GAMMA and MPI/GAMMA on Gigabit Ethernet

Giuseppe Ciaccio, Giovanni Chiola

DISI, Università di Genova
via Dodecaneso 35, 16146 Genova, Italy
E-mail: {ciaccio,chiola}@disi.unige.it

Abstract. The Genoa Active Message MAchine (GAMMA) is a lightweight communication system based on the Active Ports paradigm, originally designed for efficient implementation over low-cost Fast Ethernet interconnects. A very efficient porting of MPICH atop GAMMA as been recently completed, providing unprecedented messaging performance over the cheapest cluster computing technology currently available. In this paper we describe the recently completed porting of GAMMA to the GNIC-II Gigabit Ethernet adapters by Packet Engines. A combination of less than 10 μs latency and more than 93 MByte/s throughput demonstrates the possibility for Gigabit Ethernet and GAMMA to yield messaging performance comparable to the ones from many lightweight protocols running on Myrinet. This result is of interest, given the envisaged drop in cost of Gigabit Ethernet due to the forthcoming transition from fiber optic to UTP cabling and ever increasing mass market production of such standard interconnect.

1 Introduction

The low-cost processing power of clusters of Personal Computers (PCs) can be easily exploited by means of appropriate, high-level, standard Application Programming Interfaces such as, e.g., MPI. Several open-source implementations of MPI (e.g., MPICH [2]) can run on Linux-based Beowulf-type clusters on top of standard TCP/IP sockets. However it is well known that parallel jobs characterized by medium/fine grain parallelism exchange messages of small size (few KBytes) among processors, and that general-purpose protocols such as TCP/IP make very inefficient use of the interconnect for short messages.

On Fast Ethernet, a lightweight messaging system like the Genoa Active Message MAchine (GAMMA) [4, 7] provides far better performance at no additional hardware cost compared to the Linux TCP/IP stack. Started in 1996 as part of a Ph.D. thesis, the GAMMA project was targeted right from the beginning to low-cost Fast Ethernet interconnects. GAMMA implements a non-standard communication abstraction called *Active Ports*, derived from Active Messages, and provides best-effort as well as flow-controlled communication routines. With an end-to-end latency between 12 and 20 μs (depending on the hardware configuration and the NIC used), GAMMA provides adequate communication and synchronization primitives for tightly coupled, fine-grain parallel applications on inexpensive clusters. A complete yet efficient implementation of MPI atop GAMMA is now available [6, 8].

The link speed offered by Fast Ethernet is insufficient for many communication intensive parallel jobs to scale up. This justifies the interest towards the two most famous gigabit-per-second interconnects, namely, Myrinet and the more recent Gigabit Ethernet. The inefficiency of TCP/IP is exhacerbated here, since the physical transmission time becomes negligible compared to the time spent in the traversal of the protocol stack. Lightweight messaging systems become a key ingredient for an effective use of fast interconnects.

Currently, the per-port cost of a Gigabit Ethernet LAN is too high compared to Myrinet. This is largely due to the current fiber-optic cabling of Gigabit Ethernet. With the forthcoming transition of Gigabit Ethernet from fiber-optic to standard UTP copper cabling, however, we expect a substantial drop in cost which will push this LAN technology into a much larger segment of the marketplace, characterized by competition among different vendors and a potentially very large base of installation; eventually, the per-port cost of Gigabit Ethernet will become negligible compared to the cost of a single PC, in much the same way as it occurred with Fast Ethernet. In our opinion, Myrinet will never enjoy such a large diffusion: its segment of marketplace (system-area networks) will remain narrow compared to LANs.

Moreover, some Gigabit Ethernet NICs are programmable in much the same way as Myrinet is. For instance, the NetGear GA620 NIC, a cheap (300 US dollars) clone of the Alteon AceNIC adapter, comes with as many as two on-board microprocessors and 512 KBytes of on-board RAM, which makes it possible to upload appropriate, possibly self-made custom firmware to the NIC.

The only remaining difference between Myrinet and Gigabit Ethernet is the reliability of the physical medium, especially in case of network congestion which may cause packet losses. Myrinet prevents congestion by using hardware mechanisms for back-pressure, whose practical effectiveness has been demonstrated. Gigabit Ethernet uses hardware-exchanged control packets to block senders in case of congestion hazard, according to the IEEE 802.3x specification; this should in principle avoid congestion, and packet losses thereof, although nobody could assess the effectiveness of this mechanism so far. Another difference, of secondary concern though, is the higher communication latency of current Gigabit Ethernet switches (in the order of 3 to 4 μs), compared to the very low latency of a Myrinet switch (less than 1 μs).

To sum up, in our opinion Gigabit Ethernet is a promisingly successful and cheaper alternative to Myrinet under any respect, and an efficient lightweight protocol is definitely a must to make best use of this technology.

In order to prepare for the transition to inexpensive Gigabit Ethernet, we started developing a prototype of GAMMA for the Packet Engines GNIC-II Gigabit Ethernet adapter. The prototype was ready in September 1999; altough Packet Engines discontinued the NIC a few months later, that first prototype of GAMMA was indeed useful to experimentally demonstrate the feasibility and success of a lightweight communication protocol like GAMMA on next-generation inexpensive LANs. Overall, MPI/GAMMA on our GNIC-II Gigabit Ethernet NICs yields excellent performance figures. On two Pentium II 450 PCs networked by a Gigabit Ethernet switch, a MPI user application enjoys 16 μs end-to-end latency and 93.5 MByte/s peak throughput (77 % of the nominal link speed), comparable to if not better than many lightweight messaging systems running on Myrinet.

2 The DBDMA data transfer mode

A NIC is an interface between a host CPU and a network; as such, each NIC must implement suitable mechanisms to cooperate with the host computer, on one hand, and the network, on the other hand. A modern NIC cooperates with the host computer using a data transfer mode called *Descriptor-based DMA* (DBDMA). With the DBDMA mode, the NIC is able to autonomously set up and start DMA data transfers. To do so, the NIC scans two precomputed and static circular lists called *rings*, one for transmit and one for receive, both stored in host memory. Each entry of a ring is called a *DMA descriptor*.

A DMA descriptor in the transmit ring contains a pointer (a physical address) to a host memory region containing a *fragment* of an outgoing packet; therefore, an entire packet can be specified by chaining one or more send DMA descriptors, a feature called “gather”.

Similar to a descriptor in the transmit ring, a DMA descriptor in the receive ring contains a pointer (a physical address, again) to a host memory region where an incoming packet could be stored. The analogous of the “gather” feature of the transmit ring is here called “scatter”: more descriptors can be chained to specify a sequence of distinct memory areas, and an incoming packet could be scattered among them.

A NIC operating in DBDMA mode allows a greater degree of parallelism in the communication path, according to a producer/consumer behaviour. At the transmit side, while the NIC “consumes” DMA descriptors from the transmit ring operating the host-to-NIC data transfers specified in the descriptors, the CPU runs the protocol stack and “produces” the necessary DMA descriptors for subsequent data transfers. The reverse occurs at the receive side. Since both sides are decoupled from each other, the communication path works like a pipeline whenever traveled by a sequence of data packets, with a potentially high throughput.

However, the Linux implementation of IP forces the NIC device drivers *not* to exploit the “gather/scatter” features. This implies that at least two temporary copies of data are needed, one at the sender and the other at the receiver side, because header and payload of each packet are expected to be contiguous in the same memory area at both sides of communication when control reaches the device driver. A different organization and semantics of the communication protocol could eliminate the memory copy at the sender side by exploiting the “gather” feature: the header could be precomputed and stored somewhere in kernel space, the payload could be pointed to directly in user space, and the NIC would autonomously arrange them contiguous into its on-board transmit FIFO and send the whole packet. However, avoiding the memory copy at the receiver side is impossible: the final destination in user space for the payload of an incoming packet can be determined only *after* inspecting the header, which implies the packet be already stored somewhere, namely, into a temporary buffer. The only way to avoid a memory copy at the receiver side is to run the communication protocol by the NIC itself, and let it inspect the headers when packets are still in its on-board receive FIFO.

To sum up, a lightweight protocol for non-programmable NICs can be zero-copy on send thanks to the "gather" feature, but it must be "one-copy" on receive. Indeed, all the latest prototypes of GAMMA took this very organization, where most of the send overhead is in charge of the NIC and most of the receive overhead is in charge of the host CPU.

3 Maximizing the end-to-end throughput

The theoretical maximum throughput with Gigabit Ethernet is 125 MByte/s, roughly equivalent to the maximum throughput of the 32 bit, 33 MHz PCI bus where the NICs are plugged. However, a first prototype of GAMMA on Packet Engines GNIC-II adapters yielded a peak end-to-end throughput of only 80 MByte/s. We immediately started investigating the causes for such a low efficiency, using a pair of Pentium II 450 MHz PCs with 100 MHz system bus.

To begin with, we identified the following consecutive stages in the communication pipeline, from sender to receiver, in our opinion accounting for most of the total communication effort:

- stage 1: the "consumption" of DMA descriptors from the transmit ring and the corresponding DMA data transfers from the host RAM to the NIC, operated by the sender NIC;
- stage 2: the physical link;
- stage 3: the DMA transfers from the NIC to the host RAM and the corresponding "production" of descriptors in the receive ring, operated by the receiver NIC;
- stage 4: the "consumption" of DMA descriptors from the receive ring and the related protocol action and data movements, carried out by the receiver CPU.

Measuring the throughput of all the above pipe stages is not necessarily an easy task. Throughput of stage 1 can be measured directly, and throughput of stage 2 is known in advance (125 MByte/s). However, throughput of stage 3 cannot be easily measured by a direct technique. Moreover, we have to evaluate the effect of bus contention between stages 3 and 4 when both try to access the host RAM. Finally, we do not know in advance where the bottleneck is, and the presence of a bottleneck invalidates any direct throughput measurement taken below it in the pipeline.

The throughput of stage 1, directly measured, was 80.8 MByte/s, very close to the end-to-end throughput. This meant that stage 1 was the bottleneck, and also that we could not know much more about stages 3 and 4.

To proceed with our analysis we had to eliminate this bottleneck, even sacrificing the protocol correctness. We suspected that such a slow speed was caused by the "gather" feature: the sender NIC had to "consume" as many as two DMA descriptors for each outgoing packet. In order to prove this, we temporarily switched to a different transmission technique. The sender CPU "produces" only one descriptor for each packet, pointing to a dummy packet of appropriate size but containing only the header, without user data. This way the sender transmits wrong data, but this does not hurt performance measurements. We then observed a much higher throughput of 98 MByte/s for stage 1. The peak end-to-end throughput did not increase as much, though: it only measured 93.8 Mbyte/s, indicating that a second bottleneck was set up somewhere else. Throughput of stage 4, as directly measured, was about 100 MByte/s, therefore stage 3 was the bottleneck with its 93.8 MByte/s. By temporarily disabling the data movements in stage 4 we were able to isolate the performance degradation due to the contention between stages 3 and 4 on the memory bus; indeed, disabling stage 4 led to an increase of the stage 3 throughput from 93.8 to 96 MByte/s.

The lesson is now clear: if we want to maximize the end-to-end GAMMA throughput with the GNIC-II Gigabit Ethernet NICs, we should not use the "gather" feature of the NIC at the sender side. This however forces us to add another stage in the communication pipeline just where it begins, namely:

- stage 0: the copy of the packet payload from the user buffer to a preallocated memory buffer already containing the precomputed header, and the corresponding "production" of one single DMA descriptor into the transmit ring, carried out by the sender CPU.

Indeed we have implemented such an additional pipe stage, which exhibits a throughput of 95 MByte/s. Adding such a temporary copy on send was the paradoxical price we had to pay in order to maximize the end-to-end throughput of GAMMA with our Gigabit Ethernet adapter.

4 Improving throughput for short messages

Fragmenting a message into packets is a need when the total message size exceeds the maximum MTU size of the network. Fragmentation increases the CPU overhead for header processing, and leads to a lower utilization rate of the physical link. However, since the end-to-end communication path is a pipeline, fragmentation also leads to a much better throughput, by exploiting the parallelism among pipe stages. For this reason, message fragmentation has been exploited also with Myrinet, which is not packet oriented. Even with Fast Ethernet, it can be convenient to fragment a message even when it is smaller than the maximum allowed MTU size [5].

To improve the efficiency with short messages, GAMMA can fragment each short message into packets whose size is not necessarily maximal. Of course, this increases the number of packets exchanged. The optimal fragment size depends on the total message size and also depends on the performance profile of the communication hardware. Due to the lack of a satisfactory performance model of the whole communication system, we could only find an empyrical formula for optimal fragmentation, valid only for the GNIC-II adapters: messages up to 512 bytes are fragmented in packets of 128 bytes, messages from 513 to 1664 bytes are fragmented in packets of 256 bytes, messages from 1665 to 5148 bytes are fragmented in packets of 384 bytes, messages from 5149 to 11000 bytes are fragmented in packets of 768 bytes, messages from 11001 to 12000 bytes are fragmented in packets of 896 bytes, and larger messages are fragmented in packets of 1408 bytes.

5 Communication performance

The measurement platform was a pair of PCs, each with a single Pentium II 450 MHz CPU, 100 MHz system bus, and a Packet Engines GNIC-II NIC. The two machines were connected back-to-back by a fiber-optic cable.

Self-made "ping-pong" microbenchmarks have been run to measure the average end-to-end communication latency of GAMMA, MPI/GAMMA, Linux 2.2.13 TCP/IP sockets, and MPICH atop TCP/IP sockets. The measured one-way latency is 9.5 μs with GAMMA, 12.3 μs with MPI/GAMMA, 132.1 μs with TCP/IP sockets, and 320.9 μs with MPICH. Adding an Intel Express Gigabit Switch in between the two machines results increases the latency by 3.6 μs.

By the same "ping-pong" programs we can estimate the *end-to-end* throughput (not to be confused with the *transmission* throughput as measured by the usual one-way "stream" tests). Figure 1 reports the throughput curves of the following messaging systems: GAMMA using the optimal fragmentation scheme (curve 1), GAMMA with standard fragmentation (curve 3), MPI/GAMMA, based on the optimal fragmentation (curve 2), Linux 2.2.13 TCP/IP sockets (curve 4), and MPICH atop TCP/IP sockets (curve 5). The maximum MTU size with GAMMA is 1408 bytes (1388 bytes of payload plus 20 bytes of GAMMA header), which experimentally provided better performance on our platform (possibly due to a better cache line alignment).

We briefly sketch the main conclusions that can be drawn from the latency numbers and the throughput picture:

- The communication performance of MPI/GAMMA on a Gigabit Ethernet LAN based on Packet Engines GNIC-II adapters is comparable to, if not better than many lightweight implementations of MPI running on Myrinet, even taking into account the additional latency of a switch.
- The efficiency with short messages of Linux TCP/IP, and MPICH thereof, remains too low for running a tightly coupled, fine-grain parallel job on a commodity cluster.

- The performance improvement yielded by the optimal fragmentation technique with GAMMA short messages, as shown by comparing curve 1 against curve 3, is significant (for instance, a 64% improvement in throughput is obtained with 1388 byte messages).
- The performance degradation caused by stacking MPI atop GAMMA, as shown by comparing curve 1 against curve 2, is very modest; the same does not hold with MPI atop TCP/IP sockets (curve 5 is much lower than curve 4). This is obvious, given that the porting of MPICH atop GAMMA was made at the ADI level. The resulting MPI/GAMMA stack is thin compared to the standard MPICH/P4/TCP stack.
- Linux TCP/IP is not able to saturate Gigabit Ethernet: its peak throughput is only 42.4 MByte/s. The scenario is completely different with Fast Ethernet, where Linux TCP/IP is indeed able to almost completely saturate the physical link, thanks to recent improvements with device drivers and protocol.
- The maximum end-to-end throughput achieved by GAMMA is 93.8 MByte/s, and amounts to the measured throughput of the NIC on receive (stage 3 in the communication pipeline, see Section 3). This means that GAMMA is able to saturate the NIC, and that the NIC itself is the communication bottleneck. However, the obtained throughput is a very respectable result, given that the GNIC-II adapter does not support frames larger than 1526 bytes, which is a very small size given the high link speed (the Alteon AceNIC and its clones yield a slightly better throughput but only using a larger MTU size, a non-standard feature called "jumbo frames.").

6 Related work

Most existing efficient implementations of MPI for clusters presently run on Myrinet (e.g., [9, 10]). To the best of our knowledge, the only attempt of providing an efficient implementation of MPI for Fast Ethernet and Gigabit Ethernet, besides ours, is represented by MVICH [3], the VIA-based implementation of MPI developed in the framework of the M-VIA project [1]. However, little information is available yet on MVICH; the only available performance information is related to MVICH atop Giganet cLAN, an expensive "VIA inside" gigabit-per-second interconnect; MVICH on Giganet yields 13.5 μs latency and 97 MByte/s peak throughput, that is, just slightly better compared to MPI/GAMMA on Packet Engines GNIC-II adapters.

Acknowledgements

We gratefully acknowledge the work of Vincenzo Di Martino and Leonardo Valcamonici (CASPUR, Università "La Sapienza" di Roma), who contributed with useful discussions and helped setting up the cluster environment. We also thank the Istituto Nazionale di Fisica Nucleare (INFN) for sharing their own cluster hardware with our investigation project.

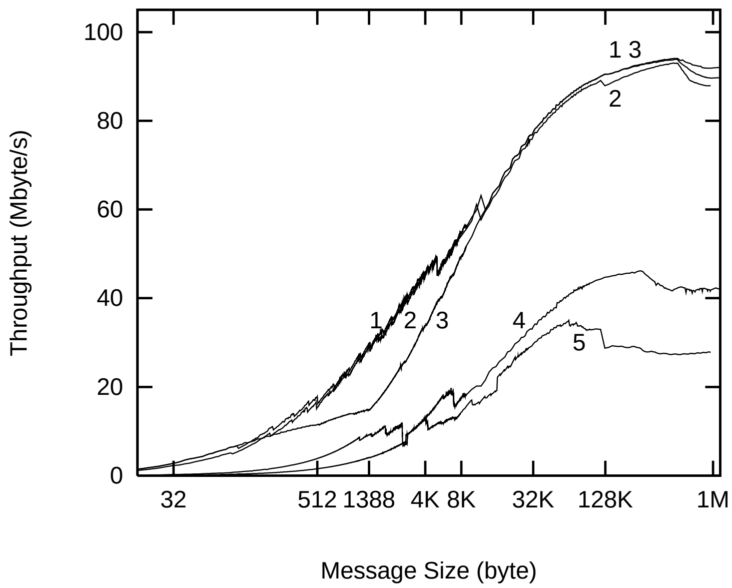

Fig. 1. End-to-end throughput of GAMMA, MPI/GAMMA, Linux 2.2.13 TCP/IP, and MPICH atop TCP/IP, using the GNIC-II Gigabit Ethernet adapters.

References

1. M-VIA Home Page, `http://www. nersc.gov /research /FTG /via/`, 1998.
2. MPICH - A Portable MPI Implementation, `http://www. mcs. anl. gov /mpi /mpich/`, 1998.
3. MVICH Home Page, `http://www. nersc.gov /research /FTG /mvich/`, 1998.
4. G. Chiola and G. Ciaccio. GAMMA home page, `http://www. disi.unige.it /project /gamma/`.
5. G. Chiola and G. Ciaccio. GAMMA: a low cost Network of Workstations based on Active Messages. In *Proc. Euromicro PDP'97*, London, UK, January 1997. IEEE Computer Society.
6. G. Chiola and G. Ciaccio. Porting MPICH ADI on GAMMA with Flow Control. In *Proc. IEEE - ACM 1999 Midwest Workshop on Parallel Processing (MWPP 1999)*, Kent, OH, August 1999.
7. G. Chiola, G. Ciaccio, L. Mancini, and P. Rotondo. GAMMA on DEC 2114x with Efficient Flow Control. In *Proc. 1999 International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'99)*, Las Vegas, Nevada, June 1999.
8. G. Ciaccio. MPI/GAMMA home page, `http://www. disi.unige.it /project /gamma/mpigamma/`.
9. M. Lauria and A. Chien. MPI-FM: High Performance MPI on Workstation Clusters. *Journal of Parallel and Distributed Computing*, 40(1):4–18, January 1997.
10. L. Prylli, B. Tourancheau, and R. Westrelin. The design for a high performance MPI implementation on the myrinet network. In *Proc. EuroPVM/MPI'99*, number 1697 in LNCS, pages 223–230, Barcelone, Spain, 1999.