

# The design for a high performance MPI implementation on the Myrinet network<sup>\*</sup>

Loic Prylli<sup>1</sup>, Bernard Tourancheau<sup>2</sup>, and Roland Westrelin<sup>2</sup>

<sup>1</sup> LIP and project CNRS-INRIA REMAP, ENS-Lyon, 69634 Lyon

<sup>2</sup> RHDAC and project CNRS-INRIA REMAP, ISTIL UCB-Lyon 69622 Villeurbanne  
Bernard.Tourancheau@inria.fr; <http://www-bip.univ-lyon1.fr>

**Abstract.** We present our MPI-BIP implementation, designed for Myrinet networks, and based on MPICH. By using our “Basic Interface for Parallelism: BIP” software layer, we obtain in this implementation of the MPI protocols results close to the peak hardware performance of the high speed Myrinet network. We present the protocols we used to implement the MPI semantics, and the overall design of the implementation. We, then, present benchmarks and application results to show that this design leads to parallel multicomputer-like throughput and latency on a cluster of PC workstations.

## 1 Introduction

In the last decade, researchers tried to use COWs (Cluster Of Workstations) as parallel computers. These clusters are typically connected by Ethernet networks and are often programmed with communication libraries like PVM (Parallel Virtual Machine [6]), or MPI over IP (Internet Protocol). There is two bottlenecks in these solutions that can restrict application programmers to coarse grain parallelism: either the network hardware (ethernet or fast ethernet) may not be fast enough, or the communication software (and this is particularly true for all IP-based systems) may have too much overhead.

It is now possible to build a platform with efficient hardware and software components. The price performance ratio of COWs depends on:

**The processing power** commodity components (like Intel Pentium or DEC alpha processors) now deliver competitive performance.

**The network speed** Massively Parallel Processor technologies (SCI and Myrinet) are available to interconnect workstations.

**The operating system** free software operating systems (like Linux) provide the flexibility and performance needed.

**The communication libraries**

This article introduce our works towards a high performance implementation of MPI for Myrinet-based clusters using the BIP protocols. We, first, released an MPI implementation on top of BIP in 1997 [15] using the MPICH Channel Interface. A re-implementation of MPI-BIP was done in 1998 (released in

---

<sup>\*</sup> This work was supported partly by a Myricom grant and the Région Rhône-Alpes.

November) based on the work done at Myricom on the MPI-GM[1] implementation. The paper describes the design of the last implementation, most of it is shared between MPI-GM and MPI-BIP.

The first part of this paper introduces previous works in this domain and BIP. Then we present the design of MPI-BIP, our adaptation layer protocol for MPICH, followed by some performance results.

## 2 Low-level layers

Methodologies for the design of protocol stacks changed with the new high performance networks. The bottleneck now can often be the interface between the network and the host or the communication system if badly designed, especially in regards to latency. Several research teams proposed new approaches:

**Active Messages (AM) [14]** use an RPC-like (Remote Procedure Call) mechanism. Each message begins with the address of a handler. It is executed by the receiver to integrate the data in the ongoing computation. A MPI implementation on top of the AM is available [3].

**Fast messages (FM) [11]** use the same RPC-like system. They include a heavy flow control protocol. The designers of FM provide an implementation of MPI [9].

**PM [8]** is a fast communication layer for the myrinet network. MPI was ported to PM [10].

**GM [1]** is the native API for the myrinet hardware developed by myricom. MPI is also available on top of GM.

**Virtual Interface Architecture (VIA) [5]** Computer industry leaders (Microsoft, Intel and Compaq) proposed a new software architecture for an efficient access to the network hardware from the user applications. VIA borrows a lot to U-Net [2]. At the time of writing, no free implementation of MPI is available for VIA.

### 2.1 BIP (Basic Interface for Parallelism)

BIP [12] is a low level layer for the Myrinet network developed by our team in Lyon. The goal of this project is to provide an efficient access to the hardware and to allow zero memory copy communications. BIP only implements a very raw link level flow control. It is not meant to be use directly by the parallel application programmer as it supplies very few functionalities. Instead, higher layers are expected to provide a development environment with a good functionality/performance ratio.

The API of BIP is a classical message passing interface. BIP provides both blocking and non-blocking communications. Communications are as reliable as the network, errors are detected and in-order delivery is guaranteed. BIP is composed of a user library, a kernel module and a Network Interface Card (NIC) program. The key points of the implementation are:

**A user level access to the network**, avoiding system calls and memory copies implied by the classical design, becomes a key issue: the bandwidth of the

network (160 MB/s in our case and 133 MB/s for the I/O bus) is the same order of magnitude than the speed of a memory copy.

**The long messages** follow a rendez-vous semantic: the send is guaranteed to complete only if a corresponding receive has been posted because of the limited capacity of the network. Messages are split into packets and the communication steps are pipelined (see [13]).

**The small messages**, because initializations and handshakes between the host and NIC program are more expensive than a memory copy for small messages, are directly written in the network board memory on the sending side and, copied in a queue in main memory on the receiving side. The size of this queue is static, it is up to the application to guarantee that no overflow occurs. On the receiving side, a memory copy puts the message in the user buffer platform.

The communication latency of BIP is about  $5.0 \mu\text{s}$ , the maximal bandwidth is 126 MB/s (95% of the PCI maximum which is the bottleneck in our case). Half the maximum bandwidth is reached with a message size of 4KB ( $= N_{1/2}$ ).

### 3 The internals of MPI-BIP

MPICH is organized in layers and designed to ease the porting to a new target. The figure 2 presents our view of the MPICH framework, and at what level we plugged the MPI-BIP specific part, different ports choose different strategies depending on what communication system they use.

There is an internal intermediate API inside MPICH called the “Channel Interface” often used for new ports. This API requires to implement a non blocking send for the control messages (small messages used to exchange informations between the MPI processes, as explained below) and the possibility to queue a high number of asynchronous requests. BIP does not fulfill these requirements. That is why we did not plug it at the “Channel Interface” level<sup>1</sup>. We implemented our network specific layer at a non-documented interface level, that we called the “Protocol Interface”, because this API allows to specify custom protocols for the different kinds of MPI messages.

Each MPI messages of the application, is implemented with one or several messages of the underlying communication system (BIP in our case). We distinguish two logical kind of BIP messages that we use to implement MPI: the “control” messages, and the “large data” messages.

#### 3.1 “Control” messages

The control messages are sent directly to the network when the MPI internals need to exchange some control information. If the receiver does not expect a control message, it is stored in the queue of BIP. This queue is static (i.e. of constant size). Even if the size of a control message varies (depending on the size of the data that are included or on the type of informations exchanged by

---

<sup>1</sup> The original MPI-BIP implementation worked around this with some tricks.

the upper layers), an upper bound can be determined. Hence, the maximum number of messages that the queue can hold is known. To guarantee that we do not send a number of messages greater than this maximum, a flow control algorithm is used (see 3.5).

There are mainly four kind of control messages:

- small application message (encapsulated in a control message),
- request of transmission of a large message,
- acknowledgement, sent when the receiver is ready for a requested transmission,
- credits (used for flow control).

### 3.2 “Large data” messages

Large messages are used to transfer the user-data for MPI messages larger than a fixed threshold. Such messages cannot get into a statically allocated queue. As described below, a preliminary handshake with “control messages” is necessary, to ensure that the receiver is ready before the user data is actually sent.

### 3.3 The internal protocols

MPICH originally implements three different protocols on top of the “Channel Interface” (see figure 2) described in[7]:

- short protocol: data is added to the payload,
- eager protocol: a control message is sent, and the payload is sent separately just after,
- rendez-vous protocol: after sending a control message, the sender must wait an acknowledgement before sending the payload.

For MPI-BIP, we needed to partly modify these protocols. First the eager protocol is not usable with BIP (because it does not allow the destination to prepare itself before the emission of the large message starts at the other end), instead we replaced it with a three-way protocol that will either allocate a temporary buffer on the receiving side if the user did not post any matching receive yet, or will directly receive into the user-buffer. Then the rendez-vous protocol has been extended to allow the payload to be sent in several fragments separated by acknowledgements to workaround the message size limitations of BIP.

The correspondence between MPI communications and those three transmissions is done with the following rules:

- MPI Synchronous sends always use the rendez-vous protocol,
- MPI Buffered sends are managed by a generic layer of MPICH and transformed into Standard sends,
- MPI Standard and Ready sends use either the short protocol, the three-way protocol or the rendez-vous protocol depending on their size.

These rules determine completely what kind of BIP messages will be needed to implement a MPI message of the application.

### 3.4 Request FIFOs

The protocols above rely on the ability to do any number of non-blocking network operations (so no deadlock is possible in their executions). We need some additional mechanisms on top of BIP to ensure this non-blocking capability on the sending side for two reasons:

- BIP allows only to queue a limited number of asynchronous sends, when the maximum is reached we would need to wait before posting a new send.
- The flow control mechanism for control messages described in the next section may prevent us to send immediately to a given destination.

When these situations occur, we put any new send request in a software FIFO, it will be transformed into a BIP send later (upon receiving flow control credits, or on completion of previous sends). This software FIFO actually allows to present to the upper layers of MPI-BIP a virtual communication systems supporting an unlimited number of non-blocking sends.

We have to deal with a similar resource limitation problem on reception. When receiving a request for a large message, one of the requirements before sending the acknowledgement, is to first post the BIP receive request. As only a small number of simultaneous receives can be posted with BIP (one for each tag), we need a FIFO containing receive requests that need to be delayed. If a request is put into this FIFO, the corresponding acknowledgement will only be sent at the time the request goes out of the FIFO.

### 3.5 The flow control algorithm for the control messages

In order to avoid overflow of the fixed-size receive queues of BIP, a credit-based algorithm is used to manage the control messages.

A machine  $i$  manages two credit accounts for each other machine  $j$ :  $C(i \rightarrow j)$  which is the number of messages  $i$  can send to  $j$  at a given time,  $B(i \leftarrow j)$  and the number of consumed messages from  $j$  for which  $i$  have not yet sent back corresponding credits.

Each time a control message is send from  $i$  to  $j$ :  $C(i \rightarrow j)$  is decremented, the value of  $B(i \leftarrow j)$  is put in a dedicated header of the message, after which  $B(i \leftarrow j)$  is reset to zero.

On  $j$ , upon reception of a control message from  $i$ , the credit value in the header is added to  $C(j \rightarrow i)$ , and  $B(j \leftarrow i)$  is incremented.

Let  $N$  be the number of credits for each peer of processes, (at all times we have  $C(i \rightarrow j) + B(j \rightarrow i) \leq N$ ), on machine  $i$ , when  $B(i \leftarrow j)$  reaches  $N - 1$ , we need to force the emission of a “credits only” control message to  $j$ . In usual applications, such dedicated messages will be very rare, credits will go back in the header of other control messages.

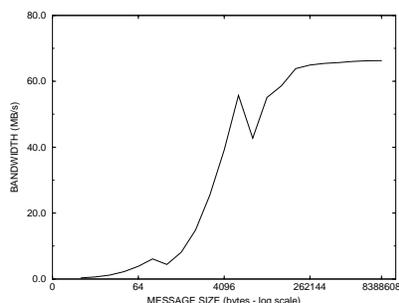
Note that we must always reserve the last credit for the eventuality of a “credits only” control message, so  $N$  must be at least 2 (other “control message” goes into the previously mentioned FIFO if  $C(i \rightarrow j) \leq 1$ ).

## 4 Performances measurements

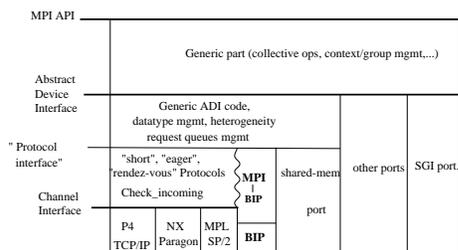
Our LHPC<sup>2</sup> experimentation platform is made with 8 nodes, each with an Intel Pentium Pro 200 MHz, 64Mbytes of RAM and a 440FX chipsets, and a myrinet board with LANai 4.1 and 256Kbytes of memory.

### 4.1 Point to point communications

We ran point to point measurements for communications between two Myrinet connected Pentium-Pros running Linux. The performance measurement method uses the median of 100 experiments. They consist in round trip communications: the message is sent, the receiver returns it only when it is fully received. The one way timing is obtained by dividing the round trip timing by 2.



**Fig. 1.** Point to point communication performance: bandwidth vs message size in logarithmic scale



**Fig. 2.** The architecture of MPI-BIP

	Latency ( $\mu$ s)	Bandwidth (MBytes/s)	$N_{1/2}$
Intel Paragon (MPI)[7]	40	70	$\approx 7000$
IBM SP-2 (MPI)[4]	35	35	3262
T3D (MPI)[7]	21	100	$\approx 7000$
SGI Power CHallenge (MPI)[7]	47	55	$\approx 5000$
Myrinet/Ppro200 (BIP)	5	126	4096
Myrinet/Ppro200 (MPI-BIP) 1st implementation from 1997	9	125	8192
Myrinet/Ppro200 (MPI-BIP) (with 10kb fragments)	9.5	66	3072
Myrinet/Ppro200 (MPI-BIP) (with 100kb fragments)	9.5	110	8000

**Table 1.** Comparison of communication performance with some parallel architectures.

Figure 1 gives the bandwidth of MPI-BIP. Three artifacts are clearly visible. They match the 3 strategies used: small messages are sent directly; medium messages are sent using the three-way protocol of MPICH; big messages are split into several fragments (10Kbytes for this experiment) using the rendez-vous protocol, (note that the frontier between three-way and rendez-vous cannot be seen in the case of a ping-pong experiment).

<sup>2</sup> Laboratoire pour les Hautes Performances en Calcul: a cooperation structure between ENS-Lyon, région Rhônes Alpes, CNRS, INRIA and Matra Système Information.

These results are very good and our software puts the Myrinet network in the range of the parallel machines as shown in Table 1. Note that although it was the fastest, the original implementation from 1997 is now obsolete, because of some MPI conformance problems.

## 4.2 Running the NAS benchmarks

The NAS parallel benchmarks (<http://science.nas.nasa.gov/Software/NPB/>) are a set of parallel programs designed to compare the performance of supercomputers. Each program tries to test a given aspect of parallel computation that could be found in real applications. These benchmarks are provided both as single processor versions and parallel codes using MPI. We selected 3 different benchmarks (LU, IS and SP) and compiled them with MPI-BIP. They are then run on 1, 4 and 8 nodes of our Myrinet cluster described previously. The table 2 gives our measurements and comparisons with several parallel computers. Data for parallel computers come from the NAS web site.

	MPI-BIP on PPro 200		IBM SP (66/WN)		Cray T3E-900		SGI Origin 2000-195		Sun Enterprise 4000	
	Mop/s	Speedup	Mop/s	Speedup	Mop/s	Speedup	Mop/s	Speedup	Mop/s	Speedup
IS @ 4 proc	2.44	4.5	2.2	3.1	3.2	N/A	2.1	3.8	1.6	3.8
IS @ 8 proc	2.11	8.8	2.0	5.6	3.8	N/A	2.4	8.6	1.5	6.9
LU @ 4 proc	23.68	6	59.0	3.7	67.6	N/A	96.8	N/A	36.9	4.1
LU @ 8 proc	22.73	11.6	57.2	7.1	66.4	N/A	103.3	N/A	37.3	8.4
SP @ 4 proc	10.10	3.4	42.1	3.6	43.0	N/A	60.3	N/A	25.0	3.7

**Table 2.** Performance for NAS Benchmarks on various platforms.

The speedup values summarized in the table 2 show the very good performance of MPI-BIP and Myrinet. Super-linear speedups in our case can be explained by the large cache size and the super-scalar architecture of the PPro. The behavior of MPI-BIP is excellent both for small messages and large messages. However, when a lot of computational power is needed (SP), a gap appears between traditional parallel machines and our cluster: the weak floating-point unit of the PPro 200 shows its limits!

## 5 Future works

Our work proves that the performance gap between workstation clusters and parallel machines is nearly filled in. However, even if parallel machines are still more powerful, COW based on inexpensive hardware has an unbeatable performance price ratio.

Our future work on MPI-BIP will investigate the increase of the performance by implementing some of the flow control at the BIP level in the Network Interface Card program. We would especially like to improve the throughput of MPI and enable a better communication/computation overlap and also get rid of the performance artifacts.

We are trying to generalize our approach with a new MPICH architecture that should ease the porting of MPI on new high speed network architectures. Furthermore, we are studying the modifications needed at the BIP level to support efficiently the MPI 2 functionalities. The software has been downloaded by more than 110 different teams worldwide and researcher are using it for large applications.

## References

1. Myricom gm. <http://www.myri.com:80/GM/>.
2. 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.
3. Chi-Chao Chang, Grzegorz Czajkowski, Chris Hawblitzel, and Thorsten von Eicken. Low-latency communication on the IBM RISC system/6000 SP. *ACM/IEEE Supercomputing '96*, November 1996.
4. Jack Dongarra and Tom Dunigan. Message-passing performance of various computers. Technical Report CS-95-299, University of Tennessee, July 1995.
5. Dave Dunning and Greg Regnier. The Virtual Interface Architecture. In *Hot Interconnects V*, Stanford, USA, August 1997.
6. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Scientific and engineering computation. MIT Press, Cambridge, MA, USA, 1994.
7. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.
8. 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>.
9. M. Lauria and A. Chien. MPI-FM: High performance MPI on workstation clusters. *Journal of Parallel and Distributed Computing*, February 1997.
10. Francis O'Carroll, Atsushi Hori, Hiroshi Tezuka, and Yutaka Ishikawa. The design and implementation of zero copy MPI using commodity hardware with a high performance network. *ACM SIGARCH ICS'98*, pages 243–250, July 1998.
11. S. Pakin, V. Karamcheti, and A. Chien. Fast messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Concurrency*, 1997.
12. Loïc Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on myrinet. *Workshop PC-NOW, IPPS/SPDP98*, 1998.
13. Loïc Prylli, Bernard Tourancheau, and Roland Westrelin. Modeling of a high speed network to maximize throughput performance: the experience of BIP over myrinet. *Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, 1998.
14. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer. 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.
15. Roland Westrelin. Réseaux haut débit et calcul parallèle: étude de myrinet. Master's thesis, ENS Lyon, Université Lyon 1, INSA Lyon, 1997. <http://lhpc.univ-lyon1.fr/PUBLICATIONS/pub/RWdea.ps.gz>.