

BIP: a new protocol designed for high performance networking on Myrinet

Loic Prylli¹ and Bernard Tourancheau²

¹ LIP, ENS-Lyon
69364 Lyon — France

`Loic.Prylli@ens-lyon.fr`

² LHPC & INRIA ReMaP^{***} — LIGIM bat710, UCB-Lyon
69622 Villeurbanne — France
`bernard.tourancheau@inria.fr`

Abstract. High speed networks are now providing incredible performances. Software evolution is slow and the old protocol stacks are no longer adequate for these kind of communication speed. When bandwidth increases, the latency should decrease as much in order to keep the system balance. With the current network technology, the main bottleneck is most of the time the software that makes the interface between the hardware and the user.

We designed and implemented new protocols of transmission targeted to parallel computing that squeeze the most out of the high speed Myrinet network, without wasting time in system calls or memory copies, giving all the speed to the applications.

This design is presented here as well as experimental results that lead to achieve real Gigabit/s throughput and less than $5\mu s$ latency on a cluster of PC workstations, with this affordable network hardware. Moreover, our networking results compare favorably with the expensive parallel computers or ATM LANs.

1 Introduction

Multimedia application as well as parallel computing and databases are asking for low latency high bandwidth networks. This kind of performance implies a new design of the protocols in order to avoid software latency and memory copies.

Indeed, the recent relative evolution of computer subsystems has created new problems: five years ago, with parallel computing over 10Mbits/s Ethernet or even on a 100Mbits/s FDDI ring, it was easy to saturate the network. The memory bandwidth or the IO bandwidth of typical workstations were an order of magnitude faster than the physical network, so the interface between the user and the hardware was not too much a problem. Nowadays relatively inexpensive network technology such as Myrinet is over 1Gbits/s for LAN, and

^{***} This work was supported by EUREKA contract EUROTOPS, LHPC (Matra MSI, CNRS, ENS-Lyon, INRIA, Région Rhône-Alpes), INRIA Rhône-Alpes project REMAP, CNRS PICS program, CEE KIT contract

although workstations have also increased in performance, the gap between network bandwidth and other inner bandwidths (memory and IO busses) has been considerably reduced. So it is time to use carefully host resources.

Our experiences with ATM LAN networks have shown two problems, first even when the wire is able to provide 155Mbit/s, a poor design of the ATM board drivers can prevent the use of more than half the hardware bandwidth. Second the latency on typical workstations is counted by hundreds of microseconds [Pry96,PT97] which is unbearable in such a context (a 500 μ s latency is equivalent to the transfer of 10Kbytes of data at the speed of 155Mbit/s).

Our software research work was driven by the idea that we wanted to exploit to its full strength the potential of the network in the applications. In the real world, what counts is not what the hardware can theoretically support (ATM 155Mbit/s, Myrinet 1.28Gbits/s) but what performance is available at the user/developer level (and which marketing will not advertise). Our research shows that the power of high speed networks can be exploited by carefully shortening the path of data from application to application and minimizing all overheads. This is necessary for both latency and bandwidth improvement.

This paper describes our ideas for the design of a software protocol that led to a sustained Gbits/s throughput with less than 5 μ s latency over a Myricom LAN of PCs.

2 Our PCs - Myrinet LAN platform

The Myricom LAN[BCF⁺95] target was chosen for its performance over the Gbits/s (OC-24 = 1.28 Gbits/s actually), its very affordable price and its software openness (all the software and specifications are freely available for customers). The results of this paper are obtained on PCs with PentiumPro200Mhz (with 256Kb cache) running Linux 2.0 with Myrinet PCI boards. These boards are based on a LANAI processor with 256Kbyte SRAM.

3 Overview of BIP

Our first objective was to implement BIP, an interface for network communication targeted towards message-passing parallel computing. The idea in BIP was to provide protocols with low level functionalities. Specialized parallel applications could interface directly with it. But the main usage will probably be through other protocol layers, like IP or APIs like the well established MPI[SOHL⁺95] and PVM[GBD⁺94] (see Figure 1). We actually already provide an MPI implementation on top of BIP.

BIP was the abbreviation for Basic Interface for Parallelism. Our basic idea was to build it with a library interface accessible from applications that will implement a high speed protocol with the fewest accesses to the system kernel (for other work in this area, see Section 6).

The BIP interface provides several functions to get parameters or set-up constants and the send and receive blocking or non-blocking communication

primitives. There is two distinct semantics for BIP communications depending on the message size:

- “long messages” sends and receives have a rendez-vous semantic where a receive need to be posted before the send. This is a requirement similar to the “ready send” mode of MPI.
- On the opposite “short” messages are stored into an circular queue on the destination so that the send calls will not block even if no matching receive has been posted, (except if the destination receive buffers becomes completely full, then the matching send will block). There is queuing for small messages. They have the “standard mode” MPI semantics.
- The limit between “short” and “long” messages is specified by a constant BIPSMALLSIZE (and depending on the release is between 100 and 400 bytes).

BIP messages can be routed through multiple Myrinet switches. With Myrinet, it is the sender that provides the route through the switches. At initialization, BIP build a table and choose (or allow the user to choose) a static route for each pair of processors.

The current BIP control flow relies on the hardware flow-control. In some contexts, it is not sufficient, for instance when one side is not able to receive for a long time (because in this case “hardware control flow” would comes into play, a message blocked on the network could cause some congestion, besides there are real-time constraints about how long a message can stay blocked on the network). MPI-BIP is doing a credit-based flow control taking into account the sizes of the BIP queues that exists for small messages, to ensure all messages send can be received by the destination node.

BIP messages can be tagged for identification. The send other arguments are the data and the logical number of the destination, the receive does not specify a particular source but can check on a tag, its other arguments are a buffer where to receive a message and the maximal length it can receive in this buffer.

The send and the receive are also available in a non-blocking semantics, where one can either test or wait for the completion of the asynchronous send and receive calls. With the actual version, at one time, a process may only have one receive per tag and one send posted, and not more. Send and receive operations are completely independent so you can intermix them in any manner. The non-blocking primitives allow overlapping of computation and communication when appropriate but there must be no more than one send and one receive operation per tag pending.

It is up to the application or upper protocol layers to provide more functionality if needed. The rational here is that BIP is intended as a intermediate layer for other higher level protocols as soon as a complex functionality is needed. But the services provided are strongly oriented to the parallel application demand.

If there is no error on the communication medium, BIP ensures reliable delivery of messages. Still BIP adds sequence numbers and checks CRC so that any error or loss of message on the medium will be detected (but not recovered).A more complete presentation is done in the BIP manual[Pry97].

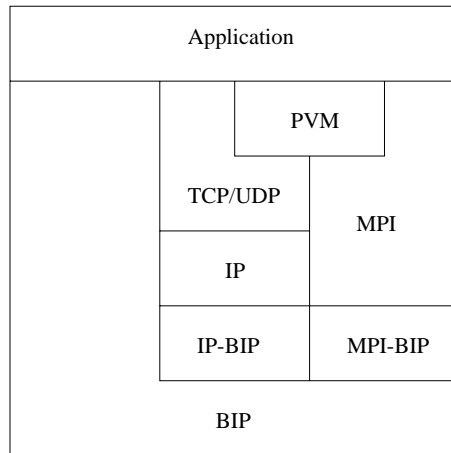


Fig. 1. Description of the protocol stack with our application performance point of view: the application can access directly the BIP level (basic message passing interface with semantic constraints) or use other functionality levels. Notice that this can be changed depending on the implementation that are done, for instance one can imagine to port the BLACS, or TCP, or PVM directly on top of BIP in order to keep very low latency.

4 Design choices and methodology

The software design was in particular guided by the high speed network platform we use: Myrinet [Myr95,BCF⁺95]. However, the general ideas are applicable to any network hardware architecture that provide the same functionalities: processor and memory on the network board to implement our BIP firmware, DMAs to do the pipeline transfers.

4.1 User level management of the network

The important point here has already been pointed out by others [PKC97,BBVvE95]: for performance reasons, we cannot afford using the OS and a heavy protocol stack as an intermediary to access the network hardware. BIP message-passing library directly manages the hardware and the BIP firmware to implement the message-passing API at the user-process level.

4.2 Zero memory copy

Five years ago, on the parallel machines, or on cluster of workstations with 10Mbits/s Ethernet, the memory bus was one or several order of magnitudes faster than the network. There was not too much concern on the way to put data from the memory to the network. Copies of communication buffers in a

memory space suited for the communication protocol, packet disassembly and reassembly, all this kind of operations did not have an important impact on the final performances.

The current situation is quite different, the network bandwidth is often comparable with the memory bandwidth : for instance with our configuration the memory bus throughput is 180Mbytes/s when reading memory, a memory copy is about 80Mbytes/s, and the network bandwidth is 160Mbytes/s. So the alternative between doing a memory copy versus putting directly user data on the network impact performance, as much as the old-fashioned store and forward strategy slow down routing, compared to circuit-switched or worm-hole strategies. In our case, each memory copy is exactly like having a store-and-forward step on the network.

BIP is designed for zero memory copy transfers.

Hardware requirements Avoiding memory copy put some requirements on the type of hardware. Not all machines are in fact capable of dealing with this situation efficiently.

For most high speed network hardwares, at some point there is a DMA from memory either directly to the physical network or to the SRAM memory on the interface boards (Myrinet is in the latter case). Data move in the opposite direction is symmetrical.

First to transfer data directly from user data space, the DMA engine should be able to address without overhead any location in main memory (which is not the case on all architectures, or requires an initialization with non negligible cost).

Then, all processors nowadays have memory caches, and it will occur frequently that the valid user data is in a cache and has not yet been updated in main memory. The system bus of the machine should ensure that coherency is respected between processor cache and main memory, when a DMA memory access is done[BS95]. That means the processor should do snooping on the system bus and provide data from its cache when it is more recent instead of leaving the memory provide it¹. This requirement is full-filled on most machines based on Pentium, Pentium Pro, SuperSparc, UltraSparc, . . . , but MicroSPARC have not this capabilities.

Software and MMU issues The second problem for a direct transfer of user data from/to the network is the way the hardware and the OS deals with the duality between physical/virtual addresses. Basic OS system considerations make the use of pagination unavoidable for common applications. Moreover a “contiguous” chunk of data in virtual address space will not be contiguous in physical memory. Before a DMA transfer, we have first to ensure the buffer area is locked

¹ This mechanism is necessary when you have several processors sharing the main memory, so in general all processors/architecture that have multiprocessing capabilities deals transparently with this problem.

into physical memory, and then for each page of this area, we need to obtain the physical address.

Our experimental test-bed consisted of PCs where the PCI bus can directly see the physical memory without translation². So before a transfer of user data, the user area is converted into a list of pairs (virtual address,block length).

This is done by a kernel module that is part of the BIP implementation. This module first provides to the user level some system functions calls to pin pages in physical memory (mark them unswappable)³. Second, it allows the conversion of a virtual address into a physical memory address (after checking its page has been marked unswappable, if not the physical address could be changed or become invalid).

The pinning of pages into memory may requires a system call, but this cost represent a small overhead for the following reasons:

- It is called at most once per page of the process (the first time a communication buffer is accessed in the page). Indeed we keep the page locked and remember the translation as long as there is enough memory.
- For long messages we have measured a throughput of 100Mbytes/s when no page is locked before any communication call, to be compared with the maximum of 125Mbytes/s in the best case.
- For small messages this mechanism is not used at all (see section 4.4.

We assume a dedicated computing platform, so we do not try a sophisticated allocation algorithm for memory pinning. In a "multi-user" usage, managing memory in a fair manner is a non trivial issue, addressed for instance in [AB97].

No sharing between applications Like in the design of most parallel machines, the BIP implementation monopolizes the network interface of one node for one application. Indeed, in the actual version, there cannot be several independent BIP applications using the same processor, or the network interface cannot be used for IP traffic while the BIP application is running.

BIP is also not intended to be fair versus other UNIX processes, it is optimized for the case where the machines are temporarily dedicated to one application and it makes for instance a lot of use of busy loops and memory pinning that would severely affect other applications performance. Anyway the system does not need any special configuration, and besides the presence of a CPU hungry process, it can be used like a normal UNIX system.

4.3 Dealing with big messages

Data path limitation When a message is sent between two computers, the data should be transmitted from main memory to the network board (step 1),

² On some architecture (some Sun or Digital systems), the DMA can be programmed with virtual addresses, there is in fact an IOMMU that must be programmed before a transfer with the physical address of the user data.

³ It allows an unprivileged user to use the "almost standard" `mlock` system call with some security check

then should be put on the wire (step 2). It will be received on the destination in the board (step 3) and then should be transferred to the final destination in main memory (step 4). With the previous technology, the transfer steps between the main memory and the network board were neglected, nowadays on some platforms this is the main bottleneck, or it is of the same order than the speed of the network.

For instance, on our test-bed, the hardware allows transfers on the PCI bus at about 130Mbytes/s. The main memory bandwidth is a bit greater, so it can sustain this rate. The current Myrinet/PCI board maximum throughput on the wire is less than 132Mbytes/s. Hence, although the data on the wire could be cadenced at 160Mbytes/s there is two steps of transmission that are done at less than 132Mbytes/s. Clearly the peak theoretical throughput of the platform is 132Mbytes/s. Moreover achieving something close to this rate suppose that all the transfer operations (step 1, 2, 3 and 4) are fully pipelined. In our case, that means at the same time four simultaneous DMAs, two on the sending host and two on the receiving host !

Note that in absence of contention, the different switches along the patch of a message do not limit the bandwidth. Indeed the Myricom switch is a worm-hole switch, and from our measures, the only visible effect is a very low latency overhead (about 100η s per switch).

Pipeline transmission In order to approach as much as possible the fully pipelined case, we decompose the message into packets of equal size, depending of the total length. The host processors are only involved at the transfer initialization to give to the board the location where to take and respectively store the target messages. After that all the transmission is managed by our protocol implementation on the boards.

In BIP, each packet is transmitted in four steps as described above, if the message is split in $n > 4$ packets, the transmission will be fully pipelined from the start of the fourth packet until the start of the last packet. Our MCP⁴ exploit the Myrinet capabilities: it does at the same time a DMA transfer from main memory to the LANAI memory, and the transfer of the previous packet from LANAI memory to the wire. The situation is symmetric on reception.

In a first approximation one can consider each step is done at the same speed (corresponding to the lowest performance link in the transmission chain). A complete model of the transfer that matches with the experimental delays was done in order to compute the optimal size of the packets [Wes97].

The optimal size of the packet to achieve the fastest transmission time for a given message highly depends on its length. Using the model of the platform from [Wes97] in order to tabulate packet sizes, the BIP protocols adapts as a function of the message length and try to maximize data pipelining. We recall also that we use memory copy and “Programmed IO” for small messages when it is faster than DMA. This is shown on Figure 4: the width between two jumps correspond to the size of a packet.

⁴ Myrinet Control Program

We said previously that a message contiguous in user memory will not be contiguous in physical memory. The host gives to the LANAI a list of gather/scatter physical areas corresponding to the user message. The MCP use this list to gather the data. Our implementation arranges so that the flow of bytes on the wire is continuous for the different packet of the same message. There is no additional cost when buffers are not on page boundaries or not aligned together between sender and receiver. Only the first packet need special care.

When several messages arrives almost simultaneously, the Myrinet network block others message until the first has completed. In our first implementation, used for this paper results, we transfer all user messages as one Myrinet messages (but in several DMA operations). We have also implemented real packetization (necessary for bigger network). In this case, our strategy is to allow only one multi-packet message sender at a time for a given destination. The arbitration is done on the destination, a small request/go round-trip is necessary before actual delivery. The rational is to have at most one message at a time being reassembled from packets.

4.4 Dealing with small messages

When dealing with very small messages, the initialization time to exchange the information between the board and the host becomes predominant over the transmission time itself. It is also be clear that the message will be transmitted in only one chunk.

At the BIP level, the cost of determining the DMA parameters by conversion of the virtual address to the physical address is bigger for small messages than doing one memory copy. Hence, for messages smaller than a given size, there is a point where our algorithm starts doing memory copies to a fixed communication buffer, on the interface card in order to decrease latency. At that optimization level, every instruction counts. Several functionalities of the processor, such as write-combining were used to decrease the latency.

4.5 Security

The BIP implementation does not protect the system against a malicious user. The implementation works almost totally at the user level and so the network board memory and registers are mapped into the user space. A consequence of giving the user access to the board, is that all system securities can be broken (at least by a malicious user), giving him an easy way to become root. But in fact nothing in BIP design prevents the implementation of a secure version of the BIP protocol and we are actually working on it. It requires that the OS protects at initialization time, by using the MMU, some data structures on both the LANAI SRAM and in main memory. The main overheads in performance are: the add of some checks in the MCP, to verify the validity of requests made by the user program; and the need for the OS to give himself the physical addresses of the destination area to the MCP.

5 Communications benchmarks

The validation of our design is in the performance it delivers on the PCs connected to a Myricom LAN. In this Section, we present the results in two parts, latency in the protocol processing and throughput achieved with the BIP messages.

Our measurements are in the general case, where the application buffers are allocated with a general "malloc" primitive, with no restrictions.

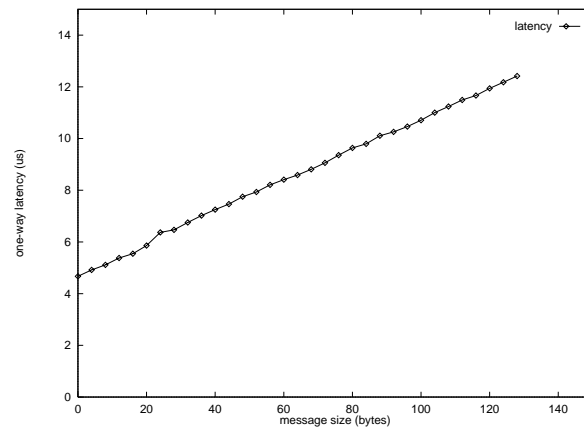


Fig. 2. Latency of BIP messages for small messages on Myrinet - PCs platform.

5.1 Latency

We measured the latency of our implementation by sending 1000 messages with no payload, sending them back and measuring for each the round trip delay divided by 2. The result we choose is the median, i.e. the 1000 values are sorted and the result is the 500th value. This seems fair to us because it gives what the user can expect, taking apart extreme values. The results obtained represent less than 1000 cycles of the 200 MHz processors: with such small timings each instruction counts !

5.2 Throughput

We measured the throughput of our implementation as a function of the message size. Each value is obtained by sending 100 messages of the same size, sending them back and measuring for each the round trip delay divided by 2. The result we choose is again the median. The curve 3 shows the very good behavior of the protocol that reaches half of its speed for small messages (4K bytes). The

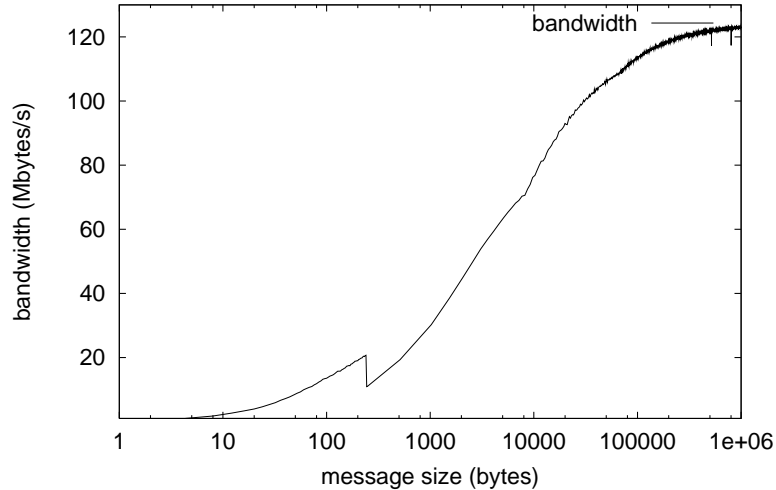


Fig. 3. Throughput of BIP messages on a Myrinet - PCs platform.

maximum throughput is 126 Mbytes/s which represent more than 96% of the 132Mbytes/s theoretical speed of our test-bed. The jump about 256 bytes, is due to the switch between the strategy for small messages, and the one for big messages.

The Figure 4 shows the impact of the adaptive strategy that maximize the pipeline effect. Thus the performance increases very rapidly.

The table 1 compares our results with well-known parallel machines and an ATM LAN of SUN Sparcstations. The Myrinet hardware on PCs with our BIP protocol compares favorably with most of these communication platforms. The $N\frac{1}{2}$ field is the size, from which half the bandwidth is obtained.

5.3 Applications

As regards end-users applications, we are providing a full conforming MPI implementation on top of BIP (and based on MPICH). Moreover note that several implementation technics (memory pinning, and port management with BIP tags) are transparent for the user, so he could also work at the BIP API level. As measured with the netlib MPI communication benchmark[DD95]: one-way latency=9us, maximal bandwidth = 125Mbytes/s.

Note that IS is the most communication intensive benchmark but also the one where floating-point processor performance is not important (that is the weaker point of the Ppro against the other processors).

The table 2 presents results for the NAS benchmarks:

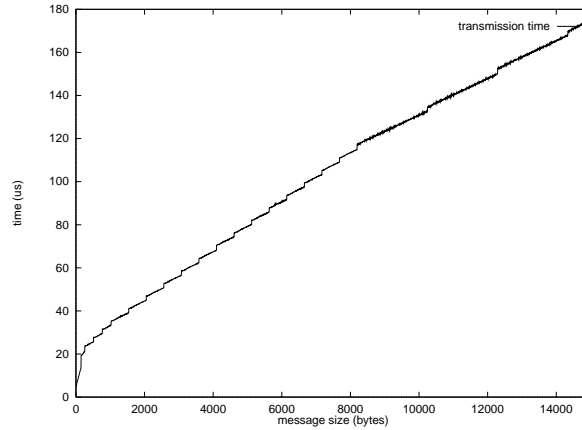


Fig. 4. Transmission time of BIP messages, this curve illustrate the adaptive packet size strategy.

machine	latency (μs)	Bandwidth (Mbytes/s)	$N^{\frac{1}{2}}$ (bytes)
ATM155/sparc5 (AAL5)[PT97,Pry96]	500	10	10000
iPSC/860 (NX)[DD95]	65	3	340
TMC CM-5 (CMMD)[DD95]	95	9	962
Intel Paragon (NX)[DD95]	29	154	7236
Intel Paragon (MPI)[GLDS96]	40	70	7000
Meiko CS2[DD95]	83	43	3559
IBM SP-2 (MPI)[DD95]	35	35	3262
T3D (shmem)[DD95]	3	128	363
T3D (MPI)[GLDS96]	21	100	7000
SGI Power Challenge (MPI)[GLDS96]	47	55	5000
Myrinet/Ppro200 (BIP)	4.3	126	4096
Myrinet/Ppro200 (MPICH+MPI-BIP)	9	125	8192

Table 1. Comparison of communication performance for some parallel architectures.

	MPI-BIP/Ppro	T3E	Origin2000
IS	1.9	3.2	2.1
SP	18	43	60.3
LU	34	70	90

Table 2. NAS benchmark CLASS A with 4 processors (Mops/node)

6 Related work

Our work is directly in the ideas that motivated the “Active Message” or the “Fast Messages” in order to get rid of the costly protocol stack and system calls.

One difference is that we do not intended to provide an other paradigm of message-passing, BIP have been designed specifically as a basis to implement efficiently MPI or PVM. We designed BIP to be able to avoid memory copy in MPI: our implementation of MPI add an header to the payload only for small messages (for which memory copy does not matter); for bigger message the payload is transmitted on a different channel than the MPI header. So no copy is necessary in MPI-BIP.

The Active Message from Berkeley[vECGS92] are providing the RISC-type communications with a handler activation at the receiving side. We are not providing such a handling facility, we propose an optimize data path, an adaptative protocol and a low level pipelining.

The Fast Messages from Illinois[PKC97,LC97] included a complex flow control protocol in order to manage the communications at the user level. It includes an interface for a general scatter/gather capability (that we do not have), it can be used for instance by an upper layer to add an header to a message.

U-net from Cornell University[BBVvE95] also provides the most of the protocol stack in the user space to avoid system calls. Its main interest is that it provides a secure framework, to multiplex the network in a secure manner between several users.

UHN-net*[HRKQ96] gives a share data space for the user and the system by rewriting the OS kernel. We do not provide such deep modifications of the kernel and avoid memory copies.

PM form RWCP[HT96] is dedicated to the special OS work that is developed in the RWCP group. A pipeline strategy must be used regarding the performances obtained but no adaptative packetization is described.

Our work is closely related to all the cited with the aim of performance for message passing application. Compared to Active Messages or Fast messages, we provide a kernel mechanism to be able to send any data in a user process without memory copy. The work at U-Net is more multi-user oriented, they provide much more functionality (multiplexing the network between several user with security), compared to them we have restricted ourselves to simple functionality, but we have focused on low latencies issues, particularly as regards the protocol between the interface card and the processor node.

One other difference is that we implemented the packet pipelining at the lower level. From our point of view, we think that flow control should be adapted to the requirement of the upper layers. Thus BIP provides only functionality to implement flow-control. For instance MPI-BIP implements a credit-based flow control based on BIP queues. But other specialized applications may only need a limited control-flow, using some application known properties.

7 Conclusions and future work

We are actually working on the upper layers that will be interfaced with BIP. These works introduce feedback to our BIP functionality and we are discussing which of them should be supported at the BIP level, such as reliability checks, multi-application support and security.

The MPI library is currently ported with the MPICH Channel interface. Our first results shows only a 10 % loss in performance for the basic MPI communications compared to BIP, which, to our knowledge, gives a world record to Myrinet/MPI-BIP other any machine/MPI implementation (see table 1).

The IP-BIP driver is our next goal in order to provide the TCP/IP support at high speed. This is used like a normal network driver and we got actually promising results at 50Mbytes/s. More results are available with the netperf benchmark on <http://www.cup.hp.com/netperf/NetperfPage.html>. This TCP environment is actually running on our platform and is running all the IP traffic (NFS,X11,...).

Actually BIP provides only raw control flow capabilities. But our MPI-BIP layer does complete it with a credit-based approach. The part that is really missing in BIP is error recovery for environment where failure rate would not be negligible. Currently in case of bit error, BIP can either exit the application with an error or interrupt it for error treatment (like in IP-BIP).

Finally the raw performance numbers show that the new networking technology is going to change many way of thinking specially when one realized that disk access now become more than one order of magnitude costly than access to remote Megabytes of RAM, it opens new opportunities in the field of distributed file systems, new opportunities for process migrations, swapping, and so on. It would be now possible to design a cluster of workstations viewed as just one server and migrate efficiently process to do load balancing. For parallel computing, it extends the field of possible applications, low latency allowing a finer parallelism granularity. General communication performance permits the scalability of applications to larger configurations.

Our future works will concern implementation of higher level API (MPI or PVM), secure version of BIP (in the UNIX security meaning), and multiplexing of application for BIP.

As our work is progressing with higher communication functionality, we will enlarging BIP with the services that are not decreasing the performances, as we did with tags and the queuing facilities for small messages.

The project home pages are available on the net
<http://lhpc.univ-lyon1.fr>

References

- [AB97] Thorsten von Eicken Anindya Basu, Matt Welsh. Incorporating memory management into user-level network interfaces. Technical report, Department of Computer Science, Cornell University, 1997.

- [BBVvE95] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, Colorado, December 1995.
- [BCF⁺95] Boden, Cohen, Feldermann, Kulwik, Seitz, Seizovic, and Su. MYRINET: A Gigabit per second Local Area Network. *IEEE-Micro*, 15:29–36, February 1995.
- [BS95] Gilles Berger-Sabbatel. PVM and ATM networks. In Dongarra, Gengler, Tourancheau, and Vigouroux, editors, *EuroPVM*. Hermes, 1995.
- [DD95] Jack Dongarra and Tom Dunigan. Message-passing performance of various computers. Technical Report CS-95-299, University of Tennessee, July 1995.
- [GBD⁺94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. *PVM: Parallel Virtual Machine*. Scientific and Engineering Computation. MIT Press, 1994.
- [GLDS96] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, 1996. <ftp://ftp.mcs.anl.gov/pub/mpi/mpicharticle.ps.gz>.
- [HRKQ96] Philip J. Hatcher, Robert D. Russell, Santhosh Kumaran, and Michael J. Quinn. Implementing data-parallel programs on commodity clusters. In *the Spring School on Data Parallelism*, March 1996.
- [HT96] Yutaka Ishikawa Hiroshi Tezuka, Atsushi Hori. Pm:a high-performance communication library for multi-user parallel environments. Technical Report TR-96015, RWC, 1996. <http://www.rwcp.or.jp/lab/pdslab/papers.html>.
- [LC97] M. Lauria and A. Chien. MPI-FM: High performance MPI on workstation clusters. *Journal of Parallel and Distributed Computing*, February 1997.
- [Myr95] Myricom. Myrinet link and routing specification, 1995. <http://www.myri.com/myricom/document.html>.
- [PKC97] S. Pakin, V. Karamcheti, and A. Chien. Fast messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Concurrency*, 1997.
- [Pry96] Loïc Prylli. Calcul parallèle sur réseau ATM de stations de travail. In *Renpar'8*, 1996.
- [Pry97] Loic Prylli. Bip user reference manual. Technical Report TR97-02, LIP/ENS-LYON, Septembre 1997. <http://www-bip.univ-lyon1.fr/software/bip-manual.ps>.
- [PT97] Loïc Prylli and Bernard Tourancheau. Parallel computing on an ATM LAN. In *ATM97*, Bradford, UK, 1997. IFIP.
- [SOHL⁺95] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, 1995.
- [vECGS92] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schausser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Int'l Symp. on Computer Architecture*, Gold Coast, Australia, may 1992.
- [Wes97] Roland Westrelin. Réseaux haut débit et calcul parallèle: étude de myrinet. Master's thesis, LHPC, CPE-Lyon, 1997.