GAMMA on DEC 2114x with Efficient Flow Control

Giovanni Chiola and Giuseppe Ciaccio
DISI, Università di Genova
via Dodecaneso 35, 16146 Genova, Italy

Luigi V. Mancini and Pierluigi Rotondo
DSI, Università La Sapienza
via Salaria 113, 00198 Roma, Italy

Abstract *GAMMA is a prototype light-weight communication system based on the Active Ports paradigm, designed for efficient implementation over Fast Ethernet interconnects. The original implementation started in 1996 based on 3Com 3C595 cards. The optimizations obtained on that NICs allowed us to obtain the lowest latency and highest throughput results ever published in the literature for Fast Ethernet. Technology evolved, however, and now all low-cost NICs available are based on Descriptor Based DMA (DBDMA) transfers, originally introduced by the DEC chipset 21140. In this paper we report on the re-implementation of the GAMMA prototype for the DEC NICs exploiting the new transfer modes and achieving substantially equivalent performance figures. We also describe the addition of an efficient flow-control algorithm, that allows loss-free communications without seriously affecting performance.*

Keywords: Active Ports; Fast Ethernet; Low Latency; Descriptor Based DMA; Flow-control.

1 Introduction

Linux provides one of the most efficient implementations of the TCP/IP stack ever. The Linux TCP/IP socket one-way latency depends heavily on the specific network driver being used, which in turn depends on what Network Interface Card (NIC) has been leveraged. One may wonder why bothering about the adoption of a specialized messaging system for cluster computing, and not just adopt standard TCP/IP communication on good NICs, as done in most Beowulf-type clusters.

Let us consider a very basic cluster configuration comprising two Pentium II 300 MHz PCs, each running Linux 2.0.29 and equipped with a 3COM 3c905 Fast Ethernet NIC. Suppose we connect the two PCs by a UTP class 5 crossover cable and work in half-duplex mode (this simulates the use of a Fast Ethernet repeater hub). Then run a simple ping-pong test to evaluate one-way latency and asymptotic bandwidth (after disabling the Nagle "piggybacking" algorithm). Under such experimental conditions you should measure a one-way latency of 77.4 μs and asymptotic bandwidth of 10.8 MByte/s at socket level, on average. The average half-power point is found at 1750 bytes. Recalling that the maximum theoretical bandwidth of Fast Ethernet is 12.5 MByte/s and assuming a reasonable lower bound for latency of 7 μs, this means that with Linux TCP/IP and Fast Ethernet: Latency is one order of magnitude worse than the hardware latency; Efficiency in the range of short (single packet) messages is below 50%; Efficiency is good (86%) only with very long data streams. Linux TCP/IP is very good for traditional networking but not perfect for cluster computing.

The Genoa Active Message MAchine (GAMMA) [3, 5, 2] is a commodity cluster based on Personal Computer (PC) and 100 Mbit/s Fast Ethernet technology. The Linux kernel has been enhanced with a communication layer implemented as a small set of additional light-weight system calls and a custom NIC driver with a fast interrupt path. Most of the communication layer is thus embedded in the Linux kernel, the remaining part being placed in a user-level programming library.

The adoption of an Active Message-like communication abstraction [7] called *Active*

Ports [4] allowed a zero-copy optimistic protocol, with no need of either kernel-level or application-level temporary storage for incoming as well as outgoing messages. GAMMA implements pipelined communication paths among user processes. Multi-user protected access to the communication abstraction is granted. The GAMMA device driver is capable of managing both GAMMA and IP communication in the same 100base-T network. On 3COM 3c595 and 3c905 NICs, GAMMA yields very low one-way user-to-user latency (12.7 μs) and high asymptotic bandwidth (12.2 MByte/s, corresponding to 98% efficiency) even when run on Pentium 133 based PCs.

It must be said that the GAMMA communication protocol offers little more than Datagram quality of service (QoS): it detects communication errors (packet losses and corrupted packets) but then it simply raises an error condition without recovering. The GAMMA approach leaves to the user (application as well as library writer) the task of using the error detection mechanisms to build recovery policies of suitable complexity. However the very low latency delivered by GAMMA potentially allows a wide range of error recovery as well as explicit acknowledge policies to be implemented in a very efficient way.

The main practical problem that has prevented a distribution of the GAMMA prototype in form of a Beta release so far is due to the fact that the 3Com NICs for which the GAMMA driver was originally developed are not commercially available anymore. Newer NICs distributed by 3Com adopt now an interaction mechanism that is similar to the one offered by other product (such as the Digital 2114x, the Intel EtherExpress, etc.) that is called *descriptor-based DMA* (DBDMA). The exploitation of this data transfer mode from NIC to RAM required a substantial re-design of the 3COM 3c595 based GAMMA prototype. In particular, the "true zero-copy" message receive that was implemented in the original GAMMA prototype is almost impossible to implement with similar performance results in case of DBDMA NICs.

In this paper we report on the re-design and porting of GAMMA on recent DBDMA based NICs, and on how we achieved pretty satisfactory results compared not only to previous GAMMA versions but also to other prototypes described in the literature that adopt the same chipset. The restructuring of the GAMMA prototype also changed the constraints on message queueing and allowed an effective introduction of a flow-control protocol.

The DBDMA version of GAMMA with flow-control (that guarantees extremely low probability of message loss in any load conditions) incurs no performance penalty compared to the DBDMA version of GAMMA without flow-control. This result allows the use of flow-control for increased reliability of the prototype and, in our opinion, compensates the (modest) loss in absolute performance with respect to the non-DBDMA previous prototype.

2 On DBDMA transfers

Figure 1 provides a comparison among throughput curves measured with different system configurations for a Linux PC cluster based on 3Com 3c905 Fast Ethernet NICs. It is apparent that using the same hardware and OS but different device drivers leads to substantial performance differences. The reason is very simple. The 3c905 NIC can be programmed and driven in two different ways, that we call *descriptor-based DMA* and *CPU-driven DMA*.

When driven in DBDMA mode, the NIC itself starts DMA transfers between host memory and the network (and vice-versa) by simply scanning a precomputed list of so called "DMA descriptors" stored in host memory. Therefore the low-level memory-to-network and network-to-memory data transfers are operated by the NIC autonomously while the CPU is running the TCP/IP protocol code and preparing the necessary DMA descriptors for subsequent data transfers. This operation mode pipelines the end-to-end communication path and increases the communication throughput.

The latest Linux drivers for the 3COM 3c905

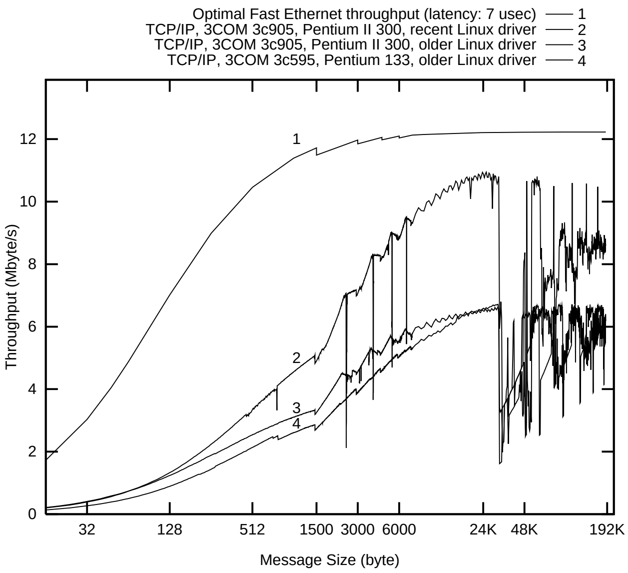

Figure 1: Linux 2.0.29 TCP/IP sockets: "ping-pong" throughput, various NICs and CPUs

NIC use the DBDMA mode. Previous drivers used the CPU-driven DMA mode, according to which the host CPU itself starts a DMA operation of the NIC. After a DMA transfer has been started, the CPU has to wait for the current DMA transfer to end before starting a subsequent DMA operation. This leads to a "store-and-forward" type of behavior and tight synchronization between the CPU and the NIC, that inevitably results either in lower throughput (if the CPU context-switches) or in higher overhead (if the CPU busy-waits).

A very similar 3Com card operated in CPU-driven DMA mode on Pentium 133 CPUs yields even lower performance. This provides an insight on the software overhead involved in running the TCP/IP stack: the larger this overhead, the larger the performance degradation with slower CPUs.

In summary, we may conclude that the advantages of use of DBDMA NICs for traditional communication protocols is quite evident. Such an advantage certainly justifies the widespread adoption of DBDMA NICs and their substitution of CPU-driven DMA NICs in the market of commodity components.

The theoretical question that remains open is whether light-weight cluster protocols such as GAMMA can benefit of DBDMA NICs or not. Thinking in terms of pure latency and bandwidth, the answer would obviously be "no." On the one hand, the GAMMA prototype already provides very high bandwidth even with CPU-driven DMA transfers. On the other hand, CPU-driven DMA transfers in the receiver side allow "true zero-copy" protocol implementation, reducing latency as compared to DBDMA. Indeed the demultiplexing of incoming messages to different receiver processes and software communication ports requires CPU activity, and calling the CPU after the incoming messages have been buffered in memory forces a subsequent copy to deliver the messages in their correct final destination.

In practice, even if one thinks that this might not be a good choice in terms of latency, the adoption of DBDMA NICs is forced by the non availability of CPU-driven DMA NICs. This was of course a practical consideration that could not be considered satisfactory from a scientific point of view, and the challenge was that of reducing the expected loss in terms of latency to a minimum.

In the re-design of the GAMMA prototype for DBDMA NICs we thus attempted to include additional functionalities that could solve part of the open problems inherent to the CPU-driven DMA version of the GAMMA driver. The main benefit that we could think to exploit in the case of GAMMA of a DBDMA approach was the availability of substantially larger FIFO queues (because they are implemented in system RAM rather than in the NIC). We thought that a good way of exploiting substantially larger input FIFO queues could have been the introduction of a credit-based, sliding window, flow control protocol similar to the one already adopted in the FM2 prototype developed at University of Illinois [6]. Such a flow-control protocol would have had little chances to be implemented in an efficient way in the context of inexpensive, CPU-driven DMA NICs due to the presence of few KBytes of on-board FIFO queues.

3 The flow-control problem

Another consideration which does not emerge from Figure 1 is related to performance in a congested LAN. It is well known that TCP uses a sliding-window flow control algorithm with "go-back-N" packet retransmission upon timeout. The flow control mechanism of TCP is an end-to-end protocol, that avoids packet overflow at the receiver side. However it cannot prevent overflow from occurring in a LAN switch in case of network congestion. Only a data-link level flow-control mechanism, such as, e.g., the one defined in the 802.3x standard extension, can prevent packet loss within the network in case of congestion. A switch that does not offer 802.3x flow-control can indeed discard frames in case of congestion. When this occurs, eventually the retransmission mechanisms of TCP on the sender hosts are triggered and start re-sending many more packets than needed, increasing network traffic and therefore making the LAN even more congested.

Clearly, LANs require better flow control algorithms or at least more sophisticated retransmission policies. In the case of GAMMA we started designing an appropriate flow-control protocol from scratch, taking into account the almost zero error rate that is exhibited by 100BaseT star topologies adopting standard class 5 UTP cabling. The idea of associating acknowledgement packets to each data packet was therefore discarded.

A pure credit-based sliding window protocol was instead chosen for pure flow control purposes, similar to the one already adopted in FM2 [6]. The availability of a potentially very large FIFO Input Queue for incoming messages, that is implemented in system RAM and handled by the NIC working in DBDMA mode, allows a splitting of the queue size among many potential message senders. Consider, e.g., the case of an input queue made up of 1024 buffers, each one of size 1.5 KBytes, in order to be able to store a complete Ethernet frame. Such an Input Queue would require the reservation of 1.5 MBytes in system RAM, which is not a big deal in a modern PC. A similar Input Queue cannot be included on-board in low-cost commodity NICs. Now consider the case of the interconnection of 100 nodes by means of a large switch. One may easily think at a "reservation" of a substantial portion of the Input Queue for point-to-point communications. Each node will have to be ready to receive messages from 99 different nodes. If we reserved 8 buffers for each possible sender, we would consume 792 out of the 1024 available buffers. In this system configuration we could then adopt a sliding window size of 8 for each communication pair. Each sender would be initialized with a credit of 8 frames towards each possible receiver, so that up to 8 consecutive frames could be sent towards a given arbitrary destination without receiving any acknowledgement. Once all credits towards a given destination are exhausted the sender must refrain from sending further frames towards that address until it receives an acknowledgement. An acknowledgement packet could restore the whole initial credit of 8 consecutive frames allowed for sending to a given destination.

In our example of utilization schema, additional 99 buffers in the input queue should be reserved for incoming acknowledgements in order to guarantee absence of deadlock. Out of the 1024 buffers available in the Input Queue, 133 would not be reserved for GAMMA communications, and could thus be used for other protocols (such as, e.g., TCP/IP) possibly sharing the same connection. The expected overhead of such a flow control protocol is expected to be almost negligible if the initial credit hold by a sender towards any possible destination is large enough, because a single acknowledgement frame is used to restore an entire credit and because acknowledgements do not involve any packet re-transmission in our pure flow-control mechanism.

4 The DBDMA GAMMA

The new GAMMA prototype based on DBDMA operating mode has been developed at the Department of Computer Science of the

University of Rome as part of two Master's thesis. One version was developed for Intel EtherExpress cards and another for Digital 21140 chipset based NICs. A similar effort has not been undertaken yet for 3Com NICs due to the lack of sufficient documentation on the programming interface of the cards. The DEC 21140 GAMMA prototype has been then ported to newer chipsets of the same family (21143), debugged and optimized at DISI, University of Genoa. A very effective flow-control protocol has been included in this version.

For message sending the mechanism implemented in the DBDMA driver is conceptually identical to the one implemented for the CPU-driven DMA ones. An explicit `gamma_send(out_port,data_pointer, length)` function is invoked by the sender process that specifies a memory address and an integer length to describe the data to be transmitted through a specified output port. The send primitive automatically fragments the message in packets of appropriate maximum size, and by means of a system call that is added to the usual Linux ones inserts the pointers to the fragmented data to be transmitted into a pre-allocated list of frame descriptors linked to the Output FIFO Queue associated with the NIC. The NIC automatically realizes that some frames in its output queue are ready to be sent, and starts sending them accessing to the data in RAM. Both blocking and non-blocking versions of `gamma_send()` are available, the former busy waiting until the last data frame has been sent, the latter returning immediately upon insertion of the pointer to the last frame to send in the output queue.

For message receive, instead, the adoption of the DBDMA transfer mode implied a different software architecture as compared to previous GAMMA prototypes. In the CPU-driven DMA version of GAMMA, incoming frames used to be kept in the on-board NIC's FIFO Input Queue until the CPU responded to the NIC's interrupt request. Then the CPU used to look at the content of the frame header by accessing it in programmed I/O (PIO) mode in order to distinguish GAMMA frames from TCP/IP frames, and do the appropriate demultiplexing. Eventually, the CPU programmed the DMA Bus Mastering transfer from the NIC's Input Queue to the final destination in RAM for the received data. Upon completion of the DMA transfer, the CPU executed the corresponding Receiver Handler.

With the DBDMA handling of the receiver queue, the FIFO Input Queue is allocated in system RAM rather than on-board. When the NIC starts receiving a frame, it immediately starts filling the next buffer available in RAM. Upon complete receipt of a new frame, the NIC interrupts the CPU to signal the availability of new data. The CPU may also poll directly the Input Queue data structure in RAM in order to discover the presence of new frames without having to wait for the execution of the Interrupt Handler. In any case, when the CPU reads the frame header and is ready to perform the demultiplexing, the message is already in RAM, but most probably at the wrong memory address. A memory-to-memory copy is usually needed (and implemented by the CPU) in order to deliver a GAMMA message in the user buffer specified by the receiver process. Only after this copy the CPU may execute the corresponding Receiver Handler.

The prototype adopting DBDMA NIC-to-RAM communication mode is therefore expected to provide less good performance than the previous CPU-driven DMA version, both in terms of latency and of CPU overhead. The greater CPU overhead in principle could be considered relatively harmless, as the increase in both CPU and memory bus clock rate are expected to mitigate the performance penalty in the near future. Our main concern before the development of the prototype was related to latency. In order to compensate for the expected performance loss we implemented an important feature that we did not know how to implement efficiently in the previous version of GAMMA, namely the flow-control. Exploiting the availability of a potentially very large FIFO Input Queue in RAM, we adopted a credit-based sliding-window protocol following the idea outlined in the previous Section.

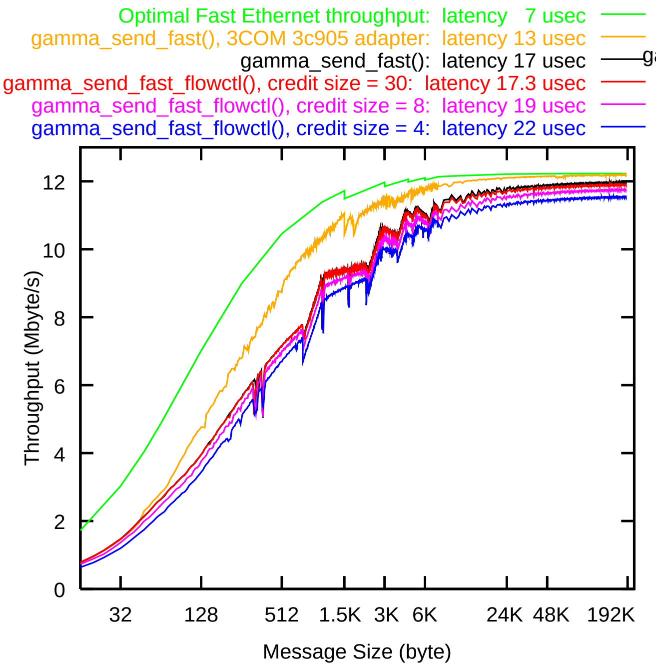

Figure 2: Throughput curves of the new GAMMA driver on DE500 NICs, Pentium 133

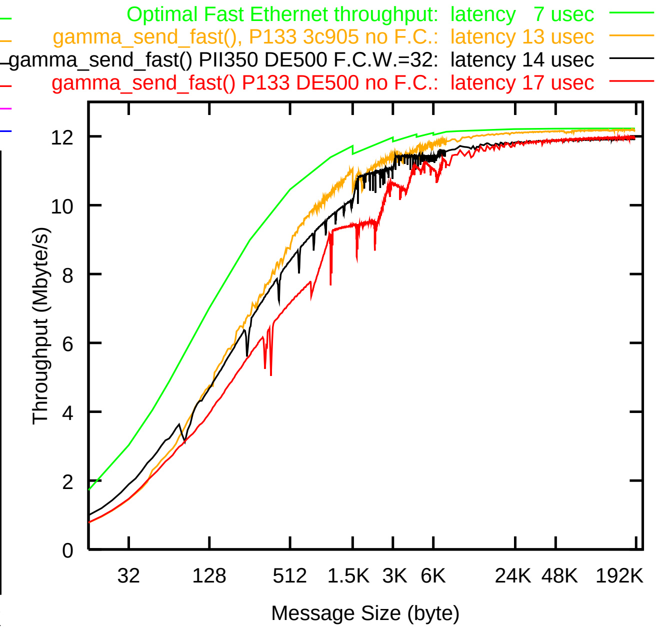

Figure 3: Throughput curves of the GAMMA driver on DE500 NICs, Pentium II 350

5 Performance

Figure 2 reports the throughput curves that we were able to measure on our old prototype GAMMA cluster (Pentium 133MHz) using the newly developed DBDMA version of the NIC driver for the DE500-BA NIC (21143 chipset). The measurements were taken using a Ping-pong experiment user-process to user-process level (including receiver handler overhead). Curves measured for the new driver implementation with and without flow-control protocol are compared to the maximum theoretical throughput computed based on pure hardware optimistic estimation of message delay and to the actual curve previously measured on the CPU-driven DMA version implemented for the 3C905 NIC. It is possible, of course, to appreciate the increase from 13 μs. to 17 μs. of message latency, even without flow-control. The throughput for small/medium size messages is also lower than the previous one, most probably due to the additional memory-to-memory copy on receive. It is interesting, however, to observe the diminishing impact of flow control on throughput and latency for increasing window size. While a window size of 4 frames still substantially affects the throughput curve and latency, a window size of 8 already has limited impact both on throughput and on latency. With a window size of a few tens, flow-control hardly reduces performance in a noticeable way.

Moreover, consider the throughput curve measured on our new prototype GAMMA cluster, which is based on Pentium II 350MHz with 100MHz memory bus, which is depicted in Figure 3. Firstly, notice the reduction in latency to 14 μs., which is almost equivalent to the latency of the previous CPU-driven DMA GAMMA driver for 3c905 NICs. Notice that the previous GAMMA driver obtained very little benefits from the upgrade of the processing nodes to faster CPUs. Indeed the throughput curve of the previous GAMMA prototype for 3c905 NICs was virtually insensitive to CPU clock rate. Secondly, notice the substantial improvement in the throughput curve of the DE500 driver obtained by simply increasing

CPU and memory bus clock rate (compare the two curves on P133 and PII350). The comparison clearly shows the diminishing overhead of memory-to-memory copy in receive due to the increase in CPU and memory bus clock rate. Thirdly, notice the very reduced difference in throughput curve between the old CPU-driven DMA with zero copy approach, and the new DBDMA with single copy on receipt if a modern CPU with high clock rate is used.

Finally, let us stress the absolute value of the performance results we obtained. GAMMA latency remains substantially lower than the latency reported for similar research projects, including U-Net [8] and M-VIA [1] on NICs using the same DEC chipset, even though our new GAMMA prototype now includes flow-control.

6 Conclusions

We have ported our Active Port based messaging system to a new device that adopts the DBDMA transfer mode to system memory. A redesign of the receive mechanism was required, that forced the introduction of one copy from memory to memory. Performance results show that, also due to the current trend of increasing speed of CPU and RAM even in extremely low cost PCs, such a change in the receive mechanism has little impact.

On the other hand, the availability of a substantially larger FIFO Input Queue (allocated in system RAM) allowed us to introduce a credit-based flow-control protocol with appropriately high window size reserved to all potential communication pairs. Our performance measurements also show that, using such protocol and the window sizes that are feasible in our low-cost prototype, the impact of flow-control on latency and throughput is negligible.

With the adoption of flow-control our prototype becomes very reliable, avoiding all sorts of message loss due to receiver buffer overflow under any load condition of the computational nodes. The only remaining source of message loss in GAMMA with flow-control is due to transmission error and subsequent CRC check failure, which almost never occurs (assuming 10^{-10} bit error rate, an error inducing frame loss is expected every 10^6 frames). In particular, using the new GAMMA prototype with flow-control we expect to be able to implement an MPI interface on top of GAMMA offering unprecedented performance on cheapest Fast Ethernet technology.

References

[1] M-VIA URL `http://www.nersc.gov/research/FTG/via/`, 1998.

[2] G. Chiola and G. Ciaccio. GAMMA home page, `http://www. disi.unige.it/project /gamma/`.

[3] G. Chiola and G. Ciaccio. Implementing a Low Cost, Low Latency Parallel Platform. *Parallel Computing*, (22):1703–1717, 1997.

[4] G. Chiola and G. Ciaccio. Active Ports: A Performance-oriented Operating System Support to Fast LAN Communications. In *Proc. Euro-Par'98*, LNCS 1470, pp.620–624, Southampton, UK, September 1998.

[5] G. Ciaccio. Optimal Communication Performance on Fast Ethernet with GAMMA. In *Proc. Workshop PC-NOW, IPPS/SPDP'98*, LNCS 1388, pp.534–548, Orlando, Florida, April 1998. Springer.

[6] M. Lauria, S. Pakin, and A. Chien. Efficient Layering for High Speed Communication: Fast Messages 2.x. In *Proc. 17th IEEE Int'l Symp. HPDC-7*, Chigago, Illinois, July 1998.

[7] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proc. ISCA'92*, Gold Coast, Australia, May 1992. ACM Press.

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