GAMMA: a Low-cost Network of Workstations Based on Active Messages

Giovanni Chiola, Giuseppe Ciaccio
DISI, Universita' di Genova
via Dodecaneso, 35
16146 Genova, Italy
{chiola,ciaccio}@disi.unige.it

Abstract

Networks Of Workstations (NOW) are an emerging architecture capable of supporting parallel processing with significantly low cost/performance ratio. At the moment the implementation of standard high-level communication mechanisms in a NOW does not provide such a satisfactory cost/performance ratio, as modern communication hardware would allow. We show how a standard, Unix-like operating system kernel can be extended with efficient and performant low-level communication primitives based on the Active Message communication paradigm. Higher level standard communication libraries, like MPI, should be implemented on top of such efficient low-level mechanisms. We provide some preliminary results obtained from an experimental prototype called GAMMA (Genoa Active Message MAchine), which is a NOW whose nodes run the operating system Linux 2.0 enhanced with an Active Message communication layer.

1. Introduction

The usual protocol stack which is predominant in the Unix environment for inter-process communication (IPC), namely, the remote procedure call (RPC) level built on top of BSD Sockets or System V Streams built on top of TCP or UDP protocols built on top of the IP protocol now constitutes a de-facto standard that allows inter-operability of different machines. This advantage is payed in terms of efficiency of use of the communication hardware capabilities.

The performance of an Ethernet based LAN exceeded the needs of midrange workstations and "supermini computers" existing fifteen years ago even with the huge overhead of inefficient communication protocol layers. Today, when Ethernet based LANs become the bottleneck of a Network Of Workstations (NOW), the cheapest solution is to switch to faster hardware technologies (such as Fast Ethernet, FDDI, ATM, etc.) rather than producing new, more efficient software.

These economic considerations are correct in most but not all applications. One noteworthy exception is the case of NOWs used as hardware support to parallel processing. We claim that NOWs built out of cheap, fast, off-the-shelf computation and communication hardware components constitute the only hope for parallel processing techniques to spread. As a matter of fact, today no one interested in real applications would see the convenience of moving to parallel processing technologies involving substantially higher costs provided that each year single processors that are much faster and much cheaper than the ones available the year before can be found.

In principle, an efficient and well tuned NOW environment may well provide reasonably good levels of performance, as modern fast LAN devices like Fast Ethernet and ATM are not that much worse than custom, expensive communication networks used a few years ago in massively parallel platforms. The problem of defining a fast NOW-based parallel processing platform is delivering a fraction of the raw performance of communication hardware to the application level much larger than the one typically delivered by usual LAN environments through traditional network protocols.

The best approach to efficient parallel processing in a NOW is to design and implement efficient low-level interprocess communication primitives from scratch, and use these primitives to build higher-level communication mechanisms, like those of MPI[4], which would still enjoy a significant fraction of the raw communication performances.

We have built an extremely cheap NOW prototype called GAMMA (Genoa Active Message MAchine) equipped with an efficient interprocess communication layer based on the paradigm of Active Messages[6]. GAMMA is a network of 12 Intel Pentium PCs connected by a 100 Mb/s Fast Ethernet LAN. Each workstation runs Linux 2.0 enhanced with a custom device driver and Active Messages communication primitives.

Our result shows that GAMMA can beat very expensive parallel platforms such as the CM-5 in terms of message latency (while of course it cannot compete in terms of communication bandwidth due to the obvious difference in communication hardware complexity and cost). Yet, several parallel processing applications exist that would benefit by the availability of a truly cheap platform offering modest bandwidth, low latency communications.

2. The GAMMA architecture

2.1. Hardware configuration

The current prototype of GAMMA is composed of a set of 12 autonomous workstations connected by means of two independent LANs: one 10 Mb/s Ethernet used with the standard protocol suite to provide network services (such as NFS access to file servers, remote login, etc.) in the Unix environment and one 100 Mb/s, isolated Fast Ethernet dedicated to the implementation of fast inter-processor communication primitives.

Each workstation comprises:

- Intel Pentium 133 MHz CPU
- PCI mother board, 256 KB of 15 ns pipelined secondary cache, PCI Intel Triton chipset
- 32 MB of 60 ns RAM
- 3COM 3C595-TX Fast Etherlink 10/100BASE-T PCI network adapter.

The Fast Ethernet 100 Mb/s LAN consists of a 3COM LinkBuilder FMS 100 repeater hub with 12 RJ-45 ports, to which each Fast Etherlink adapter is connected by a UTP cable.

No communication protocol (except for the IEEE 802.3 which is implemented in firmware inside the PCI cards) is run over the 100 Mb/s LAN.

Each workstation runs the operating system kernel Linux 2.0.0 enhanced with our own communication layer described below.

2.2. The GAMMA Active Message layer

An inter-process communication is accomplished by the sender process invoking an Active Message "send" system call. The goal of this system call is to copy a portion of memory content from the user memory space of the sender process on one node to the user memory space of the receiver on another node.

Sending a message is accomplished by copying the message from user space into the network adapter using no intermediate buffering into kernel space. The overhead is very limited thanks to the low abstraction level of our communication protocol. Messages longer than 110 bytes are split into a sequence of Ethernet frames (each one of length ranging from 60 to 1536 Bytes) that are copied right away to the adapter's transmit FIFO. This poses neither protection problems (as writing to the adapter can only be accomplished through a system call) nor memory access problems (as the transmitting process is running when the transmission function copies from user space to the network adapter).

The Ethernet frames are eventually received by the network adapter on the receiver side. An interrupt handler on the receiving PC is then launched which copies the content of the adapter's input queue to the memory space of the receiver process, again without any intermediate kernel buffering. The frame headers provide the addressing information needed to locate the correct user space buffer where the incoming message is to be stored, as well information needed to correctly rebuild the message from several frames.

Upon completion of the memory-to-memory data transfer, a user defined *receiver handler* is called on the receiver node. The role of a receiver handler is to integrate the incoming message into the data structures of the receiver process and to prepare a user space buffer for the next message to be received. It is worth pointing out that in the Active Message approach each process belonging to a running parallel application has at least two distinct threads, namely:

- the main process thread which performs the computations and sends messages; this thread is subject to the usual schedule policy of the OS kernel;
- one or more receiver threads, corresponding to the execution of the receiver handlers; they are executed right away upon message reception under the control of the device driver's interrupt routine, without involving the OS kernel scheduler.

The receiver threads cooperates with the main process thread by sharing all the global data structures of the program. This multi-threading mechanism avoids the descheduling of the receiver process that is inherent to blocking message reception such as is usually implemented in Unix "sockets", thus substantially reducing latency.

The GAMMA communication protocol provides no message acknowledgement since the high cost in terms of loss of bandwidth and additional latency time is not worth-while in a communication system with extremely low probability of frame corruption. No explicit flow control is provided either, since if the receiver handler invoked upon message receipt completes execution quickly enough, then the fast LAN becomes the bottleneck of the communication system. Indeed flow control should be implemented at the application level in case of long receiver handler code.

However GAMMA implements an error detection feature, by allowing user defined *error handlers* to be called in much the same way as receiver handlers:

- under the control of the network driver's interrupt routine in case of checksum error or frame corruptions when receiving a message;

- under the control of the "send" system call in case of frame loss due to excessive collisions when attempting to send a message.

Receiver and error handlers may be also used to implement connected communication protocols that guarentee message delivery if needed at the application level.

The GAMMA communication layer is embedded into the original Linux kernel in the form of additional system calls and data structures. The additional system calls trap into kernel through a dedicated trap address (unused in the current Linux kernel) which leads to the GAMMA code through a particularly short and optimized code path. The reception software is part of the custom network device driver, and the interrupt routine of the network driver is registered into the Linux kernel as a "fast interrupt", which means that the kernel path from the interrupt request to the driver's interrupt routine is short as well.

2.3. The GAMMA computational model

In what follows, the words "workstation" and "PC" are used as synonyms.

A *physical GAMMA* is the set of M workstations connected to the fast LAN. Each workstation is addressed by the Ethernet address of the corresponding fast network card. A *virtual GAMMA* is a set of N$\leq$M computation nodes, each one corresponding to a distinct workstation in the physical GAMMA. The nodes are numbered from zero on and this numbering provides the necessary addressing of nodes at the application level in the most straighforward way. At the kernel level each of these numbers is mapped onto the Ethernet address of the corresponding PC.

A *parallel program* may be thought of as a finite collection of N processes running in parallel, each on a distinct node of a virtual machine. GAMMA supports parallel multitasking, i.e., more than one virtual GAMMA may be spawned at the same time on the same physical GAMMA. Each one runs its own parallel program on behalf of potentially different users.

As a consequence, any workstation may be shared by more than one parallel program at a given time. Each virtual GAMMA and the corresponding running parallel program is identified by a number which is unique in the platform. We call such an identification number a *parallel PID*. Processes belonging to the same running parallel application are characterized by the parallel PID of the application itself. The parallel PID is used to distinguish among processes belonging to different parallel applications at the level of each individual workstation. A maximum of 256 different parallel PIDs may be activated on a physical GAMMA.

Each process has 255 *communication ports* numbered from one on; port number zero is reserved for kernel-level fast communications like those occurring to create a virtual GAMMA and launch a parallel program or to synchronize within a barrier synchronization. Each port is bidirectional, which means that data can be both sent and received through it. The pointer to a user space buffer where incoming messages are stored may be associated to a port in order to use it as input. A user defined receiver handler and a user defined error handler may also be associated to an input port. Each port being used as an output must be mapped to another port of another process belonging either to the same parallel program or to a different one, thus forming an outgoing communication channel. Alternatively, an output port may be bound with N-1 input ports, each belonging to a process on a different node in the same virtual GAMMA, in order to broadcast messages. However in neither cases is the binding accomplished through an explicit connection protocol. Correctness and determinism of the communication topology induced by the port-to-port mapping is up to the programmer.

3. Related Work

Active Messages were originally proposed by researchers at the University of California at Berkeley [6], as a flexible and efficient low-level communication mechanism aimed at reducing communication overhead and allowing communication to overlap computation. The efficiency of the Active Messages communication mechanisms lies in the fact that they do not require message buffering, due to the immediate managing of messages by the user defined handlers.

A well known application of Active Messages as an efficient support to parallel processing is the Active Message Layer introduced by Thinking Machines Co. in the CM-5 platform [1].

The same idea has been followed by the FM project [2], using a fast LAN called Myrinet instead of the Fat Tree interconnection network of the CM-5.

Our modest yet original contribution to the idea of exploiting Active Messages for efficient parallel processing is the application to a cheap platform which exploits only off-the-shelf and cheap hardware devices. In order to experiment this idea we also chose a standard operating system for which source code is freely available, namely Linux.

A somehow similar approach was followed also by researchers at Cornell University in the framework of the U-net project [5]. U-net follows the idea of removing communication support from the OS kernel in order to re-implement it (more efficiently) at the user application level. A pre-defined number of virtual "end points" are multiplexed over the physical adapter. Each end point can be attached to a single user process by means of a system call. This way a virtualization of fast communication devices directly accessible by user processes is obtained which does not require any system call to send messages. An Active Message layer was implemented on top of such virtual fast communication devices. Both an ATM [5] and a Fast Ethernet [7] version of U-net were released.

4. Performance Measurements

We carried out some preliminary performance measures on a GAMMA equipped with a two PCs. A more extensive benchmarking is in progress.

We defined "delay" as the time interval between the instant the sender process invokes the "send" system call and the instant the interrupt routine on the receiver node terminates to copy the message into the user space receive buffer. We used a typical "ping-pong" application to carry out standard round-trip measures. The average message round-trip time divided by two yields the average delay, which of course is a function of the message size. We define *communication latency* as the delay of a zero sized message.

We define the "communication throughput" $T(s)$ as the transfer rate perceived by the application when sending a message of size s: $T(s) = \frac{s}{D(s)}$ where $D(s)$ is the message delay. We define *communication bandwidth* $B(s)$ according to the formula $B(s) = \frac{s}{D(s)-D(0)}$ where $D(0)$ is the communication latency as defined above.

The IEEE 802.3 standard allows for the frame size to range between 60 and 1536 bytes on the physical channel. With GAMMA, each frame has a 20 byte long header. Sending a message shorter than 40 bytes implies that less than 60 bytes are written to the network adapter, which in this case pads the frame with garbage bytes and transmits 60 bytes on the channel anyway. The header carries information about the size of the significant portion of the frame, so that the receiving CPU may copy only that portion to the user space receiver buffer.

Any message whose size ranges from 40 to 109 bytes results into a single full-sized Ethernet frame. However messages longer than 110 bytes are split into a sequence of frames. We realized that with frame bursts the communication system works as a pipeline, where the receiver CPU gets a previous frame from the RX FIFO queue on the receiver side while the adapters are transmitting the current frame and the sender CPU is writing the next frame in the TX FIFO queue of the sender side. Thanks to such pipeline effect, the fragmentation of medium sized messages into a large number of small frames yields better throughput. The optimal fragmentation of a given message depends on its size, and the optimal fragmentation policy was achieved experimentally. As a consequence of having such a fragmentation policy, the communication delay of messages ranging from 110 to 1516 bytes is no longer a linear function of message size.

The exploitation of cache affects the memory transfer rate. Therefore data were collected in two cases: hot cache, when the send and receive buffers are precharged in cache before invoking the communication system calls; cold cache, when the send and receive buffers are discharged from cache before invoking the communication system calls.

Our average estimates in the case of hot cache are:

- Latency: 18.4 μs
- Maximum Bandwidth: 9.91 MB/s, with messages sized 65536 bytes.

Instead, our estimates in the case of cold cache are:

- Latency: 28.7 μs
- Maximum Bandwidth: 9.83 MB/s, with messages sized 65536 bytes.

Figure 1 depicts the plot of throughput and bandwidth as a function of the message size in the case of

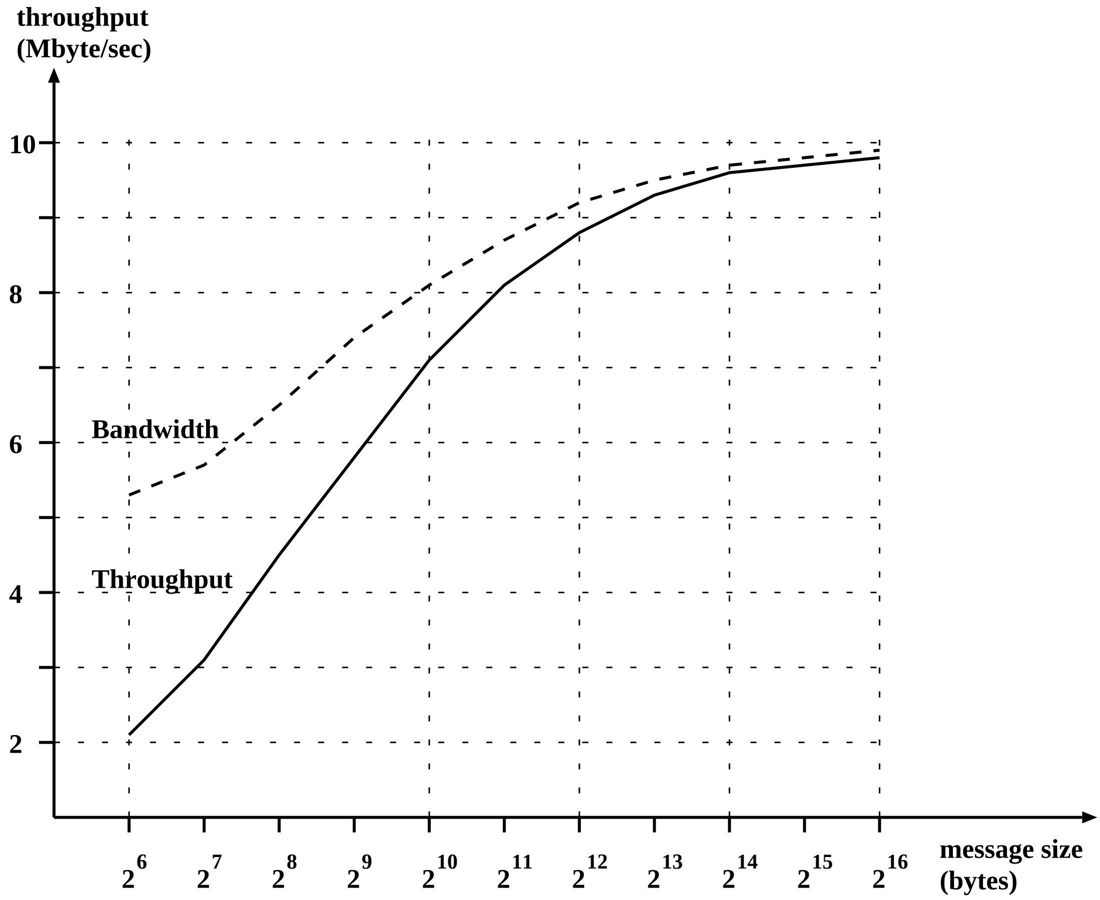

Figure1. Throughput and Bandwidth as a function of message size with hot cache.

Platform	Latency (μs)	Bandwidth (MByte/s)
100 Mb/s LAN, Linux TCP/IP sockets	275.6	3.4
100 Mb/s LAN, PARMA[2] PRP sockets	73.5	6.6
GAMMA cold cache	28.7	9.8
GAMMA hot cache	18.4	9.9
CM-5 CMMD	93.7	8.3
CM-5 CMAML	35.0	
SP2 MPL	44.8	34.9
T3D PVMFAST	30.0	25.1
U-net UAM ATM	35.5	14.8
U-net UAM Fast Ethernet	30.0	12.0

Table1. Ping-pong application: comparison of platforms.

hot cache. We observed that half the bandwidth is exploited already with messages as short as 32 bytes.

Using results provided by our collegues working on the PARMA^2 Project [3, Table 2.1] we can compare the performance of ping-pong on GAMMA with the performance of ping-pong on other platforms. Table 1 reports some of the results obtained by the PARMA^2 group in which the GAMMA results are integrated.

Like PARMA^2, GAMMA cannot compete with real parallel platforms in terms of bandwidth. However, in terms of pure message latency GAMMA provides a dramatic improvement compared to PARMA^2 and is quite competitive even compared to much more expensive parallel platforms. Of course comparing GAMMA performance to the performance of other platforms using MPI or CMMD is not completely fair since only an Active Message layer and not the whole MPI environment runs on GAMMA for the moment. However we do not expect the overhead of the MPI implementation over GAMMA to increase message latency in a substantial way. The comparisons with CM-5 CMAML and the U-net emulation of Active Messages [5] appear instead to be fair. In the case of U-net data are desumed from the cited paper and the hardware platform is constituted by 60 MHz SuperSPARCs connected by 140 MHz ATM.

5. Conclusions and Future Works

Our preliminary experimental results show an improvement of more than 4 times in short message delay as compared to PARMA^2 which in turn had already shown a substantial improvement (one order of magnitude) over the standard implementation of socket based inter-processor communication. These results suggest that with proper kernel support, MPI type communication primitives could show more than one order of magnitude improvement with respect to standard implementations on NOW platforms as currently provided by the best public domain products. Such improvement may suffice to make the use of inexpensive NOW platforms feasible and convenient with respect to expensive "commercial parallel platforms" for a large class of applications and with a fair number of processors.

Of course, our proposed NOW platform lacks one of the main requirements that a massively parallel architecture must consider, namely scalability with respect to the number of processing nodes. If we increase the number of processing nodes in GAMMA sooner or later we will saturate the LAN bandwidth. This problem can be partly alleviated by the adoption of a switched Fast Ethernet, which provides dedicated, full-duplex 100Mb/s connections to each node. The cost of Fast Ethernet switches is now decreasing, and it will soon become an affordable off-the-shelf component, in substitution of traditional repeater hubs.

On the other hand, we think that the cost of the scalability characteristics of "real" parallel platforms such as the CM-5 (with its Fat Tree interconnection network) is hardly justified in the majority of applications where parallel processing would, in principle, make sense. Only very few special applications would really require (and therefore really be worth the current cost of) the scalability characteristics of a "real" parallel platform that a special purpose network can provide with respect to off-the-shelf LAN hardware technology.

This consideration leaves space for a trade-off between the scalability of a parallel machine and the low cost and reasonably good performance of standard LAN hardware provided that the performance of the LAN hardware is not fully wasted by inefficient layers of software.

In conclusion, we think that the preliminary results of our experiment are quite encouraging and that it surely makes sense to propose the inclusion of a small set of communication primitives, implemented according to an accurate, performance-oriented approach into a standard operating system kernel.

References

[1] Connection Machine CM-5 Technical Summary. Technical report, Thinking Machines Corporation, Cambridge, Massachusetts, 1992.

[2] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet Computation. In *Proc. Supercomputing '95*, San Diego, California, 1995. ACM Press.

[3] The Computer Engineering Group. $PARMA^2$ Project: Parma PARallel MAchine. Technical report, Dip. Ingegneria dell'Informazione, University of Parma, Oct. 1995.

[4] The Message Passing Interface Forum. MPI: A Message Passing Interface Standard. Technical report, University of Tennessee, Knoxville, Tennessee, 1995.

[5] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proc. 15th ACM Symp. on Operating Systems Principles*, Copper Mountain, Colorado, Dec. 1995. ACM Press.

[6] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proc. 19th Int. Symp. on Computer Architecture*, Gold Coast, Australia, May 1992. ACM Press.

[7] M. Welsh, A. Basu, and T. von Eicken. Low-latency Communication over Fast Ethernet. In *Proc. Euro-Par'96*, Lyon, France, Aug. 1996.