MPICH/MadIII: a Cluster of Clusters-Enabled MPI Implementation

Guillaume Mercier, Olivier Aumage

MPICH/MADIII : a Cluster of Clusters Enabled MPI Implementation

Olivier Aumage* Guillaume Mercier†

Abstract

This paper presents an MPI implementation that allows an easy and efficient use of the interconnection of several clusters, of potentially heterogeneous nature (as far as the network is concerned). We describe the underlying communication subsystem used. The mechanisms within MPI that inform the user of the underlying topological structure are detailed. The performance figures obtained with this MPI implementation are discussed and advocate for the use of such a solution on this particular type of architecture.

1 Introduction

The recent worldwide trend of the high-performance distributed computing community towards providing Grid-enabled tools resulted in a great deal of hard technical issues being tackled over the last few years, despite the large geographical scale and the possibility of multiple administrative domains of the Grid computing running environments that make it hard to perform any progress.

One way to circumvent those difficulties is to look at a Grid testbed as a hierarchical structure. With that frame of mind, we can recursively define a grid as either a local cluster (the terminal leaf) or an aggregate of smaller grids (the branches of the hierarchy).From a purely technical point of view, interconnecting two grids A and B (as previously defined) into a larger grid is hence only a matter of adding one NIC to one machine (therefore used as a gateway) of a cluster of the grid A and then linking it to a switch of a cluster of grid B. In this paper, we will only focus on the first step of building such a hierarchy, that is, the case where A and B are local clusters and the objective is to build a grid out of A and B.

Considering that situation, one can see that if clusters A and B feature different high-performance networks such as Myrinet for A and Gigabit-Ethernet for B, communicating from A node to a B node will require multi-network data transfers. However, existing Grid-enabled MPI implementations do not feature multi-network point-to-point communication capabilities and would be forced to sub-optimally fall back to the use of TCP.

We showed in a previous work ([3]) that it is possible to efficiently control multiple networks within a single MPI session, that is, instead of relying on various MPI implementations to try and cooperate in an inter-operable manner, our proposal makes use of a single MPI instance *generically* and *efficiently* (i.e. in conformance to what users usually expect from an MPI implementation) controlling any available underlying network. The success of this approach has since lead us to enhance our work towards implementing carefully selected extensions of the MPI standard, in order to allow our multi-network MPI implementation to actually provide applications with multi-cluster configuration support. The design of our multi-cluster MPI also takes advantage of the cluster management capabilities of the *Madeleine III* communication library (*Madeleine* for short in the remaining of this paper) for session spawning and multi-network automatic forwarding in conjunction with the advanced polling features of the *Marcel* user-level thread library for providing high-performance inter-cluster networking on multi-cluster based architectures. Our work can thus be seen as the high performance foundation block for building the first level of complex grid hierarchies, that is, the complementary tool of inter-operable solutions such as MPICH-G2([1]), which uses local MPIs at the cluster level. The remaining of this paper is organized as follows: Section 2 exposes an overview of the underlying tools we used and presents the overall design of our proposal. We then go through more details over the actual management of multi-cluster session within MPI in Section 3. Section 4 evaluates our MPI implementation and provides corresponding benchmark results. Section 5 concludes this paper.

2 System Architecture

In this section, after a short introduction about the MPI implementation called MPICH, we detail the software elements (namely *Madeleine* and *Marcel*) used for implement-

*Vrije Universiteit, Faculty of Sciences, Division of Mathematics and Computer Science, De Boelelaan 1081A, 1081HV AMSTERDAM, The Netherlands

†LaBRI, Université Bordeaux 1, Bordeaux, France. Contact: {guillaume.mercier@labri.fr}

ing a specific piece of software, called a *network device* according to the MPICH terminology. We also put the emphasis on some *Madeleine* and *Marcel* useful abilities from which we took advantage of.

2.1 MPICH as a Basis

Our previous work was based on the popular MPI implementation called MPICH [7]. We designed an MPICH device (*ch_mad*) that acted as an intermediary layer between MPICH's Abstract Device Interface ([12], [11]) and *Madeleine III*'s previous version, that is, *Madeleine II*[2]. This device was used to process inter-node communication, while two other devices, namely *smp_plug* and *ch_self* were used to handle respectively intra-node and intra-process communication. This device was multi-threaded because it was an effective and convenient way to handle several networks within the same device but it also became mandatory due to some *Madeleine* specificities (blocking communication operations for instance). No changes to the ADI were made in order to preserve portability as much as possible but a study of the ADI structure showed that it was not fully conceived to support multi-threaded devices.

2.2 Current Design

2.2.1 Overview

Eventually, the design of MPICH/MADIII is based on both customized ADI and upper MPICH layers. Despite those modifications, the software's architecture still follows an MPICH-like design, as shown in Fig. 1. The core of the

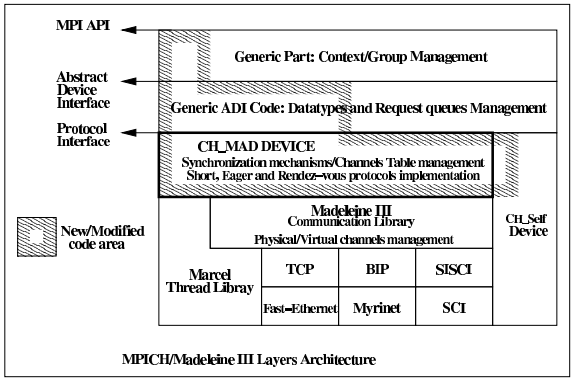

Figure 1. MPICH/MADIII Architecture

system still relies on the *ch_mad* device, but this time built upon the *Madeleine III* communication library. The *ch_self* device has partly been incorporated into *ch_mad* and no other MPICH device can be used while *ch_mad* is running, because of incompatibilities of the message polling mechanisms employed by a multi- threaded device and a regular "sequential" MPICH device.

2.2.2 The *Madeleine* Communication Library

Madeleine is a distributed computing environment providing a set of libraries and tools dedicated to launch and support session management and communication for applications running on top of clusters and multi-cluster configurations. The *Madeleine* communication interface is message passing oriented and provides an incremental message building/extraction paradigm. Specific services and/or behavior from the communication subsystem can be required on a piecewise basis by the programmer. The programming interface of *Madeleine* makes use of two main notions, namely the *connection* and the *channel*.

Connections The *Madeleine* connection object is an abstraction of a one-way point-to-point communication link between two nodes. Messages on a single connection are FIFO-ordered.

Channels The channel object of *Madeleine* represents an abstraction of a network: a set of nodes linked one to the other by connections: the connections are the edges of the channel's graph. The actual network may either be a *physical* network (made of plain wires) for regular channels, or a *virtual* network built on top of one or more physical networks. In that latter case, the corresponding channel is said to be *virtual*. It "owns" one ore more regular channels. A transparent support is provided to handle data transmissions over virtual connections spanning multiple networks. Messages are forwarded using a multi-threaded software module. This module strives to avoid unnecessary additional copies by using the same buffer for receiving and reemitting a given packet of data when possible. The use of a multi-threaded approach allows to pipeline reception and retransmission steps.

2.2.3 The *Marcel* thread library

The *Marcel* multi-threading library is a user-level library characterized by its ability to feature extremely low cost thread management operations. It relies on an *hybrid* thread scheduling mechanism (which manages pools of user-level threads on top of system lightweight processes) to allow for an extensive use of SMP nodes.

Polling with *Marcel* Besides from the core multi-threading features, the *Marcel* thread library provides a specific interface for efficiently and reactively supporting polling operations in multi-threaded contexts. It basically

allows to aggregate network events and more generally I/O events monitoring requests and to perform fast event monitoring operations on context switches boundaries ([4]). This solution allows to schedule the corresponding requesting thread immediately upon event reception, without having it to wait for its next scheduling slot, therefore both ensuring a good level of reactivity and a bounded event-delivery time. The *Madeleine* communication library is specifically designed to make full use of this polling support when compiled in combination with the *Marcel* multi-threading library. The immediate benefits are two-fold. First, we get a much better trade-off between preservation of the application computing CPU time and I/O event reactiveness *without* the painful polling delay fine tuning process. And second, we are completely relieved of those unreliable `yield` calls which — as real life experience inevitably shows — never seems to want to schedule that application computing thread desperately waiting for the CPU.

2.3 Transfer Modes

We now explain the functioning of our device: the various protocols implementation as well as message structure are described. The *ch_mad* device implements three transfer modes with one or several *Madeleine* messages where each message is composed of a sequence of one or two *Madeleine* packing operations. The description of the different transfer modes follows:

- the *short* mode: for messages up to 16 bytes, a buffer located in the message header is filled with the data. This operation is performed by sending a single message composed of a single *Madeleine* packing operation.
- the *eager* mode: for messages which size is larger than 16 bytes but smaller than a network-specific threshold, the data are sent by a single message composed of a sequence of two packing operations: one for the header and one for the user data buffer. When using this transfer mode, at *most* one intermediary copy on the receiving side is performed.
- the *rendez-vous* mode: for messages which size exceeds the selected network-specific threshold, a request message, just composed of a header, is sent. When the receiving side is ready to get the data, it sends at its turn an acknowledgement to the sending side. Receiving such a message triggers the emission of the user data, with a sequence of two *Madeleine* packing operations (header and data itself). When using this mode, *no* intermediary copy is performed.

The most appropriate transfer mode is selected dynamically according to the message size to send but also to the network type, which allows a better tuning and performance. These different modes are also used by the ADI layer to provide the different MPI communication modes (blocking, non-blocking, synchronous, etc.).

2.4 Related Work

It is to be noted that another MPI implementation features a similar architecture : MPICH-SCore. It is based on the SCore cluster operating system, which is built above the PMv2 communication library [15]. The authors claim in [15] that this communication library is able to use several networking technologies at the same time, but no performance evaluation in a heterogeneous context was to be presented in the article.

3 Multi-Cluster Management with MPICH/MADIII

This section describes the mechanisms implemented in our software in order to push the topology knowledge up to the outermost layers of MPICH. To achieve this goal, we rely on the following key-points: the use of configuration files containing both network and channel information for *Madeleine* and the introduction of a new set of tools in MPI.

3.1 Multi-Cluster Support with *Madeleine*

We describe here the facilities offered by the *Madeleine* communication library in the multi-cluster management department. For explanation purpose we suppose that two three-nodes clusters at are our disposal: the *foo* cluster with a SCI interconnection network and the *goo* cluster with a Myrinet interconnection network. The protocols used are SISCI and BIP ([14]) respectively.

3.1.1 Channels Building

The different channels mechanisms provided by *Madeleine* constitute a practical means to virtualize a cluster of clusters configuration. All the relevant pieces of information are contained in two configuration files written by the user: a network configuration file and a channels configuration file. For instance, a network configuration file could be:

```
networks : ({
       name  : tcp_net;
       hosts : (foo0,foo1,foo2,
                goo0,goo1,goo2);
       dev   : tcp;
   },{
       name  : sci_net;
       hosts : (foo0,foo1,foo2);
       dev   : sisci;
   },{
```

```
        name  : bip_net;
        hosts : (foo2,
                 goo0,goo1,goo2);
        dev   : bip;
        mandatory_loader : bipload
});
```

In that particular case, we suppose that the node *foo2* features both interconnection networks and thus can be a potential gateway between both clusters. The names of the networks may freely chosen, but the NIC names (`dev`) are fixed.

A *Madeleine* channel is an object that virtualizes a physical network. Several channels can even be built over the same physical network. This type of channels is called in the *Madeleine* terminology a *physical* channel. Physical channels are used as a basis to build *virtual* channels. A *Madeleine* application makes no distinction between such physical and virtual channels. So, if a physical channel is used to represent a cluster, an interconnection of (potentially different) clusters can easily be represented by a virtual channel. In our example, we might declare the following channels:

```
application : {
  name     : sample;
  flavor   : mpi-flav;
  networks : {
  include  : mynetworks.cfg;
     channels : ({
       name : tcp_channel;
       net  : tcp_net;
       hosts : (foo0,foo1,foo2,
                goo0,goo1,goo3);
  },{
       name : sci_channel;
       net  : sci_net;
       hosts : (foo0,foo1,foo2);
  },{
       name : sci_channel2;
       net  : sci_net;
       hosts : (foo0,foo1,foo2);
  },{
       name : bip_channel;
       net  : bip_net;
       hosts : (foo2,
                goo0,goo1,goo2);
  },{
       name : bip_channel2;
       net  : bip_net;
       hosts : (goo0,goo1,goo2);
  });
       vchannels : {
       name : default;
       channels :
       (sci_channel, bip_channel);
       };
  };
};
```

In this file, we declare five physical channels: one over TCP, in which all nodes are part of, and two channels over both SCI and BIP. In the latter case, one of the two channels (`bip_channel`) contains a supplementary node, *foo2*. We then construct a virtual channel based on the two physical channels `sci_channel` and `bip_channel`.

A physical channel used to build a virtual channel cannot be directly used anymore: the existence of the physical channel is hidden to the application. That's the case for both the `sci_channel` and `bip_channel` channel. This is the reason why we're declaring two physical channels over each of the high-performance networks: we want to keep as much as possible topological information about the different clusters.

3.1.2 Forwarding Mechanism

The channels mechanism can be utilized to either create a cluster of clusters or to create several node partitions within a cluster. But in order to get a good level of performance, a forwarding mechanism between networks has to be activated. This is set automatically by *Madeleine* by an analysis of the channel configuration file when an application is launched. For instance, if a node belongs to both a channel virtualizing a SCI network and a channel virtualizing a Myrinet network, this node will play the role of a gateway between both channels.

In our example, since the *foo2* node belongs to both channels used to build the virtual channel, all communications from one cluster to the other will be forwarded by this node (when using the relevant channel). As for intra-cluster communication, the local high-performance network will be used. In that case, we give the programmers two ways to perform inter-cluster communication: communication through TCP still remains possible, but also communication through a "virtual heterogeneous" high-performance network can be performed.

3.2 Interfacing MPI with *Madeleine*

We rely on those both mechanisms to integrate topology information into the upper MPICH layers and the MPI interface. Applications programmed with MPICH are regular *Madeleine* applications: the configuration files are used to convey the topological information while channels act as cluster virtualizations.

3.2.1 Channels/Communicators Matching

Since a *Madeleine* channel can be understood and seen as a variant of an MPI *communicator* (in the *Madeleine* world), the natural and intuitive idea is to create associations between a MPI *communicator* and a such a channel. As those communicators will be public objects and available at the

MPI user's level, we thus offer to the programmers the possibility to access the whole set of *Madeleine* channels (both physical and/or virtual). As a consequence, the underlying topology can be fully utilized.

During the initialization phase, we create all the communicators that can potentially be used by the programmers. Several different communicators may be bound to the same channel. As for the `MPI_COMM_WORLD` communicator, it is bound to the *default channel*. This channel plays a particular role and is mandatory for a program to run. The only requirement of this channel is that it has to encompass all the nodes of a given configuration; it can be either physical or virtual.

In our example, both the physical TCP channel and the virtual channel can be attached to `MPI_COMM_WORLD`. In order to choose the channel which will effectively be bound to this communicator, we name the channel "default".

To sum up, anytime a process is performing a communication operation over the `MPI_COMM_WORLD` communicator:

- If the receiving process belongs to the same cluster, the interconnexion network of that cluster will be used.
- If the receiving process belongs to the other cluster, the message will be sent to the gateway (*foo2*) with the local high-performance network; the message will then be forwarded to the right destination by the gateway and the other high-performance network will be employed. This operation is performed transparently to the user by *Madeleine*.

This last point constitutes the major difference compared to inter-operable based solutions, which usually favor TCP for inter-cluster communication, or even consider a cluster of clusters as a huge TCP interconnected cluster, neglecting the use of the different local high-performance hardwares ([9] for instance).

3.2.2 Description of the Interface

Since each communicator is associated to its dedicated channel, we can use the set of already existing MPI primitives that allow a user to manipulate communicators in order to create the needed objects. With such a scheme, the users do not have to take care of the problem of the underlying topology and the corresponding communicators. Moreover, the creation of new primitives is unnecessary and the set of MPI extensions remains minimal.

Even though the number of extensions is limited, they are necessary to handle easily topological information directly at the MPI level. The two important introductions are:

- `MPI_USER_COMM`: an array of communicators. The communicator called `MPI_USER_COMM[i]` is attached to the i-th channel seen by the application as ordered in the relevant configuration file.
- `MPI_COMM_NUMBER`: an integer corresponding to the number of available channels. It corresponds to the size of the `MPI_USER_COMM` communicator array.

The particular `MPI_COMM_WORLD` communicator is also a member of the `MPI_USER_COMM` array and its index is `MPI_COMM_WORLD_INDEX`. The index of a given communicators within the array can differ from one node to another. This is due to the fact that the set of available channels isn't likely to be the same on each node.

The same kind of extensions do also exist in MPICH-G2 (that is, the Globus-based MPICH) in order to provide simply for topological information. The mechanisms involved also take advantage of the MPI communicators and the information are used to optimize collective operations ([10]).

3.2.3 MPI Programs Portability Issues

It is very important to note that the use of these extensions isn't mandatory and that existing regular MPI programs can also take advantage from an heterogeneous configuration. The extensions are solely provided to simplify the use of a given configuration (as previously described, for instance) directly at the MPI level. As far as network use is concerned, a programmer can choose between several solutions :

- The programmer can declare only one virtual channel based on two physical channels (one for each type of high-performance network). In that case, the virtual channel will play the role of the *default channel* since it encompasses the whole set of nodes. The TCP network doesn't have to be used and no TCP channel has to be declared. Thus, any MPI program will be able to run on the heterogeneous configuration : the forwarding mechanisms are completely transparent to the MPI layers and handled at the *Madeleine* level. The extensions don't have to be employed since only one network is seen from the MPI layers. This network can be labeled as both virtual and heterogeneous.
- The programmer can declare several physical channels (one for each type of network). In that case, only one channel can be designated as the *default channel* : that is, the TCP channel. The extensions may be utilized to access the different channels (and thus the underlying networks). The user is bestowed the responsiblity of choosing the best network to perform a communication. It is even possible to implement a forwarding mechanism at the MPI level in order to only use

high-performance networks and bypass the TCP network. But these forwarding mechanisms will be much costlier than those of *Madeleine*. The advantage of such a solution is that the underlying topology isn't hidden anymore to the MPI layers.

So, the choice depends upon the necessity of accessing or not the topological information at the MPI level. It is not compulsory, neither is the use of our extensions. But in both schemes, we want to guarantee that a high level of performance can be achieved. In practice, the advantages of both schemes can be combined by declaring several physical channels over the high-performance networks and a virtual channel over one physical channel of each type. Then, the *default channel* will represent a virtual heterogeneous network (in particular, the *Madeleine* forwarding mechanisms can be fully employed) and still, the topology can be seen at the MPI layers since remaining physical channels are present.

4 Performance Evaluation

This section will describe point-to-point experiments conducted in order to assert the level of performance achievable by our MPI implementation. Since we focus on the implementation oh MPICH over the *Madeleine* communication library, no comparisons with other existing solutions are presented (such comparisons can be found in [3]). The figure will be shown for two types of high-performance networks: SCI and Myrinet. The protocols used are respectively SISCI and BIP [14].

The hardware used for the experiments are Bi-Pentium II nodes 450 MHz, with 128 Mbytes memory and 32 bits-wide PCI bus. As for the network interface cards, Dolphin's D310 and Myricom's Lanai 9.xx were at our disposal. The test is a ping-pong and we measured both transfer time and throughput. As far as transfer time is concerned, round-trip and one-way measures have been carried out. Since MPICH/MADIII internally uses threads acting as net servers for the different channels/network, the one-way should give us a basic evaluation of the reactivity possible in an application.

We made comparisons for homogeneous as well as heterogeneous cases between *Madeleine* and MPICH/MADIII. Since we explained that each message larger than 16 bytes in MPICH/MADIII was built by a succession of *Madeleine* packing operations in which the first data to be packed is a fixed-size header (actually 56 bytes) we simulated this behavior within *Madeleine* test programs in order to evaluate the cost of this extra packing step. The other source of overhead comes from the synchronization mechanisms which are mandatory to protect data structures and allow a correct thread behavior. 1 Mbyte represents 1024*1024 bytes.

4.1 Homogeneous Network Performance

4.1.1 SCI

Fig. 2 shows the performance obtained above the SCI network with the SISCI protocol. As expected, the raw *Madeleine* test outperforms the others. One might note that the latency for both roundtrip and one-way modes is almost the same in that precise case. In the case of *Madeleine* simulating our internal communication protocol, performance are the same as raw *Madeleine* for small messages, and for messages larger than 16 bytes, a 15 μs gap due to the extra packing operation can clearly be seen. The transfer time then gets in the same range of performance as MPICH/MADIII. The difference between MPICH/MADIII and the protocol simulation is also roughly of 15 μs. To conclude, the minimal latency achieved with MPICH/MADIII is 20 μs. The bandwidth figures, shown by Fig. 3, can give

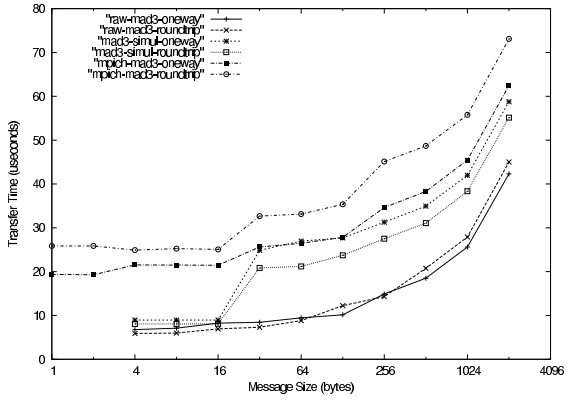

Figure 2. Transfer Time Comparison

us another idea of the overhead. Here again, we can see that roughly 50% of the overhead is due to the extra packing operation. A more interesting fact is that for message which size is larger than 16 kilobytes the gap between raw *Madeleine* and MPICH/MADIII gets narrower. This proves that the use of the *ch_mad* device as well as the synchronization mechanisms doesn't prevent us from exploiting almost the whole bandwidth delivered by *Madeleine*.

4.1.2 Myrinet

As far as transfer time with Myrinet network is concerned, the same kind of remarks can be made as in the SCI case. The extra packing operation cost is about 15 μs and represents roughly 50% of the total overhead, the remaining being synchronization and extra software layers. For messages larger than 1 kilobyte the cost of the extra packing operation become almost UN-noticeable compared to the other sources of overhead.

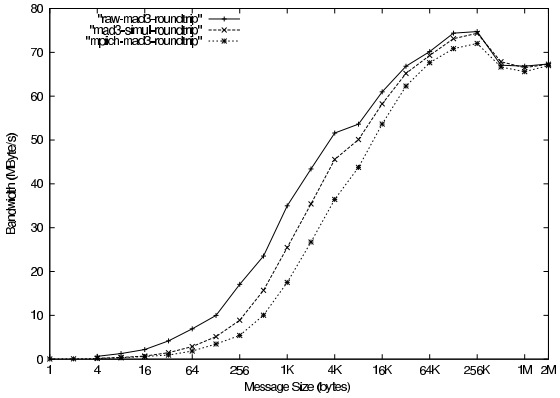

Figure 3. Bandwidth Comparison

The difference between SCI and Myrinet lies essentially in the latency for very small messages (< 16 bytes). Otherwise, the performance pattern follows the same scheme.

As for the throughput, we can notice (Fig. 5) that the

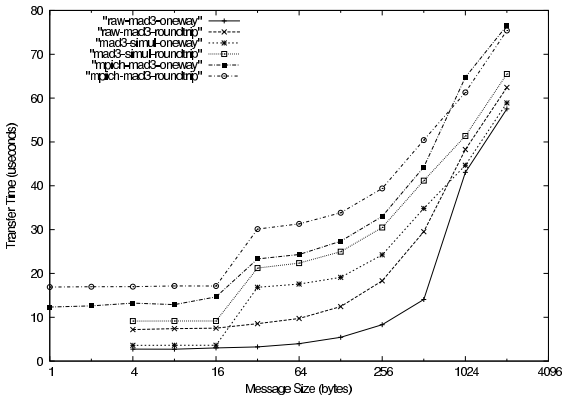

Figure 4. Transfer Time Comparison

overhead is considerably less noticeable than in the SCI case. For messages smaller than 1 kilobyte, the overhead is clearly due to the extra packing operation and confirms the phenomenon already experienced for transfer time. Otherwise, the synchronization mechanisms as well as the cost of MPICH software layers impact lightly the global performance. In the Myrinet case also, we manage to exploit *Madeleine* capabilities at almost their maximum.

4.2 Heterogeneous Network Performance

In the case of experiments for heterogeneous networks, it is interesting to measure both the performance of round-trip and one-way mode: this will emphasize on the asymmetrical behavior of the forwarding gateway.

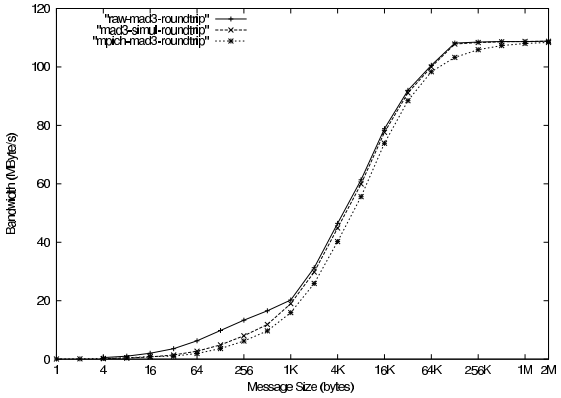

Figure 5. Bandwidth Comparison

The latency when sending a message forwarded by a gateway is more important than the sum of the latencies of each network used: *Madeleine*-generated overhead (roughly 30 μs) is added. We can observe that a message sent with Myrinet and forwarded to SCI goes quicker than in the other direction. The minimal transfer time achievable with MPICH/MADIII is about 70 μs. Since this is the half of the transfer time obtained with a FastEthernet/TCP network, this result clearly advocate for the use of an MPICH/MADIII-like solution. The same conclusion can be drawn when seeing the bandwidth figures. With a maximum throughput of more than 35 Mbytes/s (more than three times the maximum of FastEthernet/TCP), this solution is a good alternative to inter-operable based schemes.

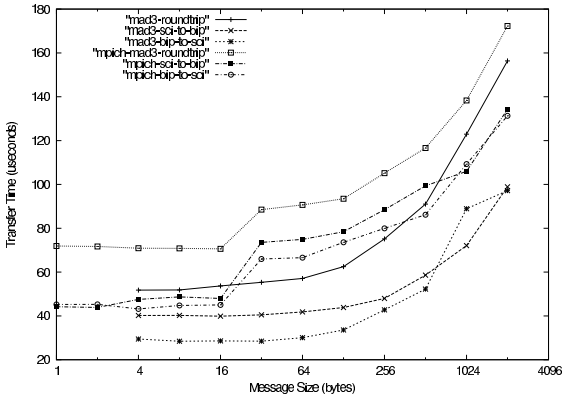

Figure 6. Transfer Time Comparison

5 Conclusion and Further Development

In this paper, we propose a solution to exploit efficiently an interconnection of clusters, without sacrificing the whole

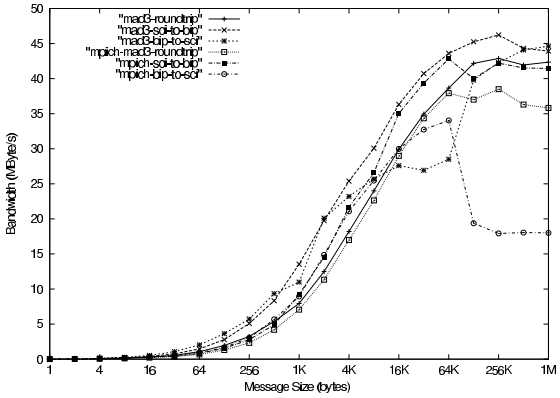

Figure 7. Bandwidth Comparison

system's easiness of use. It is based on a custom MPICH implementation, with dedicated extensions allowing multi-cluster management. The core of this implementation relies on a specific multi-threaded device built upon a dedicated communication library called *Madeleine*. We managed to convey the topological information accessible with *Madeleine* up to the outer layers of MPICH by creating matchings between *Madeleine* channels and regular MPI communicators.

The performance evaluation shows that the device manage to deliver almost the maximum of *Madeleine* capabilities, as far as bandwidth is concerned. We now intend to pursue the study of the impact of the forwarding mechanism at the MPICH/MADIII level, in both homogeneous and heterogeneous cases. The gateways can be potential bottlenecks for performance. Also, the experiments have been conducted in mono-processor mode: we will test the SMP mode of MPICH/MADIII, where a single MPI process actually use several processors at the same time. This functioning scheme might impact on the reactivity of applications. Since point-to-point experiments can only give us basic assertions about a system's performance level, we intend to conduct benchmarks such as Linpack or the NAS parallel benchmark.

References

[1] MPICH-G2: a Grid-enabled Implementation of MPI. `http://www3.niu.edu/mpi/`.

[2] Olivier Aumage, Luc Bougé, Alexandre Denis, Lionel Eyraud, Jean-François Méhaut, Guillaume Mercier, Raymond Namyst, and Loïc Prylli. High performance computing on heterogeneous clusters with the Madeleine II communication library. *Cluster Computing*, 5:43–54, 2002. Special Issue on the Cluster 2000 Conference.

[3] Olivier Aumage, Guillaume Mercier, and Raymond Namyst. MPICH/Madeleine: a True Multi-Protocol MPI for High-Performance Networks. In *Proc. 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, page 51, San Francisco, April 2001. IEEE. Extended proceedings in electronic form only.

[4] Luc Bougé, Vincent Danjean, and Raymond Namyst. Improving Reactivity to I/O Events in Multithreaded Environments Using a Uniform, Scheduler-Centric API. In *Euro-Par*, Paderborn (Deutchland), AUG 2002.

[5] Luc Bougé, Vincent Danjean, and Raymond Namyst. Improving Reactivity to I/O Events in Multithreaded Environments Using a Uniform, Scheduler-Centric API. In *Euro-Par*, Paderborn (Deutchland), AUG 2002.

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

[7] Nathan Doss, William Gropp, Ewing Lusk, and Anthony Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Technical report, Argonne National Laboratory, 1996.

[8] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, and G. Thruvathukal ans S. Tuecke. Wide-Area Implementation of the Message Passing Interface. In *Parallel Computing*, volume 24, pages 1735–1749, 1998.

[9] 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.

[10] Sébastien Lacour. MPICH-G2 collective operations: performance evaluation, optimizations. Internship report, Magistère d'informatique et modélisation (MIM), ENS Lyon, MCS Division, Argonne Natl. Labs, USA, September 2001.

[11] Ewing Lusk and William Gropp. MPICH Working Note : the implementation of the second generation ADI. Technical report, Argonne National Laboratory.

[12] Ewing Lusk and William Gropp. MPICH Working Note : The Second-Generation ADI for the MPICH Implementation of MPI. Technical report, Argonne National Laboratory, 1996.

[13] Raymond Namyst and Jean-François Méhaut. PM2: Parallel multithreaded machine. a computing environment for distributed architectures. In *Parallel Computing (ParCo '95)*, pages 279–285. Elsevier Science Publishers, September 1995.

[14] Loïc Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on myrinet. In *Parallel and Distributed Processing, IPPS/SPDP'98*, volume 1388 of *Lecture Notes in Computer Science*, pages 472–485. Springer-Verlag, April 1998.

[15] Toshiyuki Takahashi, Shinji Sumimoto, Atsushi Hori, Hiroshi Harada, and Yutaka Ishikawa. PM2: High Performance Communication Middleware for Heterogeneous Network Environments. In *SuperComputing'2000*, pages 52–53, 2000.