Clusters. Procc. of the Scalable Parallel Libraries Conf., Oct. 1994.
7. A. Skjellum, P. Vaughan, C. Roberts. UNIFY: Interoperable MPI and PVM Programming in a Workstation-Network Environment. MPI Developers Conference, University of Notre-Dame, June 1995.

This article was processed using the LaTeX macro package with LLNCS style