Low-latency and High-throughput message passing solutions

M. Maniezzo
Politecnico di Torino
Corso Duca degli Abruzzi 24,
I-10129 Torino (Italy)
marco.maniezzo@polito.it

A. Sanna
Politecnico di Torino
Corso Duca degli Abruzzi 24,
I-10129 Torino (Italy)
andrea.sanna@polito.it

Abstract

The main solutions currently adopted in deploying parallel applications are based on the use of high-performance parallel platforms and Networks of Workstations (NOW) exploiting off-the-shelf communication hardware. However, the former solutions are highly expensive, while the latter ones only achieve limited performances. An optimal solution consists in employing NOW (workstations or high-end PCs) combined with high performance Network Cards, effective parallel environments such as PVM or MPI and in modifying the standard communication protocol layer. In this paper, performance comparisons between PVM and MPI, as well as the optimizations achieved exploiting the GAMMA (Genoa Active Message MAchine) Active Message paradigm, are presented, and a GAMMA implementation for 3COM 3c966 NICs is proposed.

1. Introduction

Nowadays, research projects based on high-performance parallel platforms can be easily developed by using commercially available workstations interconnected by high-speed local area networks (LANs). This kind of configuration is called Network Of Workstations (NOW) and offers several advantages. The most important one is its low cost, as compared to expensive dedicated parallel hardware platforms, thanks to the steady decrease of the cost of commercial hardware and high speed LAN devices. Another considerable advantage is the possibility of simple replacement of each individual component with a newer and faster model; this allows the user to constantly update a NOW system. Dedicated parallel hardware cannot boast similar features. Once set up a NOW, it is necessary to install a parallel environment such as PVM[1] or MPI[2] in order to develop parallel/distributed applications. These softwares provide libraries of routines to be used in parallel user-defined applications.
Both architectures are based upon message passing approach and have the same structure of parallel routines. They differ in some features and, above all, have different performances. A simple benchmark application has been developed to test and compare PVM and MPI performances. The application sends and receives buffer of data in a ping-pong way. Execution times for both PVM and MPI are then measured, for different packet size. These tests have proved MPI is more efficient than PVM providing lower latency times and higher throughputs.
However, the NOW approach suffers for the poor performance of the communication mechanisms usually made available at the application level. The use of traditional protocols yields extremely high latency and very low message throughput as compared to the physical limits of the LAN hardware.
Let us consider, for instance, real-time parallel graphics applications, where constraints are the network latency rather then CPU calculation, and the problem of rendering a frame using several clustered machines; rendering operations cannot spend more than 1/30s for maintaining user interactivity. As explained later, we measured 100μs of latency with MPI; for massive parallel applications this latency can yield delays of the order of milliseconds, reducing the available time for calculation. With our implementation of GAMMA [3, 4], we can reduce up to 10% the latency time, with clear advantages for the above mentioned applications. The Active Messages approach is actually being implemented in projects such as [5, 6] based on expensive and high performance LAN technologies (e.g. ATM and Myrinet).
The GAMMA project demonstrates that the NOW approach can be profitably pursued even using standard, low cost LAN devices such as 100Mb/s Fast-Ethernet. Communication software layers have to be carefully designed according to a performance oriented approach exploiting the Active Messages communication paradigm. GAMMA has been developed with a modular structure, in order to be customized with every network device driver, enhancing the kernel of Unix-like Operating System, namely Linux. In the proposed work we take advantage of the GAMMA architecture by designing an ad-hoc driver for 3COM 3c996 NICs. Results obtained by means of different parallel test applications demonstrate:

- Advantages in using MPI (in particular the mpich 1.2.4 implementation) instead of PVM.

- Optimization in terms of lower latency and higher throughput of the GAMMA Active Messages MPI implementation compared with the original mpich 1.2.4.

2. Background

In this section the differences between MPI and PVM and the Active Messages mechanism will be briefly discussed.
Both MPI and PVM use a daemon-based communication system and use a similar set of routines for parallel programming (blocking/non blocking send and receive, broadcast send and so on). The main differences involve the communication system (level 2 ISO/OSI): PVM does a set of copies from user space to kernel space and a packing operation (needed to convert user defined data type, like structures, to standard data type) before sending data, that MPI does not perform. Moreover, PVM uses the UDP protocol, instead of TCP in MPI. These differences lead to worst performances.
Active Messages optimizes latency and throughput, while avoiding some defects typical of the TCP protocol. One is the memory to memory copy that occurs when the sender process writes the message content into a buffer allocated in its virtual memory space and then it starts the sending system call. The system call, in fact, invokes a copy of the buffer into the kernel memory space and, then, to the NIC sending buffer. Further delay is introduced on the receiver side, when a message is received. In fact the receive operation involves memory copies and context switching of the process, plus scheduling and de-scheduling waiting for the completion of the receiving buffer.
The Active Messages protocol introduces evident optimizations by copying directly the sending buffer (split in correct Ethernet frames) into the adapter's FIFO queue (after switching in kernel mode), without intermediate buffering. The same behavior occurs in the receive part: an interrupt signals that a frame has arrived, and data is copied from the adapter's input queue, to user memory space by an interrupt handler. The left part of Figure 1 illustrates the differences between traditional and Active Messages send operation, while the right part shows the differences in the receive part. GAMMA introduces further improvements by avoiding error recovery for the communication protocol (this is justified by the fact that frame corruptions occur seldom in switched LAN environments) and by exploiting the *interrupt ahead* and *forced fragmentation* mechanisms. The former consists in programming the receiver adapter in such a way that the interrupt request is raised upon receipt of the first few bytes of an incoming frame, rather than waiting for the complete frame to arrive in the receive queue. This allows compensating the latency time of the CPU in answering to an interrupt request. This kind of optimization is only available for Fast-Ethernet card, on which GAMMA was originally developed, but not for newer NICs such as Gigabit-Ethernet NICs. In this case, only the *forced fragmentation* can be exploited, which consists in calculating an optimal frame size to be used for fragments; GAMMA researchers have measured greater throughput, correctly tuning this parameter.

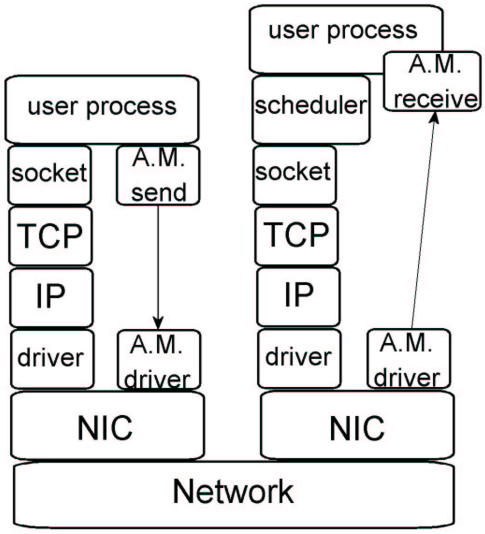

Figure 1. The Active Messages mechanism

3. GAMMA compatible NIC driver developing

The GAMMA driver is constituted by a core, which consists of a set of library functions written in C and built on top of a set of custom additional system calls enhancing the Linux 2.x kernel, and a set of macros that interface the GAMMA system calls (such as tx_enq_pkt for enqueue a frame or rx_read_pkt_info to read information about a received frame) to any NIC card. Following this approach, GAMMA can be extended to use almost all Fast and Gigabit-Ethernet cards.
The operations required to adapt GAMMA top a specific NIC can be divided in two phases:

1. Modify the original NIC driver in order to call GAMMA routines when the receive interrupt is raised, and to disable interrupts of transmission complete; even this operation need to be managed by the GAMMA protocol.

2. Create the macros needed by GAMMA.

In the first phase, it is necessary to modify the interrupt handler in the original driver, in such a way that the complete transmission interrupts are disabled (their

	mpi/GAMMA			mpich 1.2.4		
	Throughput[Mb/s]	Latency[μs]	t[s]	Throughput[Mb/s]	Latency[μs]	t[s]
Pov-Ray3.1	–	–	416	–	–	450
pingpong	756	12	0.0807	476	100	0.1281
NetPIPE	726	15	0.0880	472	69	0.1357

Table 1. Results

task is accomplished by the GAMMA macros, implemented in the second phase). This is necessary because the NIC buffers, may contain both IP packets (generated by normal applications) and GAMMA packets (prepared by GAMMA when parallel applications invoke GAMMA or mpi/GAMMA [7] routines), so after they have been sent, proper operations must be performed to empty the buffer; this is accomplished by GAMMA transmit macros.
As explained in the Background section, the major improvement introduced by GAMMA is avoiding the ISO/OSI protocol stack by mean of the Active Messages paradigm; in this way, GAMMA implements the entire communication system from the packet preparing phase, to the NIC send phase. Thanks to this capabilities, GAMMA protocol is able to use a dedicated network (with dedicated NIC) for parallel communication, separated from the IP network, thus achieving better performance; the complete transmission macros are intended to manage both network topologyof many NICs (IP and GAMMA separated) and the case of one NIC shared for GAMMA and IP communications.
The complete transmission macros scan the NIC send buffers for sent packets and prepare them to be reused (free used memory, reset variables and so on). Then, the receive interrupt handler must be redirected to another GAMMA macro; incoming frames can be from IP or GAMMA protocol, the receive macro manages both typologies of frame by computing the data of the header which every frame carries. Figure 2 shows the 20 bytes header used, and the two highlighted bytes reserved for IP and GAMMA distinction; currently they are 0xff and 0x00 for usual GAMMA communication, and 0xfe and 0x00 for set-up GAMMA communications.
The second phase involves the creation of the macros. Macros manage the send part by queuing packets (by means of pointers to buffers in the user memory space) and flushing sent packets from the queue; in the receive part, the macros divide IP frames (which are redirected to the original management routines) from GAMMA frames; the latter ones are handled by routines in the core of the GAMMA architecture, by checking the header for errors and then by using the data for the Active Messages routines.
The header must be read with a macro, because of the data format in the packet is not standard; GAMMA core code interfaces with different NICs by exploiting the macros mechanisms.
There are also macros that enables and disables all the interrupts of the NIC. These are necessary when GAMMA is configured in *polling receive*. In this operating mode, GAMMA uses polling to check when there are new incoming packets; in this way the CPU is always in use, but the performance is improved, when compared to normal interrupt method. This configuration can be profitable when the cluster is dedicated to parallel communication, with no other applications using the CPU.

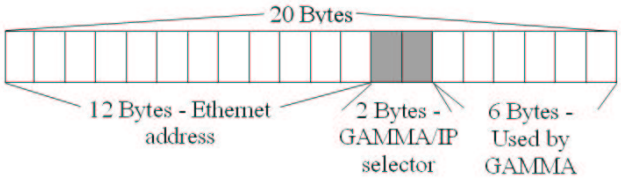

Figure 2. GAMMA header

4. Results and Conclusions

Once set up GAMMA working with 3COM 3c996, we initially used a cluster composed by two workstations to run tests with NetPIPE (Network Protocol Independent Performance Evaluator - http://www.scl.ameslab.gov/netpipe/), Pov-Ray 3.1 (a ray tracing program - http://www.povray.pov.org) and a "ping-pong" application.
A "ping" process runs on a node, sends a buffer of *n* bytes to a "pong" process and then waits for a buffer of the same size, in response. The size of the buffer is varied from 0 up to 8MB.
Such applications were executed with mpi/GAMMA (a porting of mpich 1.1.2 on the GAMMA driver) and with the original NIC driver and mpich 1.2.4. Total execution time, latency and throughput have been measured. Table 1 summarizes the results; for ping-pong and NetPIPE buffers of 0 byte and 8MB were used to measure the latency and throughput, respectively. An image at a resolution of 1600x1200 pixels and default parameters for lights have been computed by Pov-Ray; in this case, the only measured parameter has been the rendering time (Pov-Ray is a CPU-bound application rather than a network transmission one, therefore, throughput and latency cannot be calculated).
In Figure 3 are shown the graphs obtained with NetPIPE. In the upper one, i s shown the throughput (Mb/s) for buffers up to 8MB with mpich, mpi/GAMMA and TCP;

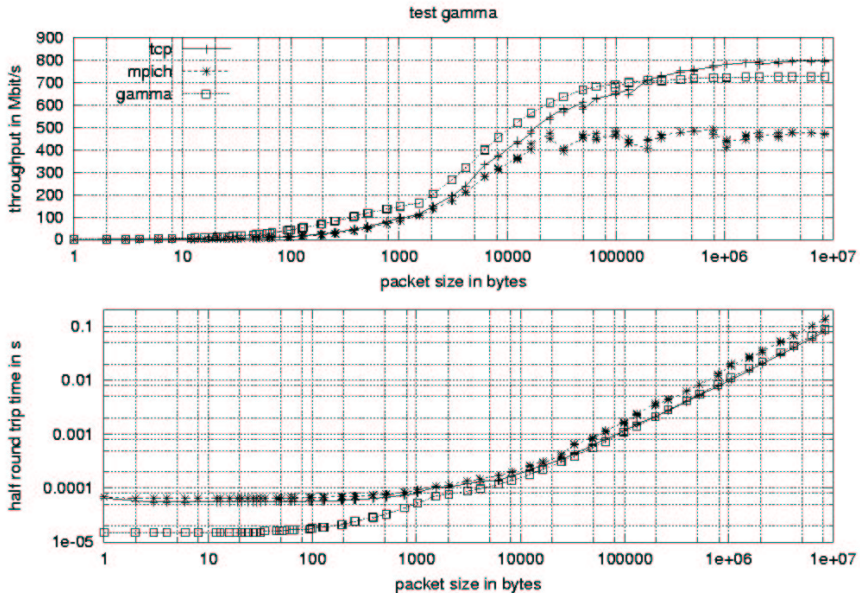

Figure 3. NetPIPE results

in the bottom the different values of latency (s), for the three different implementations are shown.
We achieved a gain of 58.8% for throughput with ping-pong and 53.8% with NetPIPE. Latency times measured testing the ping-pong application are about 7 times lower when mpi/GAMMA is used, and about 5 times lower using NetPIPE as test. Pov-Ray gains 8.1% using mpi/GAMMA compared with mpich; this gain is lower than the values measured for throughput and latency because the CPU-intensive (rather than network-intensive) nature of this application.
In conclusion, GAMMA Active Messages implementation demonstrates it improvements in communication throughput and latency; with its extensible architecture to almost all NICs, it can be a valid support for parallel applications with demanding communication requirements.

5. Acknowledgments

We thank Giuseppe Ciaccio (ciaccio@disi.unige.it) and Marco Elhert (mehlert@cs.uni-potsdam.de), for their important support in understanding GAMMA protocol and developing the macros for 3COM 3c996 Gigabit NIC. We also thank Willy Gardiol (gardiol@libero.it), for testing our implementation of GAMMA with NetPIPE and collecting performance results.

References

[1] V.Sunderam, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience*, pp. 315–399, December 1990.

[2] "The Message Passing Interface Forum. MPI: A Message Passing Interface Standard. Technical Report, University of Tennessee, Knoxville, Tennessee," 1995.

[3] G. Ciaccio, M. Ehlert, and B. Schnor, "Exploiting Gigabit Ethernet Capacity for Cluster Applications," in *Proceedings of 27th Annual IEEE Conference on Local Computer Networks (LCN 2002), Tampa, FL, USA*. LCM, November 2002, pp. 669–678.

[4] G. Chiola, G. Ciaccio, L. V. Mancini, and P. Rotondo, "GAMMA on DEC 2114x with Efficient Flow Control," citeseer.nj.nec.com/210324.html.

[5] T. von Eicken, V. Aula, A. Basu, and V. Buch, "Low-latency Communication Over ATM Networks Using Active Messages," *IEEE Micro*, vol. 15, no. 1, pp. 46–64, February 1995.

[6] S. Pakin, M. Lauria, and A. Chien, "High Performance Message on Workstations: Illinois Fast Messages (FM) for Myrinet Computation." in *Proceedings of Supercomputing '95, Nielson G. M. and Bergeron D.* San Diego, CA: ACM Press, 1995.

[7] G. Chiola and G. Ciaccio, "GAMMA and MPI/GAMMA on Gigabit Ethernet," in *Proceedings of 7th EuroPVM-MPI*. LNCS, September 2000.