Efficient Inter-Device Data-Forwarding in the Madeleine Communication Library

Olivier Aumage Lionel Eyraud
Raymond Namyst
LIP, ENS-LYON
46, Allée d'Italie
69364 Lyon Cedex 07, France
{Olivier.Aumage,Lionel.Eyraud,Raymond.Namyst}@ens-lyon.fr

Abstract

Interconnecting multiple clusters with a high speed network to form a single heterogeneous architecture (i.e. a cluster of clusters) is currently a hot issue. Consequently, new runtime systems that are able to simultaneously deal with multiple high speed networks within the same application have to be designed. This paper presents how we did extend an existing multi-device communication library with fast internal data-forwarding capabilities on gateway nodes. On top of that, efficient high-level routing mechanisms can be implemented. Our approach is easily applicable to many network protocols and is completely independant from the application code. Efficiency is achieved by avoiding extra data copies when possible and by using pipelining techniques. The preliminary experiments show that the observed inter-cluster bandwidth is close to the one that can be delivered by the hardware.

1 Introduction

The current success of cluster computing in both academic institutions and compagnies leads many people to consider interconnecting several clusters to form powerful heterogeneous infrastructures for parallel computing. Indeed, this approach is very attractive as far as the performance/cost ratio is concerned. However, runtime systems for such architectures are still relatively experimental and their development raises many research issues. Among them, the design of the communication subsystem is perhaps the most delicate one. Because network links between clusters may be as fast as internal cluster links, clusters of clusters significantly differ from grid architectures where inter-cluster links are assumed to be slow. Consequently, communication environments that were designed for grids may not be suitable for clusters of clusters.

The network protocol used to handle inter-cluster communication on grids of computers is typically TCP/IP. For instance, environments such as PACX-MPI [5] use native implementations of MPI [2] to handle intra-cluster communication and use TCP for all inter-cluster communication. Obviously, this is not acceptable for fast clusters of clusters where all the links are able to deliver more than one gigabit per second. Actually, inter-cluster communication should also use efficient communication libraries such as MPI, VIA [7], BIP/Myrinet [9], SISCI/SCI [6], etc. On gateway nodes (i.e. machines connected to several physical network devices), it means that multiple communication libraries should be integrated in a common subsystem that would provide facilities for message forwarding between clusters. Nexus [3], the communication component of the Globus [4] environment, is a good example of such a runtime system. It features a multi-device communication layer that is able to exploit several networks simultaneously. However, Nexus does not provide any support for message routing between networks. It is up to the application to forward messages from one network device to another one, using regular receive and send operations. This raises two major problems: the routing is not transparent to the application and the data transfers are inefficient in terms of bandwidth since extra copies of data are performed and no pipelining techniques can be used.

This paper presents a fast data-forwarding mechanism that we have integrated into Madeleine, an existing multi-device communication library originally designed for high performance clusters. Contrary to approaches such as PACX-MPI which try to link separate communication libraries by an "interconnecting glue", our approach consists in designing a natively multi-device communication library able to efficiently transfer messages across devices. This way, the inter-device data-transfer mechanism is completely

hidden to the upper layers and some low-level characteristics of network devices can be used to optimize transfers (preallocated buffers, DMA operations, etc.) On top of Madeleine, high-level traditional routing mechanisms can easily and efficiently be implemented. Moreover, our approach is portable on top of many network protocols thanks to a low-level generic interface. The preliminary experiments we conducted using one Myrinet cluster connected to a SCI cluster (with a Myrinet link) revealed a maximum bandwidth of 49.5 MB/s between two nodes located on each clusters.

2 Portable and efficient multiprotocol forwarding

2.1 The Madeleine communication interface

Madeleine [1] aims at enabling an efficient use of the complete set of communication software and hardware available on clusters of workstations. It is able to deal with several networks (through possibly different interfaces) within the same application session and to manage multiple network adapters (NIC) for each of these networks. The library provides an explicit control over communication on each underlying network. The user application can dynamically switch from one network to another, according to its communication needs. Moreover, just like FAST-MESSAGES (FM) [8] or NEXUS [3], *Madeleine* allows applications to incrementally build messages to be transmitted.

Nowadays communication libraries are expected to fulfill two seemingly contradictory aims. They must achieve effective portability over a wide range of hardware/software combinations, whilst achieving a high level of efficiency using these network components. To meet these goals, the *Madeleine* design follows a modular approach leading to a highly flexible architecture. This approach allows the library to very tightly fit and optimally exploit the specific characteritics of each targeted network component.

2.1.1 Structure

Data Transmission Modules *Madeleine* is organized as two software layers (Fig. 1), following a commonly used scheme. *Protocol/network-specific interfacing* is realized by the lower *Hardware Abstraction Layer*, providing the portability of the whole library. This layer relies on a set of network specific *Transmission Modules* (TM). TMs are grouped into Protocol Management Modules (PMM). There is one PMM for each supported protocol (e.g., BIP/MYRINET or TCP/FAST-ETHERNET). Each PMM implements whole or part of a generic set of functions. This set of functions constitutes the protocol driving interface. It insures independence between the upper layer and the communication protocols.

Buffer Management The upper layer is independent of the supported network interfaces and is in charge of the *generic buffer management*. Indeed, while some TM can benefit from grouped buffer transfers, other may not, depending on the functionalities implemented by the underlying network. Each TM should thus be fed with its optimal shape of data. As a result, each TM is associated with a *Buffer Management Module* (BMM) from the buffer management layer. The set of BMMs constitute the upper layer of *Madeleine*. Of course, it is expected that several TMs share the same preferred data shape so that BMMs can be reused, which results in a significant improvement in development time and reliability.

A BMM may either control *dynamic buffers* (the user-allocated data block is directly referenced as a buffer) or *static buffers* (data is copied into a buffer provided by the TM), but not both. Moreover, each BMM may implement a specific aggregation scheme to group successive buffers into a single virtual piece of message in order to exploit optional scatter/gather protocol capabilities. On the contrary, a BMM may adopt an eager behavior and send buffers as soon as they are ready.

2.1.2 Interface

The *Madeleine* programming interface provides a small set of message-passing-oriented primitives. Basically, this interface provides primitives to send and receive *messages* and to allow the user to specify how data should be inserted into/extracted from messages. *Madeleine* is able to efficiently deal with several network protocols within the same session and to manage multiple network adapters (NIC) for each of these protocols. It allows the user application to dynamically switch from one protocol to another, according to its communication needs.

Communication objects Such a control is offered by means of two basic objects. The *channel* object defines a closed world for communication. A channel is associated with a network protocol, a corresponding network adapter and a set of *connection* objects. Each connection object virtualizes a point-to-point reliable network connection between two processes belonging to the session. It is of course possible to have several channels related to the same protocol and/or the same network adapter, which may be used to logically split communication. Yet, in-order delivery is only enforced for point-to-point connections within the same channel.

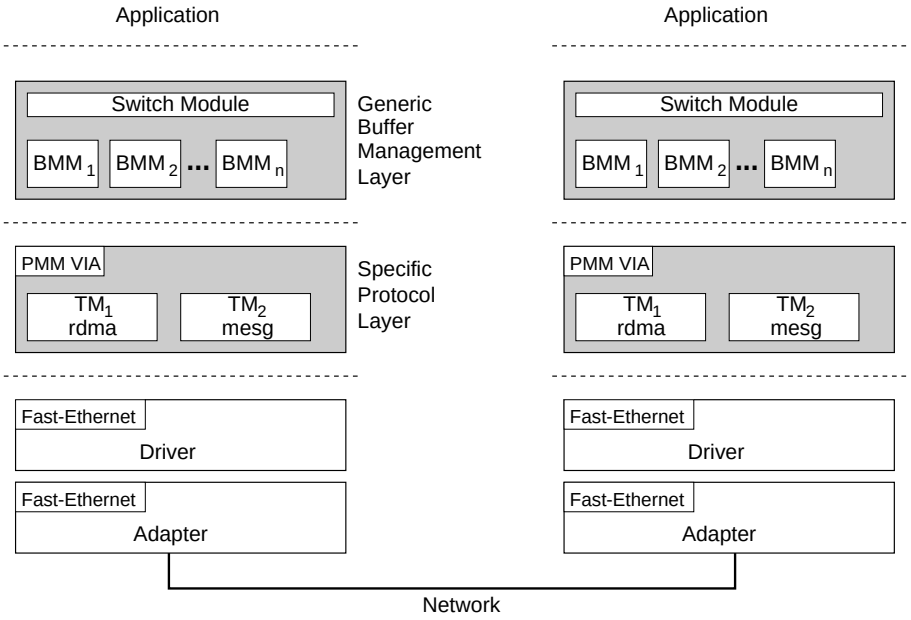

Figure 1. Madeleine's modular architecture.

Message Building A *Madeleine* message is composed of several pieces of data, located anywhere in user-space. It is initiated with a call to `mad_begin_packing`. Its parameters are the remote node *id* and the channel object to use for the message transmission. Each data block is then appended to the message using `mad_pack`. In addition to the data address and size, the packing primitive features a pair of *flag* parameters which specify the semantics of the operation. Eventually, the message construction is finalized using `mad_end_packing`. This final step guarantees that the whole message has been actually transmitted to the receiving side through the network.

One should note, though, that for the sake of efficiency, the messages are not *self-described* at the level of *Madeleine*: the data blocks should be *unpacked* precisely in the order as they were *packed*, with the same flag specification.

2.2 Portable and efficient multiprotocol forwarding on clusters of clusters

In order to allow *Madeleine* to be able to benefit from configurations including a larger number of nodes, which are not always connected altogether, but rather as clusters of clusters, we had to insert into *Madeleine* a forwarding mechanism. This mechanism had to be transparent to the user application and to keep the whole *Madeleine*'s portability while being as efficient as possible with regards to the capabilities of high-speed networks.

2.2.1 General description

User interface Our solution is mostly transparent to the user. The changes were indeed made inside *Madeleine*, thus keeping compatibility with the former applications. The only change in the interface is due to the necessity to describe more precisely the configuration of the network when a channel is created. Instead of simply creating a channel using a network protocol, we now create a *virtual channel* that includes a set of real channels. When sending a message over a virtual channel, the appropriate underlying real channel is dynamically chosen depending whether it is necessary to forward the message through a gateway or not.

Integration into *Madeleine* The first issue we had to face is related to the level at which we would implement our forwarding mechanism within the *Madeleine* layered architecture.

Because transmission Modules are strongly protocol-dependent, it is not possible to implement the forwarding mechanism at this level without compromising *Madeleine*'s whole portability. On the other hand, modifying the Buffer Management Layer could be an option, yet with a high development cost because of the number of modules in that layer. Furthermore, *conversion* modules between Buffer Management Modules would have to be written because the BMMs used on each side of a virtual channel may differ. Also, we could implement this mechanism just on top of *Madeleine*, which would perfectly meet our needs in terms of portability. Obviously, it would also have dramatic im-

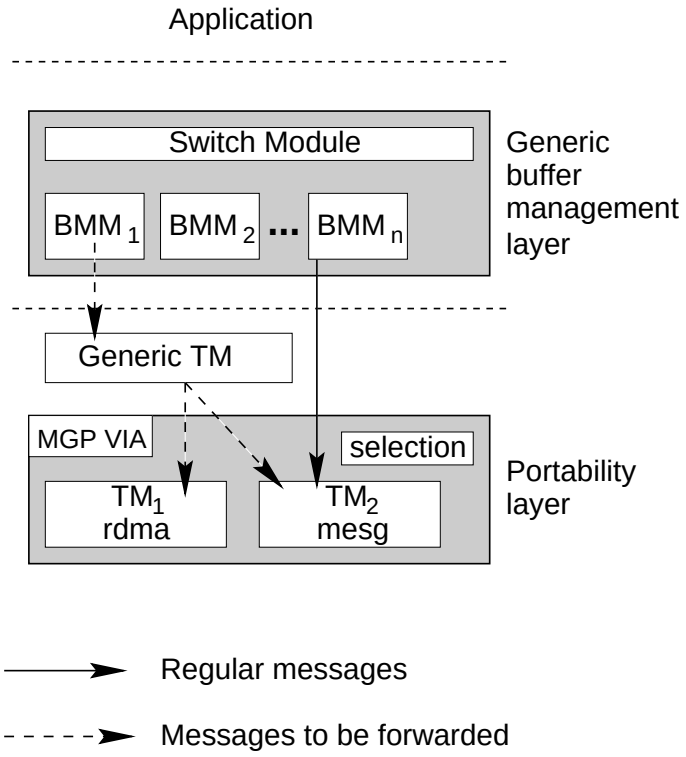

Figure 2. Generic Transmission Module integration into *Madeleine*: emission example.

pact on the efficiency because all data transfers would have to though *Madeleine* twice on the gateway nodes, incuring an overhead due to unavoidable extra copies in temporary buffers.

Hence, the best solution seems to be the one which would place the forwarding mechanism between BMMs and TMs. That said, the most efficient way of performing the actual data transfers would be to consider raw transfers between transmission modules. However, such a raw forwarding is impossible because *Madeleine* may use different BMMs for different network devices (in order to optimally exploit the characteristics of the underlying networks) and thus may group buffers in a different way for each device. As a result, when a message is to be transfered through two different networks, it would be very expensive for a gateway to ungroup buffers and then regroup those buffers in a different way. To solve this problem, we have designed a generic TM which is used for every message that has to travel through at least two different networks. This TM, used by both the sender and the receiver of a message as an interface between BMMs and real TMs (see Fig. 2), guarantees that data is handled in the same way on both ends. Of course, some optimizations are lost but, most importantly, the cost of ungrouping and regrouping buffers is saved.

The generic TM is also used to construct self-described messages by adding the information needed by the gateway to get the size and actual destination of a message. Within homogeneous *Madeleine* applications, this was not necessary since the user gave this information when receiving a message. Yet, this information is not available to the gateway, unless the code of that gateway is written as part of the application (precisely what we try to avoid). Hence, self-describing messages are mandatory to our transparent forwarding mechanism.

2.2.2 Detailed architecture

Virtual channels Because a gateway node is also a regular node that supports the execution of some application code, it must distinguish the messages destinated to itself from the ones that have to be forwarded. A good way of doing this is to send messages over two separate *Madeleine* real channels, depending on whether they should be delivered to the gateway itself or not. Since a *Madeleine* channel is related to a network device, there will be two channels per network device per virtual channel (see Fig. 3). The first one is used for messages that do not have to be redirected to another node (they are called *regular messages*), the second one is used for messages that cross the gateway.

Another issue concerns the choice of the appropriate real

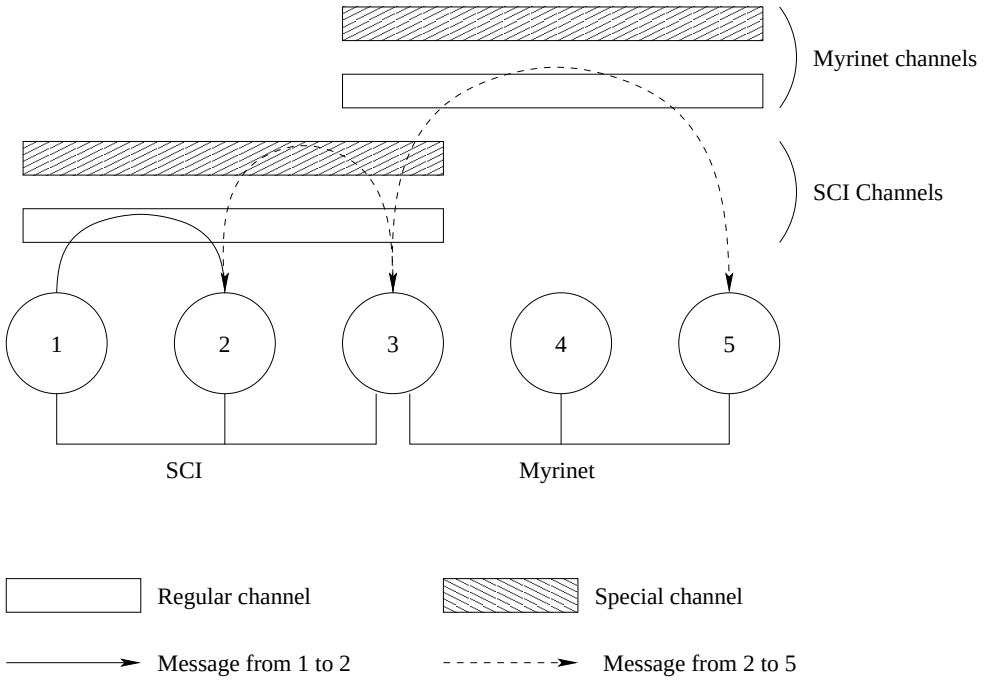

Figure 3. The different channels in a virtual channel

channel when messages are forwarded from the last gateway to their final destination. A possible solution is to use the special channels designed for non-direct messages. This would allow the receiver to distinguish easily between regular and special messages, so as to know if it has to use the GTM or not.

However, this solution suffers from two major drawbacks. First, regular nodes would have to poll two different channels when waiting for a message, depending on whether it arrives directly or through a gateway. And since channels are totally independent in *Madeleine*, this would involve a quite complex polling mechanism. Second, it raises the problem of distinguishing messages in configurations using multiple gateways. For instance, in a configuration with two gateways, it is possible that a message sent to the first gateway should be forwarded to the second one. If the destination gateway receives this message on the special channel, it will not be able to distinguish it from a message that has to be forwarded. The right solution thus consists in sending messages over a regular channel once they have crossed the last gateway (see Fig. 3).

As a counterpart, this policy does not allow the receiver to know whether the message it receives has been forwarded or not. To be able to chose between a regular Transmission Module and the Generic one, it needs some additional information. We chose to transmit this information before the actual message body transmission.

Message scrutation Another issue is raised by the fact that the sender of a message is unknown before the receive operation actually begins. Remember that the application sends messages over virtual channels that includes several real channels. On a regular node, there is only one useable protocol/network, so messages can only be received on one real channel. Things are different for a gateway node where at least two real channels can be used. A polling mechanism involving complex threads synchronizations has to be implemented so as to poll multiple networks at the same time.

There are also other threads on the gateway responsible of performing the forwarding of messages itself. These threads are listening to the special channels and are responsible for re-transmitting messages to their appropriate destination. The main issue is to let messages transit as fast as possible so as to optimally use the full bandwidth of the underlying networks. For improved efficiency, we have implemented a multithreaded pipeline mechanism for receiving and re-transmitting messages. Two threads are bound to each network and share two buffers. The first thread receives data into one buffer while the second sends data from the other buffer that has previously been filled. This allows the gateway to receive a data paquet while sending the one it has received at the previous step. The figure 4 illustrate this organization on a gateway between a SCI and a MYRINET network.

Note that this thread-based solution is even more inter-

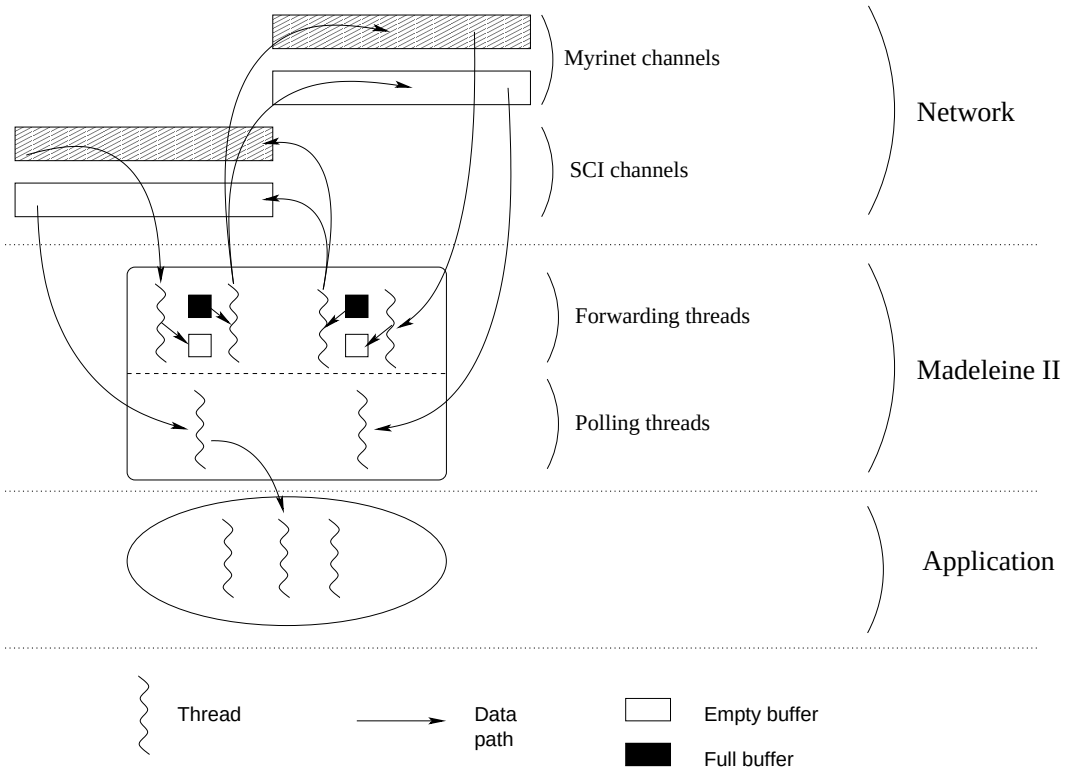

Figure 4. The threads running on a gateway node

esting on multi-processors machines where true parallelism can be achieved.

2.3 Implantation

Generic Transmission Module Because of the diversity of the BMMs in *Madeleine*, we have designed a specific Generic Transmission Module for the messages that have to be forwarded, so as to guarantee that buffers are grouped in the same way on both ends. This way, the gateway node does not have to group/ungroup buffers in several manners.

This Generic Transmission Module plays another role: it determines a MTU (Maximum Transmission Unit) so as to send messages with an optimal packet size for every network they go through. To optimally use the pipeline mechanism on the gateways, the messages have to be fragmented into several packets. The size of those fragments is defined so that each network is able to send them without having to fragment them further. Note that in our case, an appropriate paquet size can be chosen at compile time because the network configuration is statically configured.

Self-described messages The Generic Transmission Module is also used to add self-description information to the messages that go through the gateways. This description is compulsory for those messages, because a gateway knows nothing about what kind of messages it will receive. Since some information (like the destination of the message) is common to several buffers, it is sent only once, as part of the first packet. The protocol between the sender and the gateway (implemented in the GTM) is the following one:

- the sender sends the rank of the destination node, and the MTU used for this connexion;
- then, for each user buffer to be sent, the sender transmits the size and the emission and reception constraints and finally the buffer itself (fragmented into several pieces if necessary);
- to end a message, the sender sends the description of an empty message.

Minimizing copies In order to optimally use the capabilities of high-speed networks, one of our priorities is to avoid copying messages, which can take as much time as the reception of a message. This is easily done when dealing with dynamic buffer networks, but some networks like SBP [10] require data to be written in special buffers before being sent. In that case, using an additional temporary

buffer to receive data should be avoided. *Madeleine*'s support for static buffer protocols allowed us to easily implement a *zero-copy* mechanism. The *Madeleine* transmission modules may be asked for protocol/network-specific static buffers when a message has to be sent.

If the sending-side network uses static buffers while the network on the receiving side uses dynamic buffers, we only have to ask the outgoing protocol/network transmission module for a static buffer which we use to receive data into. Obviously, an extra copy is unavoidable when both networks require static buffers.

3 Evaluation

To evaluate the efficiency of our forwarding mechanism, we did perform several inter-cluster ping tests between a regular node (i.e. not the gateway node) of a Myrinet cluster and a regular node of a SCI cluster, the two clusters being connected together by a Myrinet link. Consequently, all these tests involved three nodes: the two endpoints and a gateway node that was equipped with both a Myrinet card and a SCI card and was running the forwarding code. More precisely, both clusters were composed of dual INTEL PENTIUM II 450 MHz PC nodes equipped with 128 MB of RAM and with a 33 MHz 32 bits PCI bus. The operating system kernel was Linux v2.2.13. The first cluster interconnection network was Myrinet (NICs specs: LANai 4.3, 32bit bus, 1 MB SRAM) and the second one was Dolphin SCI (D310 NICs). The underlying network interface used on Myrinet was BIP [9]. On the SCI network, we used the SISCI library provided by Dolphin.

3.1 Test programs

To get an accurate evaluation of our mechanism, we distinguished two cases: messages going from the SCI subnetwork to the Myrinet one and messages going the other way. In the following, we refer to "the *SCI-to-Myrinet* test" to describe the experiment that consisted in sending messages from one endpoint located within the SCI cluster to the other endpoint located within the Myrinet cluster. All the messages are obviously crossing the gateway. We refer to "the *Myrinet-to-SCI* test" to decribe the experiment that consisted in sending messages in the other direction.

For both tests, we obtained the performance of one-way transmission by using a single ping test. The ping program transmits messages of the desired size using the high speed networks in one direction (through the gateway) and transmits only a small ack over a Fast-Ethernet connection in the over direction. Since we exactly know the latency of the ack, it is easy to deduce the one-way message transmission time from the observed round-trip time.

3.2 Preliminary remarks

3.2.1 Bandwidth *vs* latency

Our forwarding mechanism is basically designed to provide a high bandwidth when transmitting messages over clusters of clusters connected by high speed networks. In this context, we obviously are not expecting interesting latencies by using such a mechanism. The first reason is that we did not perform a lot of code optimizations in the current implementation, so the resulting latency of a inter-cluster transmission not only includes the native latencies of each networks, but it also includes a significant amount of software overhead. The second reason is that we did not implement any particular policy that would minimize the pipeline startup time: the gateway always manipulate paquets of the same size. For this reason, in the following figures, we only provide bandwidth numbers to discuss the efficiency of our approach.

3.2.2 On the paquet-forwarding pipeline

On the gateway node, our implementation uses two threads to pipeline the re-transmission of paquets. Ideally, the best performance is achieved when the sending and the receiving of paquets takes approximately the same time (Figure 5) and if the software overhead incured when the threads exchange their buffers is neglectible. In this case, the first buffer can be sent while the second buffer is received, and then the second one can be sent while the first one is received, and so on.

However, if the gateway node bridges two different networks (as it is the case in our experiments), the corresponding transmission times may differ for a given paquet size. For instance, SCI achieves very good performance for small messages whereas Myrinet competes better for large messages. In fact, *Madeleine* achieves approximately the same performance on top of Myrinet and SCI for messages of size 16 KB (latency = $250\mu s$, bandwidth = 60 MB/s), which suggests that the correct paquet size should be set to 16 KB. Unfortunately, we will see in the next sections that several other factors have an impact on the behaviour of the pipeline, making it quite difficult to predict its performance.

3.3 The SCI-to-Myrinet experiments

We first did some experiments to measure the performance of our forwarding mechanism in the *SCI-to-Myrinet* direction. We have evaluated the bandwidth achieved for several paquet sizes: Figure 6 reports the results for paquet sizes going from 8 KB to 128 KB. As one can see, the asymptotic bandwidth obtained when using 8 KB paquets is only 36.5 MB/s. But for larger paquets, the obtained asymptotic bandwidth is greater than 45 MB/s and

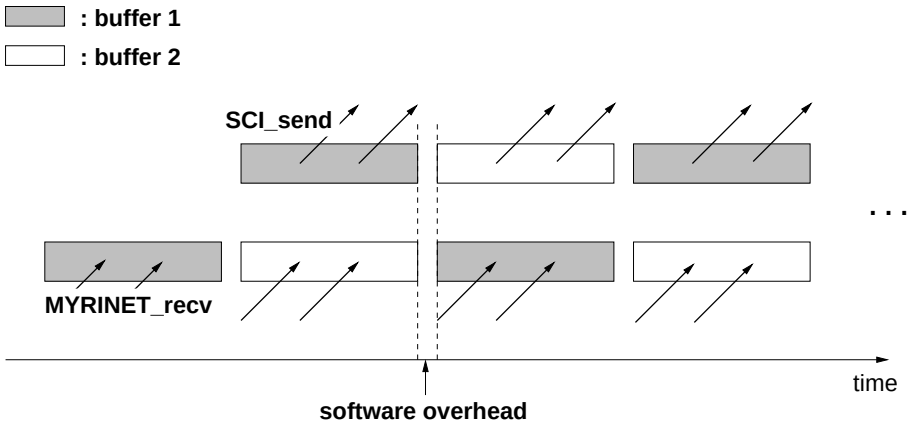

Figure 5. The paquet-forwarding pipeline on the gateway node.

even close to 50 MB/s for 128 KB paquets. This can be considered as a very good result, since the theoretical maximum bandwidth one can achieve on a machine equipped with a single 33 MHz PCI bus is 66 MB/s.

3.3.1 Discussion

For 8 KB paquets, a pure *Madeleine* ping-pong program achieves a bandwidth of 58 MB/s over SCI and 47 MB/s over Myrinet. Thus, the period of the pipeline (i.e. the duration of a single step) for 8 KB paquets is at least $166\mu s$. In practice, the observed bandwidth of 36.5 MB means that the effective pipeline period is rather $215\mu s$. This seems to indicate that the software overhead that we pay at each buffer switch is almost $50\mu s$, which is not neglectible.

For larger paquets however, another phenomenon appears. Indeed, for paquet sizes greater that 16 KB, a pure *Madeleine* ping-pong program achieves a bandwidth of more than 60 MB/s, which can be considered as the maximum one-way bandwidth one can get over a 32 bits PCI bus in practice. So it appears that the incoming paquets occupy so much place on the bus that the outgoing paquets cannot be sent with a bandwidth greater than 60 MB/s. However, we only reach an asymptotic bandwidth of 49.5 MB/s in practice: we assume that this is due to some conflicts appearing on the PCI bus when doing intensive full-duplex communications.

3.4 The Myrinet-to-SCI experiments

We did conduct the same experiments in the "*Myrinet-to-SCI*" direction. Figure 7 reports the results for paquet sizes going from 8 KB to 128 KB. One can see that the obtained performance are by far lower than the one obtained in the opposite direction. The asymptotic bandwidth obtained when using 8 KB paquets is only 29 MB/s, and the asymptotic bandwidth obtained for larger paquets never exceeds 36.5 MB/s!

Obviously, such results are not simply due to some software overhead nor to the saturation of the PCI bus. We thus conducted some additional measurements to explain such a poor performance.

3.4.1 Discussion

To understand the behaviour of the pipelining algorithm in the *Myrinet-to-SCI* experiments, we have instrumented the low-level code in *Madeleine* that deals with message receiving on Myrinet and message sending on SCI. We have used the Intel Pentium specific internal clock-tick register (`rdtsc`) to get very accurate timings. After having rerunned our tests again, we have discovered that our pipeline mechanism was no longuer able to perform a complete message sending and a complete message receiving simultaneously.

Over Myrinet, a receive operation consists in a series of DMA PCI transactions initiated by the network interface card upon message arrival. Over SCI, a send operation consists in several PIO PCI transactions initiated by the processor. These transactions are accelerated by the use of a Write Combining buffer, which groups the data in order to perform PCI transactions using chunks of size 128 bytes.

According to our measurements, it appears that DMA PCI transactions initiated by the Myrinet card have a greater priority than PIO PCI transactions initiated by the processor. Consequently, during a (Myrinet-) buffer receiving, the sending of the other buffer over SCI is slowed down by a factor of two, while the receiving operation itself is evolving at the nominal speed. As illustrated on Figure 8, this results in the sending steps lasting a lot more time than the receiving steps.

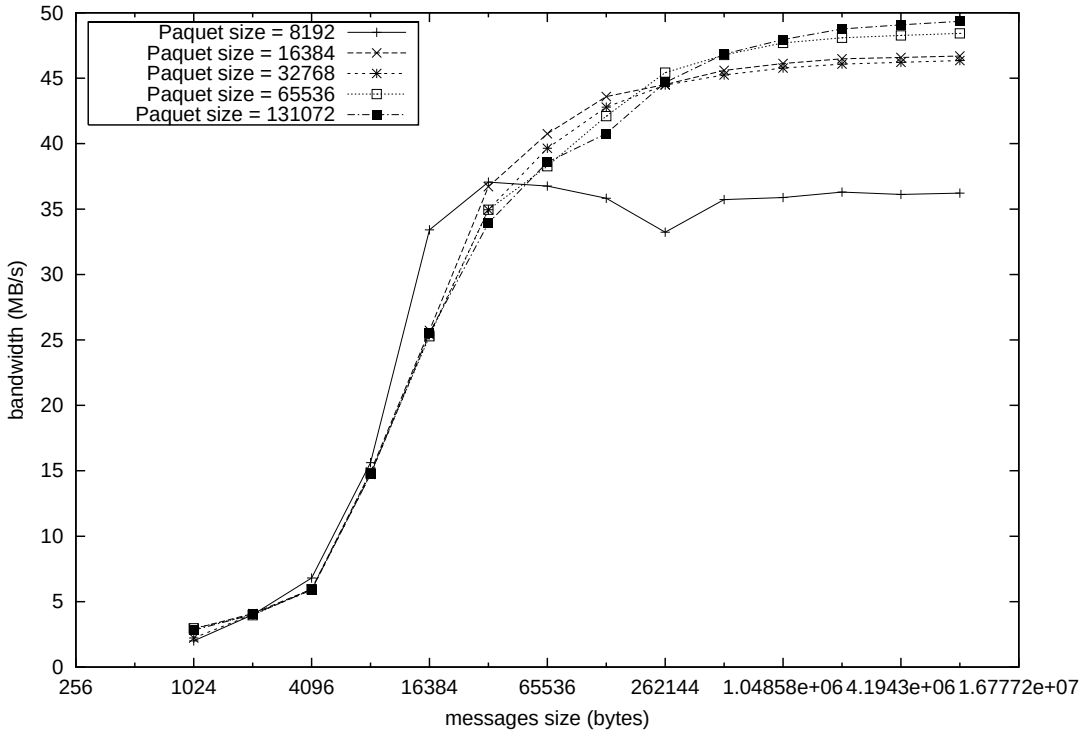

Figure 6. Madeleine's multiprotocol forwarding bandwidth when messages are coming from a SCI network and are going to a Myrinet one.

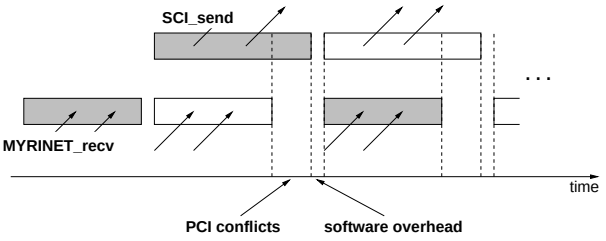

Figure 8. PCI bus conflicts and software overhead may strongly decrease the performance of the pipeline.

For 16 KB paquets, for instance, the sending operation lasts $460\mu s$ instead of $250\mu s$! In fact, the sending operation itself lasts appromimately $410\mu s$, and one must add the $50\mu s$ of software overhead that arise at each buffer switch.

As of now, we do not know if it will be possible to avoid such a phenomenon on the PCI bus. However, we are currently investigating several work-around solutions, such as using the SCI DMA engine instead of PIO operations to send buffers over SCI.

4 Conclusion and future work

We presented an efficient and yet portable data-forwarding mechanism for network-heterogeneous clusters of clusters that we integrated into the Madeleine multi-device communication library. We showed that by integrating such a mechanism at the right abstraction level, it can be completely transparent from the application point of view and portable on a wide range of network protocols while remaining efficient. In particular, we did demonstrate that zero-copy mechanisms together with pipelining techniques are mandatory to keep a high bandwidth over inter-cluster links.

The preliminary experiments we conducted between a Myrinet cluster and a SCI cluster confirmed that our approach can deliver an important part of the performance achieved by the underlying hardware. However, some one-way communication experiments revealed that the sharing of the gateway internal system bus bandwidth seems to be an important issue. More precisely, it seems that some sophisticated "bandwidth control" mechanism is needed to regulate the incoming communication flow on gateways. This is a point we intend to investigate in the future.

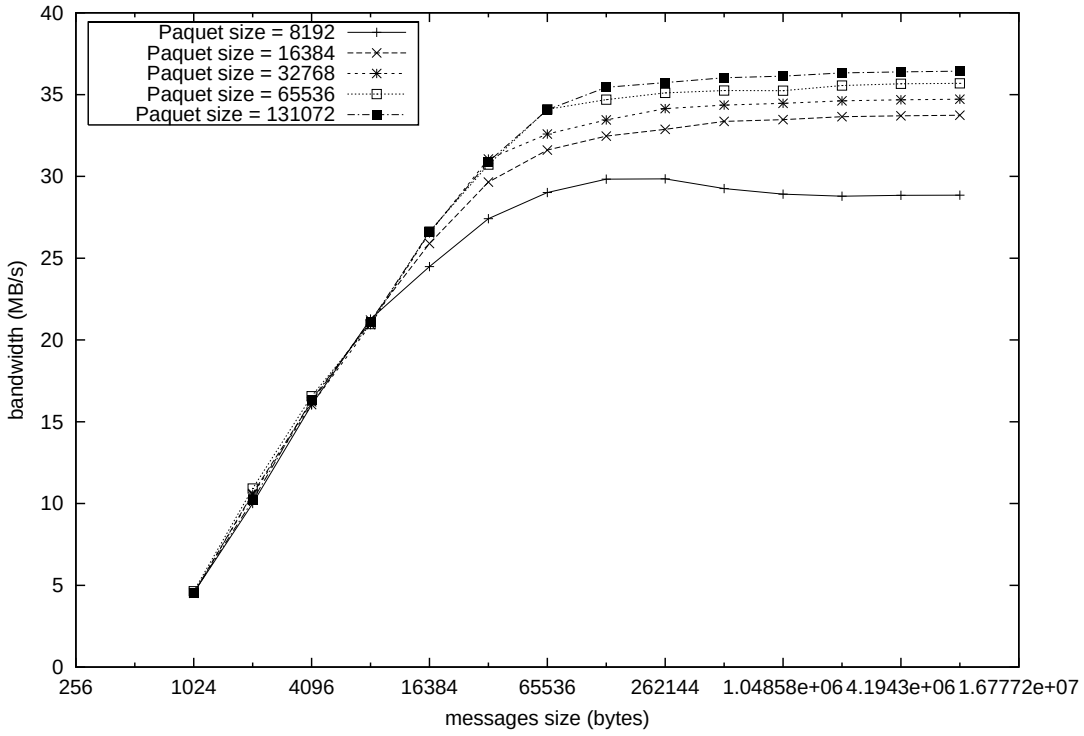

Figure 7. Madeleine's multiprotocol forwarding bandwidth when messages are coming from a Myrinet network and are going to a SCI one.

References

[1] Olivier Aumage, Luc Bougé, and Raymond Namyst. A portable and adaptative multi-protocol communication library for multithreaded runtime systems. In *Parallel and Distributed Processing. Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP '00)*, volume 1800 of *Lect. Notes in Comp. Science*, pages 1136–1143, Cancun, Mexico, May 2000. Held in conjunction with IPDPS 2000. IEEE TCPP and ACM, Springer-Verlag. Electronic version available.

[2] Jack Dongarra, Steven Huss-Lederman, Steve Otto, Marc Snir, and David Walker. *MPI : The Complete Reference*. The MIT Press, 1996.

[3] I. Foster, C. Kesselman, and S. Tuecke. The Nexus approach to integrating multithreading and communication. *Journal on Parallel and Distributed Computing*, 37(1):70–82, 1996.

[4] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Super-Computer Applications*, 11(2):115–128, 1997.

[5] Edgar Gabriel, Michael Resch, Thomsa Beisel, and Rainer Keller. Distributed Computing in a Heterogeneous Computing Environment. In Vassil Alexandrov and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Lecture Notes in Computer Sciences. Springer, 1998.

[6] IEEE. *Standard for Scalable Coherent Interface (SCI)*, August 1993. Standard no. 1596.

[7] Xin Liu. Performance evaluation of a hardware implementation of VIA. Technical report, U. of California in San Diego, 1999.

[8] S. Pakin, V. Karamcheti, and A. Chien. Fast Messages: Efficient, portable communication for workstation clusters and MPPs. *IEEE Concurrency*, 5(2):60–73, April 1997.

[9] Loïc Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In *1st Workshop on Personal Computer based Networks Of Workstations (PC-NOW '98)*, volume 1388 of *Lect. Notes in Comp. Science*, pages 472–485. Held in conjunction with IPPS/SPDP 1998, Springer-Verlag, April 1998.

[10] R.D. Russell and P.J. Hatcher. Efficient kernel support for reliable communication. In *13th ACM Symposium on Applied Computing*, pages 541–550, Atlanta, GA, February 1998.

Lionel Eyraud is a student at the École Normale Supérieure de Lyon. He is the designer of the fast forwarding mechanism implemented within the *Madeleine* communication library.

Olivier Aumage is a PhD. Student at the LIP computer science laboratory (École Normale Supérieure de Lyon, France). He is

the main designer of the *Madeleine* message passing communication library. The *Madeleine* library is part of the PM^2 multithreaded distributed programming environment (`http://www.pm2.org`).

Raymond Namyst holds an assistant professor position at the LIP laboratory (École Normale Supérieure de Lyon, France). He received his PhD in Computer Science with honours from the University of Sciences and Techniques of Lille in 1997 and his MS in C.S. with honours from the Lille University in 1993. His research interests include parallel programming environments and multithreaded runtime systems for high speed networks. See `http://www.ens-lyon.fr/~rnamyst` for further information.