Protocol and Performance Analysis of the MPC Parallel Computer

O. Gluck, A. Zerrouki, J.L. Desbarbieux, A. Fenyo, A. Greiner,
F. Wajsburt, C. Spasevski, F. Silva, E. Dreyfus
University P. & M. Curie, Laboratoire LIP6, ASIM Team
4, Place Jussieu 75252 Paris Cedex 05 France Tel : (+331) 44 27 52 53 Fax : (+331) 44 27 72 80
E-mail : { Olivier.Gluck, Amal.Zerrouki, Jeanlou.Desbarbieux, Alexandre.Fenyo, Alain.Greiner, Franck.Wajsburt, Cyril.Spasevski, Fabricio.Silva}@lip6.fr
Web : http://mpc.lip6.fr

Abstract

This paper presents the MPC parallel computer and its MPI implementation performed at the Laboratoire LIP6 of Univ. Pierre and Marie Curie, Paris. MPC is a low cost and high performance parallel computer using standard PC main-boards as processing nodes connected through the specific FastHSL board to a high speed communication network using HSL 1 Gbits/s serial links, IEEE 1355 compliant. Two Asics are presented : RCUBE which is the HSL network router, and PCI-DDC the network controller implementing the Direct Deposit State Less receiver protocol.

The software part of the MPC parallel computer consists of 2 zero-copy layers leading to a latency of 5 to 40µs and a throughput of 490Mbits/s. An efficient MPI implementation based on MPICH is presented and evaluated on an MPC parallel computer. Measures show a latency of 26µs and an useful throughput of 450Mbits/s

This paper presents also a performances study of the MPI implementation on the MPC computer. This reveals several possible optimizations to reduce the overall MPI transfer latency on the MPC Computer.

Keywords: PC-cluster, parallel computing systems, Interconnection networks, Message passing.

1. Introduction

The MPC computer (MultiPC) is the result of a 5-year research project at the LIP6. The goal was the design of a very low cost and high performance parallel computer under the form of a PC cluster. From the hardware point of view, MPC consists of several processing nodes interconnected by a high speed communication network (HSL) which complies with the IEEE-1355 standard[1]. Figure 1 presents the architecture of the MPC computer.

The low cost objective is attained through the use of off-the-shelf products. Nodes are nothing more than standard monoprocessor/multiprocessor main-boards. The only specific hardware element is the single board network controller FastHSL which connects the nodes to the HSL network and implements the communication protocol.

The differentiating and original element of the MPC computer resides in its HSL (High Serial Links) network designed at UPMC. It is a high performance network using 1 Gbit/s serial links.

Basically, a MPC node is a PC main-board connected with the FastHSL through the PCI bus. The FastHSL board has been developed by the Architecture Department of the LIP6. The board carries 2 ASICs, both developed by the Architecture Department of the LIP6 : The PCIDDC chip[2] is a PCI controller that implements the communication protocol. The RCUBE router[3] is a single chip 8*8 dynamic cross-bar that offers 8 bi-directional HSL ports. Thanks to the highly integrated RCUBE router, there is no centralized switch in this architecture, as each node contains a routing capability.

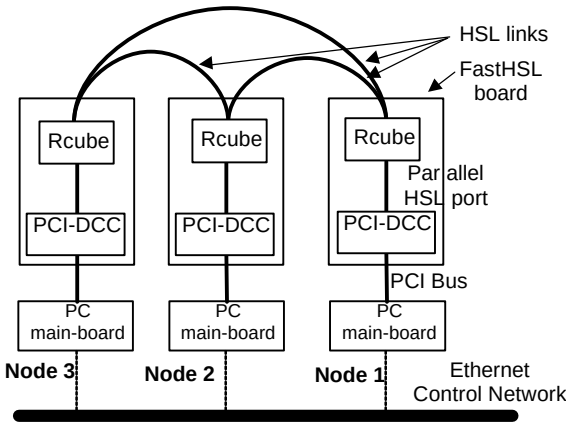

Figure 1. MPC architecture

2. The HSL network

2.1 The HSL link

The physical media is a point-to-point, full duplex and high speed serial link supported by coaxial cables.

Communication on the link is asynchronous : emitter sends data at its own speed and the receiver recovers both the emitter's clock and the data. Such a technology avoids the distribution of a synchronous signal over all the nodes and the resulting skew issues. The link carries data bytes encoded as shown in figure 2.

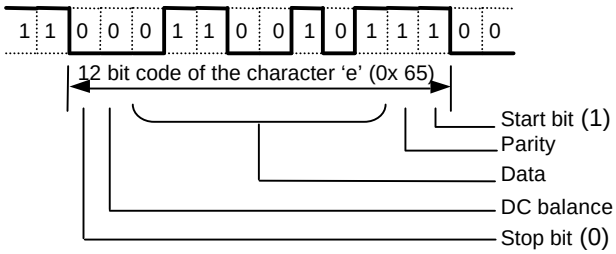

Figure 2. HSL 12 bits symbol coding

8 bits of data are enclosed in the stream. Start and Stop bits are used to create a rising edge every 12 bits. Such an edge is used by the receiving end for synchronizing its local clock with the one of the emitting end clock. A parity bit is also provided to allow error detection in the stream. In addition to the 256 possible data bytes, specials symbols are defined to ensure flow control, packet signaling, etc.

Given that 12 bits are necessary for each character, the maximum effective throughput is of 80 Mbytes/s (1Gbit/s/12bits/symbol = 80 Mbytes/s).

The maximum length of the HSL link in the coaxial version is of 5 meters. This limits the use of this technology to a local interconnection network. Optical fibers could extend the maximum length to 70 meters. However, this requires the use of external optical transceivers.

2.2 The serializing/de-serializing cell

This macro-cell developed at UPMC in collaboration with BULL performs the serialization/de-serialization of data for the HSL link as well as clock recovery. It has been designed in a portable technology and has been integrated in several ASICs and in various technologies from 0.5µm to 0.25µm. In a 0.5µm process its area is of 1mm^2 and its power consumption is of 300mW. Such a cell can be easily integrated in a VLSI component at low cost and does not require any external device.

2.3 The router

RCUBE (Rapid Reconfigurable Router) is a low latency dynamic router (150ns without contention) for transputing and networking applications. Designed with an internal clock of 80 MHz, it includes 8 bi-directional high speed links, 8 independent programmable routing tables (one routing table per input link), and one 8x8 non blocking crossbar switch which enables the routing of a packet from any input link to any output link. The architecture of the router is fully parallel and the transfer of a packet between one pair of links does not affect the data rate nor the latency of any other packet flowing through another pair of links.

It implements a wormhole flow control to reduce latency, prefix and interval routing schemes, and provides efficient supports for adaptativity.

RCUBE can route packets of any length. A packet is composed of a header which includes the destination address in the network, followed by a sequence of characters which compose the payload of the packet and ended by an end of packet character (figure 3). RCUBE uses a basic packet format to simplify the implementation of application specific protocols.

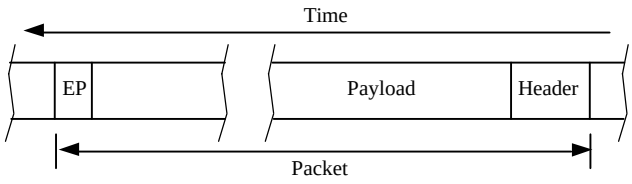

Figure 3. IEEE 1355 Packet Format

The flow control uses a credit mechanism : a router sends data to its neighbor if and only if it received the necessary emission credit corresponding to a flit (32 bytes of data). This technique introduces 3% penalty on the bandwidth but in return guaranties zero-loss transmissions.

Based on CMOS serializing/de-serializing cells, it delivers a global throughput of 1 Gigabit/s per link per direction, achieving a peak global throughput of 640 MBytes/s.

The RCUBE chip contains 500K transistors and has been fabricated by SGS-THOMSON in 0.5µm, 3.3 V technology.

2.4 The network controller

Each processor node is connected to an RCUBE router using a dedicated PCI to HSL interface named PCI-DDC. The chip implements the Direct Deposit State Less Receiver Protocol (DDSLRP) [4], developed at LIP6 to reduce the processor overhead. Classical data transfer protocols usually require several copies of data in

intermediate buffers before and after transmission through the network.

PCI-DDC has been designed to act as a master on the PCI bus so that it can access directly the host memory. In order to enhance performance, PCI-DDC implements a "remote write" primitive. This can be seen as a DMA request where PCI-DCC directly fetches data from the host memory and writes data directly into the remote memory. This is a "Direct Deposit" protocol. Furthermore, the sender processor defines the physical address where it wants the data to be written into the remote memory; this is a "State Less Receiver" protocol.

Page descriptors of a message are pushed by software into the LPE, a specific list located in host memory. This list contains the descriptors of pages to be emitted (local and remote page address, length, destination node, etc…). The emitting PCI-DDC reads the message descriptor through DMA accesses on the PCI-bus. Then, it starts data transmission, using again DMA accesses to the host memory relieving thus the processor from data transmission.

The physical remote address is transmitted with every packet. At the receiver end, as soon as PCI-DDC receives a packet, it starts writing incoming data at the corresponding memory location.

When the last page is written, the receiving PCI-DDC can notify its host processor, either with an interrupt signal, and/or by writing in a specific table located in the memory of the destination node.

The PCI-DDC chip is 32-bit 33MHz PCI-compliant interface which can operate at up to 120 MHz with an HSL port. The chip contains 200K transistors an has been fabricated by ALCATEL/MIETEC in a 0.5µm, 3.3 V technology.

2.5 The network controller

Myrinet[5] is the direct competitor of the MPC parallel computer. Its philosophy is similar but instead of serial links, the Myrinet network uses parallel Gigabit/s links. Moreover, the routing function is centralized and handled by a external unit. From a hardware point of view, the differentiating advantage of the MPC parallel computer resides in its scalability. Each FastHSL board used by an MPC node is provided with 8 serial HSL input/output ports connected to the RCUBE router. Adding a new node to the network can be performed at no extra cost. Using the Myrinet network, adding nodes requires cascading additional external routers since each router is limited to 16 ports.

3. The MPC software

From the software point of view, MPC runs a standard Unix-based operating system on top of which is added a set of protocols to drive efficiently the FastHSL board. The MPC software is built on top of Linux or FreeBSD-3.x operating systems.

The software part of MPC consists of 2 major layers : PUT, SLR. A zero-copy strategy is implemented in the layers to take advantage of the performance offered by the Direct Deposit protocol implemented in PCI-DDC.

3.1 The PUT layer

PUT is the lowest level software layer. It offers basic kernel communication services using the remote-write primitive of PCI-DDC. This layer provides a kernel API that writes page descriptors into the list of page descriptors to be sent (LPE), and handles event signaling.

PUT is located inside the kernel. It offers services to kernel protocols through function calls and to user space processes through system calls. To let multiple users call PUT simultaneously, and to support fast signal processing, PUT gives users disjoint sets of Message Identifiers (later called MI). Each user must associate its page descriptors with one MI, and PUT inserts it into the LPE. Thus, when a message is received by a node, the hardware inserts the MI associated to the data that has just been received in a table that PUT can access : the LMI (List of Message Identifiers).

Event signaling is implemented to notify message transmission and reception. It can be done within a polling loop or through hardware interrupts. Each PUT user must indicate a callback function just after the attribution of a set of MI. Signal processing is performed by calling this function with the MI associated to the received message as a parameter.

When the user requests an event-driven reception, an interrupt handler within PUT is called when data is received. PUT activates a callback function for each new LMI entry. When interrupts are disabled, the user on the reception node polls the LMI to check the arrival of new messages. Here again, PUT calls the callback function associated to the MI.

Latency measures for a short message containing 4 bytes is only 4 µs on a Pentium II 350MHz. It is composed of 1.9 µs PUT latency, 1.7 µs hardware latency and less than 0.4 µs for the signaling (polling). Figure 4 details the throughput measured for the PUT API.

3.2 The SRL layer

On top of PUT is implemented SLR (State Less Receiver) communication layer. This layer offers higher level communication services such as zero-copy virtual channels, but it is only available on top of FreeBSD.

Moreover, the network is seen by users as a set of virtual channels, multiplexed by this layer. Users indicate the local memory address of the data to be

transferred/received and the associated virtual channel on which data will be transmitted or received. Unlike the PUT user, the SLR user does not need to know the data location in the remote memory. The only necessary information is the channel number on which data is to be received or transmitted.

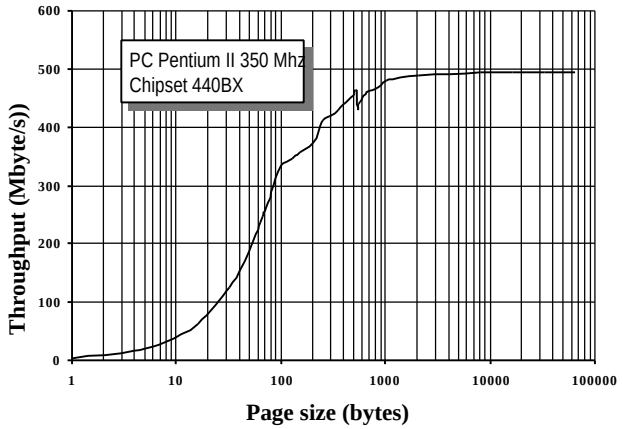

Figure 4. PUT Latency measures

Virtual to physical address translation is taken in charge by SLR, including the case of discontinuous physical memory addresses. Address translation is fast since SLR reads the Pentium MMU tables. Only two memory reads are necessary to perform the translation that has to be done for each 4Kbytes page.

SLR layers allows a throughput of up to 490Mbit/s of useful data, nearly like PUT. However, SLR introduces an additional latency of about 40 µs, essentially due to virtual channel management.

4. Message passing Implementation & results

On top of the SLR layer, an implementation of PVM has been realized. However, the performances obtained were poor because of the latency introduced by SLR and the multiple memory copies generated by PVM making useless the efforts spent in implementing a zero-copy policy in the SLR layer.

Consequently, efforts have been concentrated on an MPI implementation on top of the PUT layer. An advantage of this choice is that MPI implementation is available on both FreeBSD and Linux systems.

MPICH, developed by the Argonne National Laboratory, has been chosen for this implementation because it combines portability with performance due to the "CH" interface [6]. Indeed, it is designed to ease the porting on new platforms such as the HSL network [7].

However, MPICH requires that the interface layer supplies queuing facilities and flow control not available in the current PUT layer. The RESAM Laboratory (Lyon, France) ported MPI on BIP (Basic Interface for Parallelism)[8],[9]. They implemented their network specific layer at a non-documented interface level, called the "Protocol interface", because it allowed the specification of custom protocols for different kinds of MPI messages. Such an approach is also adopted for the implementation of MPICH on PUT (MPI-MPC). Figure 5 shows the MPICH architecture considered.

There are two kinds of messages at the "Ch_MPC" level : control messages, and large data messages. The following sections present the strategy used for implementing MPI-MPC. On its initial implementation, for the sake of simplicity and performance, each MPI node runs only one task.

4.1 Control messages

Control messages are used to transfer rapidly on the network some control information or limited size user-data. The maximum size of a control message is set to 180 bytes in the current MPI-MPC implementation. There are mainly four kinds of control messages :

- Small user-data message (encapsulated in a control message);
- Request of transmission of a large data message;
- Acknowledgement, reply to a received request;
- Credits, used for flow control.

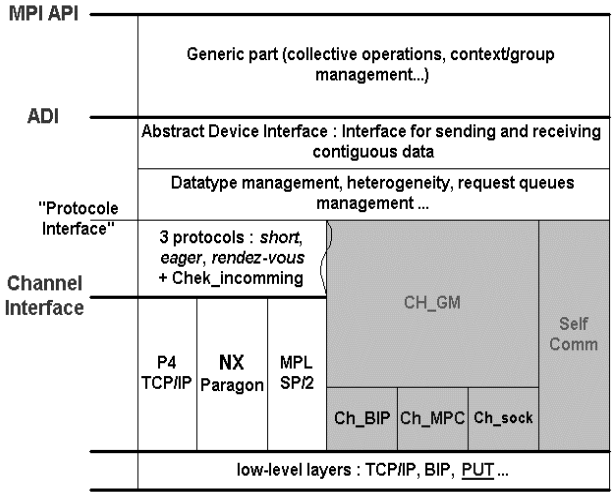

Figure 5. MPICH architecture

The specifications of PUT create two main problems. Firstly, data transmitted by PUT must be physically located in a contiguous memory space. Secondly, the sender has to know where to write data in the remote physical memory. The MPC software allocates at boot time an array of contiguous physical memory on all MPC nodes. When an MPI task starts, it gets a slot of this memory for control messages and maps it in virtual process memory. Each node gets the physical address of all remote slots through the control network (all nodes are connected by an Ethernet network for configuration).

As shown in figure 6, a slot is composed of a set of sub-slots used for emitting or receiving control messages. The number of sub-slots is determined by the number of nodes declared in the MPC machine (3 nodes in the figure). Each sub-slot is associated to a node identifier.

For instance, sub-slots 2 and 3 of node 1 are used to receive messages from nodes 2 and 3 respectively. Sub-slot 1 of node 1 is used by the local task for emission purposes. Moreover, each sub-slot actually consists of N buffers of 180 bytes (N = 4 in the example).

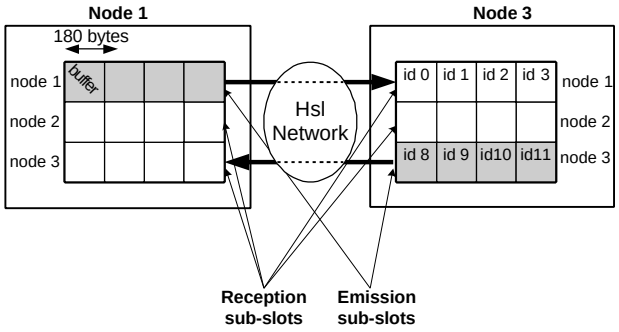

Figure 6. Memory slot structure

On the emission side, it is necessary to send a message to know the start address in the remote sub-slot. This can be determined from to the start address of the remote slot, the emitting node identifier and the number of messages already sent.

On the receiver side, when an incoming message is detected, the only information available is the message identifier (MI) delivered by PUT. The problem is to determine the sub-slot holding the message as well as the buffer within the sub-slot. This last is identified using the MI in which the sender has included the buffer identifier.

4.2 Large data messages

Large data messages are used to transfer messages larger than the maximum size of a control message (180 bytes) or for MPI synchronous sends.

Large data messages cannot get into statically allocated buffers. Therefore, it is required that a receiver sends a message to the sender when the receiver is ready to receive. This implies either a rendez-vous or an eager protocol using control messages to get the physical address on the receiver side.

The problem is that the sender/receiver supplies a virtual process address and the corresponding buffer is not necessarily a contiguous physical memory array. Therefore, address translation is performed during the Send/Receive which requires locking data into memory.

4.3 Preliminary results

This section presents the preliminary performance results obtained for the MPI implementation on the MPC parallel computer compared to the MPI implementation on BIP on a Myrinet network.

Although the current MPC-LIP6 parallel computer is composed of 4 Bi-Pentium PII-350 nodes running under FreeBSD3.4, measurements have been realized using the Linux MPC platform consisting of 2 standard PCs P166 to enable the comparison with MPI-BIP. The BIP/Myrinet platform consists of 2 PCs P200 also running Linux.

Both benches run the same application which is a Ping-Pong using the MPI_Send and MPI_Receive primitives. Time measurements use the internal Pentium clock counter (RDTSC).

For latency measurements, message size varies from 1 to 128 bytes (by a step of power of 2). Figure 7 presents the latencies obtained. The latency observed on MPI-MPC is of 26 µs compared to the 22 µs on MPI-BIP.

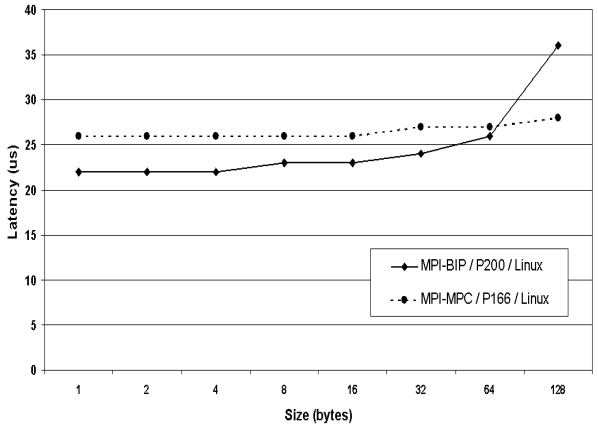

Figure 7. MPI-MPC and MPI-BIP latency comparison

Regarding the throughput, results are presented in figure 8.

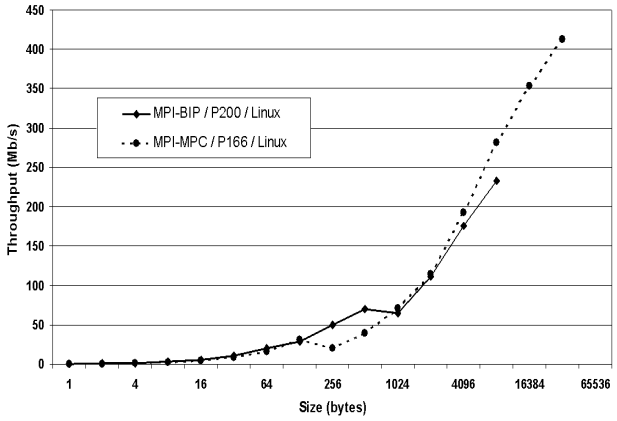

Figure 8. MPI-MPC and MPI-BIP throughput comparison

The throughput of MPI-MPC is about 450Mbit/s. An inflection point is observed for messages of 128 bytes in

the MPC results. This corresponds to messages longer than 128 bytes for which user data cannot be encapsulated in a control message. The same inflection point can be observed for MPI-BIP but for messages longer than 512 bytes; this last being the maximum size of control messages.

The performance comparison of MPI-MPC and MPI-BIP shows comparable throughputs and close latencies, accounting for differences in clock speed. But still, one advantage of the MPC platform resides in its scalability discussed earlier. Moreover, the encouraging performances presented in here are preliminary and can be enhanced. Software optimizations can be considered to improve more specifically latency. These are discussed in the section below.

5. MPI Performance analysis

The MPC parallel computer performances have been presented in the sections above. This section will rather focus on the MPI measures and specifically on the latency. It will detail the different stages of an MPI transaction to locate possible optimizations.

As mentioned in the description of the MPC software layers, PUT offers two event signaling policies: interrupts or polling. In the first implementation of MPI on MPC using interrupts, a latency of 28 µs has been measured. The current implementation of MPI over the PUT layer uses exclusively a polling policy allowing thus a latency reduction of 2µs.

But still, another way of reducing the latency can be considered. Let us consider the MPI ping pong pseudo-code (figure 9).

Node A	Node B
Forever do	***Forever do***
Start_Timer	
MPI_Send (n bytes)	MPI_Receive (n bytes)
MPI_Receive(n bytes)	MPI_send(n bytes)
Stop_Timer	
Next	***Next***

Figure 9. MPI Ping-Pong pseudo code

The Ping-Pong involves two nodes (A and B). Each of them runs an MPI task. Each node alternatively acts as a sender or a receiver. Time is measured starting from the MPI_Send call to the end of the MPI_Receive on one node. The resulting delay corresponds thus to the time necessary for a message to leave and return to a node. The latency is half the measured delay. Of course, such a measure is repeated several thousand times to obtain an average latency.

Figure 10 presents the different steps of an MPI Ping-Pong between two nodes. Basically, a message transfer between two MPI tasks can be decomposed in six steps : two software steps on the emitter (node A in figure 10), the network transfer, and three software steps on the receiver (Node B in figure 10).

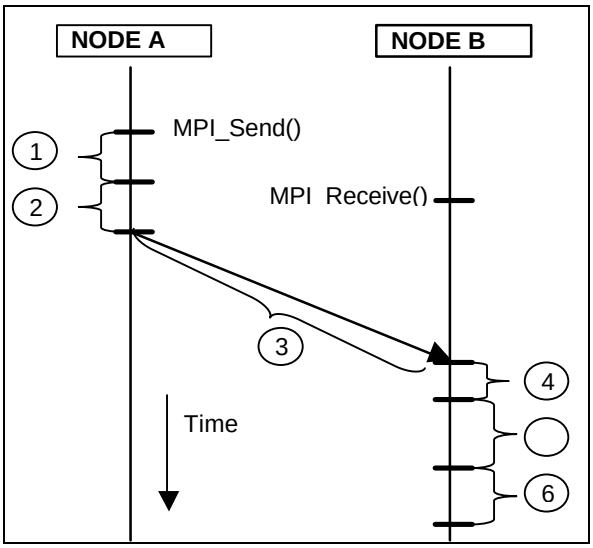

Figure 10. MPI task to task transfer schedule

The table below (table 1.) details the actions undertaken during a message transfer between two MPI tasks. Average latencies are indicated for each step.

Table 1. Latency decomposition in MPI message transfers

Step	Lat.	Actions
1. MPI emitting functions execution	7µs	▪ Data preparation & copy in the emission buffer
2. PUT invocation (emission)	4µs	▪ LPE add entry & network controller (NIC) notification
3. Hardware transfer	3µs	▪ LPE fetch by the NIC ▪ DMA read by the emitter NIC. ▪ Transfer on the HSL link ▪ DMA write by the receiver NIC. ▪ LMI update
4. Pooling loop	3µs	▪ LMI update detection
5. PUT invocation (reception)	2µs	▪ LMI entry fetch and MI interpretation ▪ Corresponding callback function call
6. MPI receiving function execution	7µs	▪ Data read from reception buffer and copy to the receiving MPI task.

In terms of execution, some steps are run in the user mode while the remaining is integrated in the OS. As shown in figure 11, throughout the execution flow, system calls are frequent. At least, 3 execution context switches are necessary during the transfer of a message between two MPI tasks running on two different nodes.

Given their frequency, systems calls are costly on latency due to context switches. Any reduction of system calls can contribute to a significant latency reduction. Many solutions are under study among which a new implementation of PUT to be run in the user mode is planned. As a consequence, all system calls are removed. Such a layer is under development and results are expected in the near future.

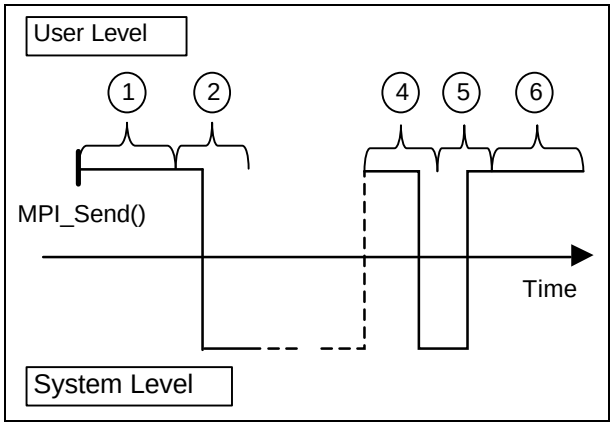

Figure 11. Context switches during an MPI transfer

6. Conclusions & perspectives

In this paper , MPC, a high performance and low cost parallel computer has been presented. This parallel computer is a PC-cluster built around the high speed HSL network using Gbit/s HSL links. FastHSL, a specific board to be plugged on each PC has been designed to connect each processing node to the HSL network. This board holds two ASICs : RCUBE which implements the routing function on the HSL network and PCI-DDC the network controller. This last implements a low-level protocol (DDSLR) using the "remote-write" primitive. It performs data transfers from the node memory to the HSL network (and vice versa) relieving thus the processor from communications.

From the software point of view, basically two zero-copy layers taking advantage of the parallel capability of the hardware have been presented. Results show a latency varying from 5 to 40µs and a throughput of 490Mbits/s.

The first generation of MPC is now available in 5 laboratories. The MPC computer of the LIP6 contains 8 processors and will soon be extended to 16. The software part of MPC is distributed as a freeware and is downloadable from the MPC web site[10]. FastHSL boards as well as their two specific components, i.e., PCI-DDC and RCUBE are now distributed by Tachys, a start-up of UPMC. RCUBE is now used for Gigabit and ATM switching fabrics. Such an evolution guarantees both the availability and the low cost of the MPC components and goes towards the cost reduction of parallel computers.

An MPI implementation on MPC has also been presented in this paper. Measures show a latency of 26µs and a useful throughput of 450Mbits/s. These encouraging results are similar to the MPI-BIP results, even though the MPI-MPC platform processor was less powerful. The implementation is still under evaluation, and significant improvements can be expected in the near future. Modifications of the PUT API include a drastic reduction of system calls.

Although, the performances of the MPC parallel computer and Myrinet are close, MPC differs significantly from its competitors by its distributed routing capability offered in RCUBE. This feature brings scalability and reduces the overall cost of the MPC parallel computer.

Regarding the hardware, two major research directions are currently investigated. The first concerns the development of new 2.5Gbits/s serial links. The second concerns the development of a programmable network controller that will replace PCI-DDC on the new generation of FastHSL boards. This will enable the evaluation of new protocols to improve the overall performance of the MPC parallel computer.

7. References

[1] IEEE 1355, IEEE1355 Standard for Heterogeneous Interconnect (HIC) Low Cost Low Latency Scalable Serial Interconnect for Parallel System Construction, (IEEE Standards Department, Aug. 1994).

[2] F. Wajsbürt, J.L. Desbarbieux, A. Greiner, C. Spasevski, S. Penain, An Integrated PCI component for IEEE 1355 Networks, In Proc. of EMMSEC'97, Florence, Italy, 1997.

[3] V. Reibaldi, Conception et réalisation d'un router de paquets à hautes performances, PhD thesis of University Pierre et Marie Curie, France, 1997.

[4] F. Potter, Conception et réalisation d'un réseau d'interconnexion à faible latence et haut débit pour machines multiprocesseurs, PhD thesis of University Pierre et Marie Curie, France, 1996.

[5] W. Gropp, E. Lusk, MPICH working note : Creating a new MPICH device using the channel interface, technical report, (Argonne National Laboratory, Chicago, 1995).

[6] http://www.myri.com

[7] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high performance, Portable Implementation of the MPI Message Passing Interface Standard, In Parallel Computing, 22(6), 1996, 789-828.

[8] L. Prylli, B. Tourancheau, R. Westrelin, Modeling of a High Speed Network to maximize throughput performance : the experience of BIP over Myrinet, In Proc. of Parallel and Distributed Processing Techniques and Applications, Vol. 2, Las Vegas, USA, 1998, 341-49.

[9] L. Prylli, B. Tourancheau, R. Westrelin, The design for a high performance MPI implementation on the Myrinet network, In EURO PVM/MPI'99, vol 1697 in LNCS, Barcelona, Spain, 1999, 223-230.

[10] http://mpc.lip6.fr