NEWMADELEINE: a Fast Communication Scheduling Engine for High Performance Networks [*]

Olivier AUMAGE, Élisabeth BRUNET, Nathalie FURMENTO, Raymond NAMYST

INRIA, LABRI, Université Bordeaux 1,
351 cours de la Libération
F-33405 TALENCE, FRANCE
{aumage, brunet, furmento, namyst}@labri.fr

Abstract

Communication libraries have dramatically made progress over the fifteen years, pushed by the success of cluster architectures as the preferred platform for high performance distributed computing. However, many potential optimizations are left unexplored in the process of mapping application communication requests onto low level network commands. The fundamental cause of this situation is that the design of communication subsystems is mostly focused on reducing the latency by shortening the critical path. In this paper, we present a new communication scheduling engine which dynamically optimizes application requests in accordance with the NICs capabilities and activity. The optimizing code is generic and portable. The database of optimizing strategies may be dynamically extended.

1 Introduction

The success of cluster architecture as the most widespread platform for high performance computing mainly comes from their aggressive performance/cost ratio. However, it is also the result of the substantial progress made in the field of fast cluster interconnects. Hardware improvements towards lower latency and higher bandwidth have been followed by huge progress at the software level. Using a large panel of mechanisms such as user-mode communications, zero-copy transactions and communication operation offload, the critical path in sending and receiving a packet has been drastically reduced. The cost is now only a few hundreds of microprocessor cycles per transaction with libraries such as Elan/Quadrics [9] or MX/Myrinet [7]. Even more impressive is the fact that some recent implementations of the MPI standard such as MPICH2 [3] or OpenMPI [5], which have been carefully designed to directly map *basic* point-to-point requests onto the underlying low-level interfaces, almost exhibit themselves the same level of performance.

However, for that matter, only the very basic point-to-point messaging requests are considered first-class citizens. More complex requests such as non-contiguous messages are left mostly unattended, and even more so are the irregular and multi-flow communication schemes. The purpose of the NEWMADELEINE communication engine introduced in this paper is to address this situation thoroughly. With a carefully designed architecture, explained in Section 3, the NEWMADELEINE optimization layer delivers much better performance on complex communication schemes with a negligible overhead on basic single packet point-to-point requests as it will be shown in Section 5. Through MAD-MPI, our simple, straightforward proof-of-concept implementation of a subset of the MPI API, we show that MPI applications can also benefit from the NEWMADELEINE communication engine.

2 New optimization potentials for communications

The art of mapping simple contiguous point-to-point requests from the MPI API onto fast, low latency networks arguably reached its climax with MPI implementations such as MPICH2 [3] or OpenMPI [5], in terms of pure efficiency. However, needs of real-life applications in communication requests not always meet such a narrow focus. The *message layout* may be more complex. Messages may be made of multiple fragments disseminated irregularly in memory.

*This work was funded through the ANR-05-CIGC-11 grant.

The preferred *optimization strategy* may differ from favoring the latency, and instead favoring the bandwidth may be a better bet for applications using a remote storage system. A preference for communication overlap may be more suitable for computing intensive applications. Applications may even have need for different optimization strategies at different stages or for different requests.

Even the MPI API itself is not always the most suited API. We showed in the past ([1]) that applications and programming environments that make use of high-level communication protocols (DSM, RPC, etc.) may draw a substantial benefit from using a communication interface more powerful that the regular MPI. Whilst this interface behaves optimally for the classical *ping-pong* tests and the simple regular communication patterns, the overall lack of expressiveness of MPI does not capture subtle informations such as the dependencies between the multiple fragments (service request, arguments, targeted object) of a remote method invocation.

As a result, many communication schemes and needs are left mostly unexplored. These schemes —and combination of theses— represent a rich field of optimization potentials. Our new NEWMADELEINE communication engine has been designed with the purpose of harvesting these potentials without incurring a prohibitive penalty for basic communication requests. Our engine dynamically optimizes the communication flow or multiple flows (increasingly found in nowadays composite applications). It may decide to accumulate packets in order to make use of some gather/scatter capabilities or to aggregate several short requests into a single larger one if constraints allow it, or to reorder packets, or even to favor an earlier delivery of high priority fragments (such as a RPC service id, needed for preparing the data areas to receive the service arguments). For such classes of applications characterized by complex, irregular communication schemes, it is indeed not a matter of simply translating communication operations into calls to the network driver, but rather to dynamically *interpret*, *reorganize* and *optimize* the communication operation flow.

3 The NEWMADELEINE communication engine

Our new communication support framework has been designed and implemented in a communication library called NEWMADELEINE [1]. We present in the rest of the section the main features of NEWMADELEINE.

[1]This library is a new evolution of the communication library known as MADELEINE.

3.1 Architecture

The NEWMADELEINE architecture is organized in three layers as shown on Figure 1, an application data collect layer, an optimizing and scheduling layer and a transfer layer that controls the network cards. Only the transfer layer is NIC-specific.

The transfer layer mimics a process scheduler, which when called by a processor, will select the new ready process to be run. Indeed, the transfer layer controls the activities of the NICs, and requests from the upper layer a new optimized packet to be sent, as soon as a card becomes idle. Specific drivers are available for each NIC. This layer is minimal and does not need any extra knowledge from the other layers.

The optimizing and scheduling layer is queried by the transfer layer when a NIC gets idle and needs to be refilled. If any packet has been previously prepared (see Section 3.3), it is then responsible for analyzing the backlog of accumulated packets in order to build a request both compatible with the application constraints and efficient performance-wise. This request is then immediately submitted to the idle NIC.

The collect layer is in charge of registering the pieces of data submitted by the various communication flows of the application as well as the meta-data necessary in their identification by the receiving side (tag number, sender id, sequence number) (see Section 3.2). Once encapsulated, and in order to load balance the packets among the available NICs (possibly from heterogeneous technologies), the collected pieces of data are inserted onto a dedicated list for a specific network technology selected by the application or (by default) on the common list for automatized load-balancing among all the NICs (possibly from heterogeneous technologies).

3.2 Optimization window

To be efficient, dynamic optimizations for multi-packets and multi-flow communication schemes require the use of an optimization window that accumulates packets. Traditionally, communication libraries, being synchronous, tightly link the communication requests to the application workflow, and therefore transmit incoming packets immediately to the lower network layer without any accumulation. A communication library that would choose to accumulate packets with the aim of possible optimizations would lead to increased latency and possibly cause deadlocks. To avoid such drawbacks, it is necessary to untie the processing of the communication requests from the application workflow, and instead to link this processing to the activity of the NICs (Network Interface Card).

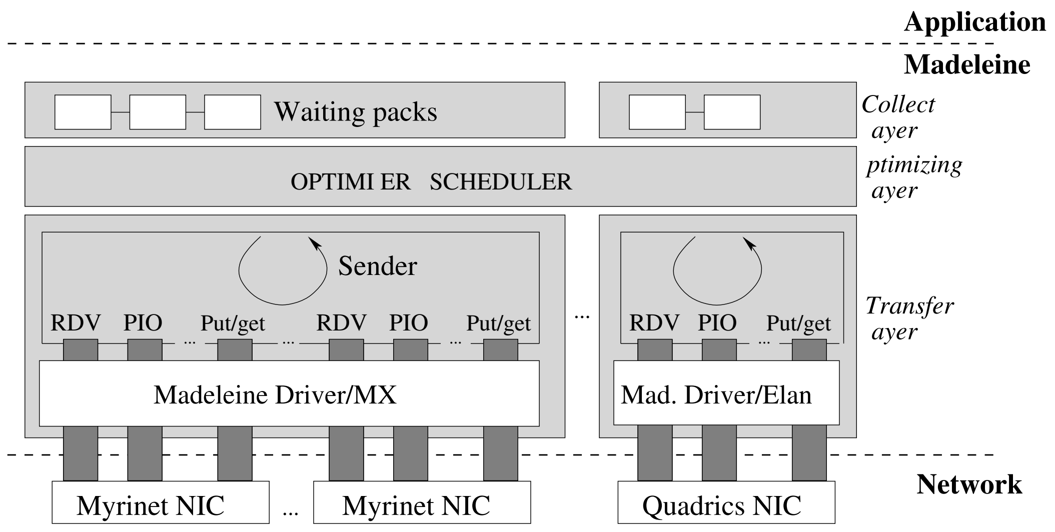

Figure 1. NEWMADELEINE **architecture.**

While the NICs are busy, NEWMADELEINE keeps accumulating packets in its optimization window. As soon as a NIC becomes idle, the optimization window is analyzed so as to create a new ready-to-send packet to be transfered through the card. This approach does thus not have the cons of the synchronous solutions. It also brings two advantages: NICs are exploited at their maximum (they are not overloaded when there is a high demand of transfers and under exploited when there is not) and the communication optimizations are made *just-in-time* so they closely fit the ongoing communication scheme at any given time.

3.3 Strategies, tactics and optimization selection

The optimization window being built, one has now to deal with the actual optimization work. Let us consider the state of the physical multiplexing units at a given time. If at least one of the multiplexing units is idle, one has to assign it some work. The assignment is done by calling an *optimization function* to elect the next request to be submitted to each idle unit. In doing so, it may select a packet to be sent from the optimization window, or for instance, synthesize a request out of several packets from that window. A wide panel of arguments may be used as an input to the optimizing function: the number of packets in the window, the specific characteristics of each packet (destination, flow tag, length, sequence number, dependency attributes), the nominal and functional characteristics of the underlying network, possibly some hints given by the application itself with respect with the packet scheduling policy, and the number and state of multiplexing units at this point in time.

Given such a large amount of potential parameters, several optimization tactics may be available, but the optimal combination of these tactics is a difficult problem. We thus propose a (dynamically in the future) selectable optimization function instead of a fixed optimizing heuristic. The optimization function is to be selected among an extensible and programmable set of *strategies*. Each tactic applies some elementary optimizing operations selected from the panel of usual operations toward some particular optimizing goal.

While any multiplexing unit is available, the communication requests are just accumulated. Another possibility would be to prepare a single *ready-to-send* packet to anticipate for any upcoming completion of the current activity of one of the multiplexing unit and immediately re-feed it once it becomes idle. A third possibility would be to run the optimization function unconditionally once the packet backlog has reached a predefined threshold length.

3.4 Application Programming Interfaces

The communication engine NEWMADELEINE provides several interfaces. The first interface is similar to the interface of the former MADELEINE library, it allows to incrementally build messages. With this interface, a NEWMADELEINE message is made of several pieces of data, located anywhere in user-space. The message is initiated and finalized with a flush point which acts as a *memory barrier*. The beginning flush of a given message is actually the ending flush of the previous message. The flush ensures that any piece of message appended since the previous flush has been actually sent and that the associated memory area may safely be reused by the application.

Thereafter, in order to exhibit the performance of NEW-

MADELEINE with MPI applications, we have implemented a subset of the MPI standard on top of NEWMADELEINE. This implementation called MAD-MPI is based on the point-to-point nonblocking posting (`isend`, `irecv`) and completion (`wait`, `test`) operations of MPI, these four operations being directly mapped to the equivalent operations of NEWMADELEINE.

MAD-MPI also implements some optimizations mechanisms for derived datatypes [4]. MPI derived datatypes deal with noncontiguous memory locations. The advanced optimizations of NEWMADELEINE allowing to reorder packets should lead to a significant gain when sending and receiving data based on derived datatypes.

4 Highlights of the implementation

A NEWMADELEINE prototype has been implemented over GM/MYRINET, MX/MYRINET, ELAN/QUADRICS, SISCI/SCI and TCP/ETHERNET. The implementation of each corresponding *transfer layer* consists in a minimal network API (initialisation, closing, sending, receiving and polling methods) for which each function is, at the best, a direct call to the adequate function of the network driver. In addition, some information are collected such as the threshold for the *rendezvous* protocol or the availability of the *gather/scatter* or as well the remote direct access (RDMA) functionality.

Scheduling strategies are as well based on a minimal interface and are independent from the network technology. Developing a new strategy only requires to write a few methods such as an initialisation method, and a request method which returns the next communication request to be sent or received. Information about the underlying network can be obtained in a generic manner through a specific API. Thus, any strategy can be directly combined with any network protocol supported by NEWMADELEINE. Currently, the NEWMADELEINE scheduler offers three strategies: a straighforward strategy which simply sends data segments as they are submitted by the application, an *aggregation* strategy and a *multi-rails* strategy. The aggregation strategy is implemented in the following way. While the NIC is busy, the optimizing code considers new message segments for aggregation. If a given segment is short enough not to require any rendez-vous, then the segment is aggregated with previous unsent segments. If, on the contrary, the segment is long and requires a rendez-vous for being sent, then the segment itself is not aggregated but kept separately. However in that later case, the rendez-vous request segment is aggregated with previous segments. In both cases, if the aggregated length would exceed the rendez-vous, threshold then a new sequence of aggregated segments is started. Finally, as soon as the NIC becomes available again, the sequence (or the oldest sequence if many sequences have been accumulated) of aggregated segments is fed to the NIC. The multi-rail strategy is detailed in [2]. For short segments (that is the segments that do not require a rendez-vous), the multi-rail strategy is basically the same as the aggregation strategy. The aggregated request is fed to network which is the most efficient for sending a message of the aggregated length. For long segments, the multi-rail strategy differs from the aggregation strategy. The long segment is split into as many fragments as the number of idle NIC(s). The pieces of the fragments are not necessarily of same length if the NICs are heterogeneous because the length of each piece of segment is calculated so that sending each piece of segment takes approximatively the same amount of time on each related NIC.

5 Evaluation

In this section, we present the results obtained by comparing our lightweight implementation of MPI (MAD-MPI) against dedicated implementations of MPI. The first test evaluates the software overhead introduced by MAD-MPI and the following ones highlight the gain obtained by reordering communication requests.

All our experiments have been carried out on the same platform, a set of two dual-core 1.8 GHz OPTERON boxes with 1MB of L2 cache and 1GB of main memory. The OS kernel is Linux version 2.6.17. Nodes are interconnected through MYRI-10G NICs with the MX1.2.0 driver and through QUADRICS QM500 NICs with the ELAN driver.

5.1 Overhead of NEWMADELEINE

To evaluate the overhead of the NEWMADELEINE scheduling engine under situations where no optimization is possible we force the straightforward strategy at compile time and use a MPI ping-pong program exchanging single-segment messages (i.e. contiguous arrays of bytes). Figures 2 and 3 shows the transfer times obtained with MPICH-MX, OPENMPI-MX 1.1 and with MAD-MPI over MYRI-10G and Figures 4 and 5 ones obtained with MPICH-QUADRICS and MAD-MPI over QUADRICS.

On both networks, MAD-MPI introduces a constant overhead of less than 0,5 μs and reaches 1155 Mbytes/s in bandwidth over MYRI-10G and 835 Mbytes/s over QUADRICS. In the both case, this small overhead actually results from different factors. On one hand, an extra header is systematically added to the data by NEWMADELEINE for allowing the reordering and the multiplexing of the packets. Thus, the exchanged packets are slightly larger with NEWMADELEINE than with MPICH-MX. On the other hand, the NEWMADELEINE scheduler introduces some extra operations on the critical path to inspect the "ready list" of packets and check if some optimization action would be

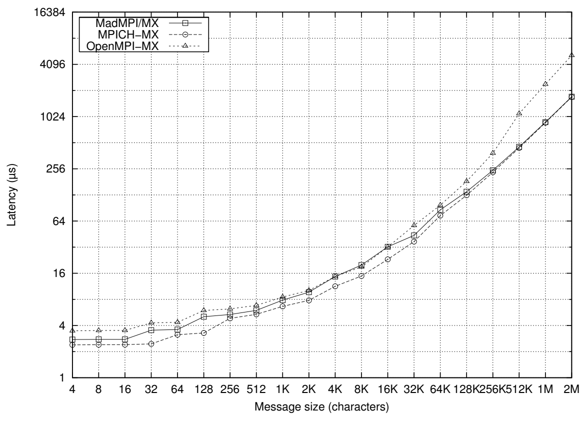

Figure 2. Raw point-to-point ping-pong - Latency over Myri-10G.

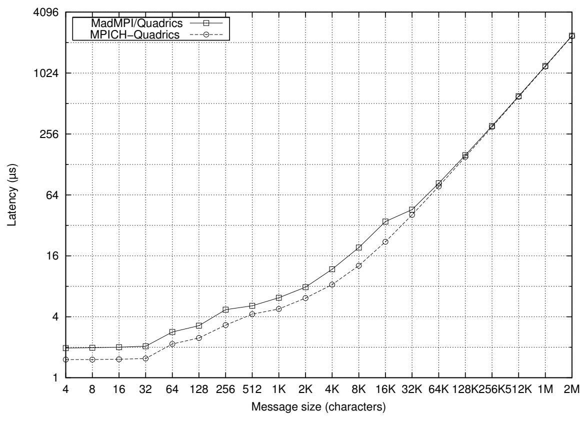

Figure 4. Raw point-to-point ping-pong - Latency over Elan/Quadrics.

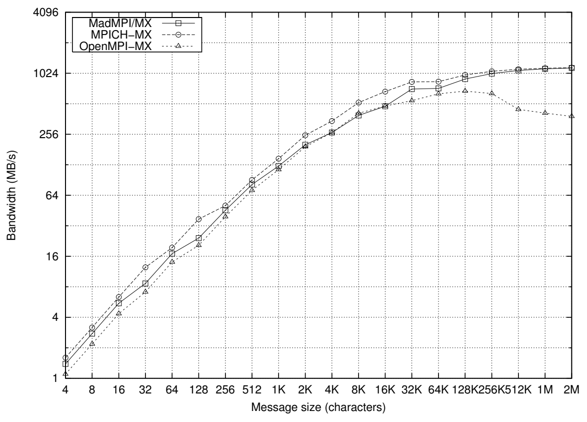

Figure 3. Raw point-to-point ping-pong - Bandwidth over Myri-10G.

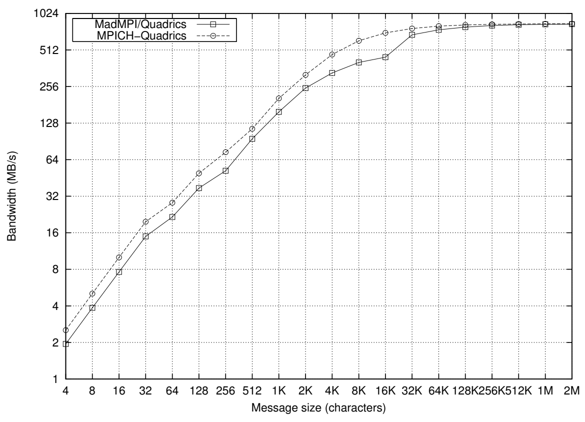

Figure 5. Raw point-to-point ping-pong - Bandwidth over Elan/Quadrics.

appropriate. The performance target of NEWMADELEINE not being regular communication schemes, we consider that this latency overhead of MAD-MPI remains reasonable.

5.2 Aggregation of small messages

To evaluate the benefits of using a communication scheduler able to perform optimizations over the whole communication flow between a pair of processes, we compare the performance of a multi-segments ping-pong program with several implementations of MPI. 8 (and then 16) segments are sent in each ping or pong message. Each segment uses an independent `MPI_Isend` (resp. `MPI_Irecv`) operation over a separate MPI communicator (to demonstrate that the scope of MAD-MPI optimizations is really global).

The MAD-MPI implementation was configured to use the *aggregation* optimization strategy. Results obtained over MYRI-10G and QUADRICS are reported on Figures 6 to 9. The latency reported is the average latency (over 1000 transfers) for the entire series of segments sent in a ping or pong message.

Neither the MPICH nor the OPENMPI try to aggregate individual messages submitted in a short time interval. Nevertheless, the MPICH-MX and MPICH-QUADRICS implementations are able to pipeline the transfer of a series of messages in a very efficient manner. Despite this fact, the MAD-MPI implementation outperforms MPICH and OPENMPI thanks to its agressive optimizer which is able to coalesce packets even if they belong to different logical communication flows (i.e. MPI communicators). We can observe that MAD-MPI is up to 70 % faster than other implementations of MPI over MYRI-10G, and up to 50 % faster that MPICH over QUADRICS.

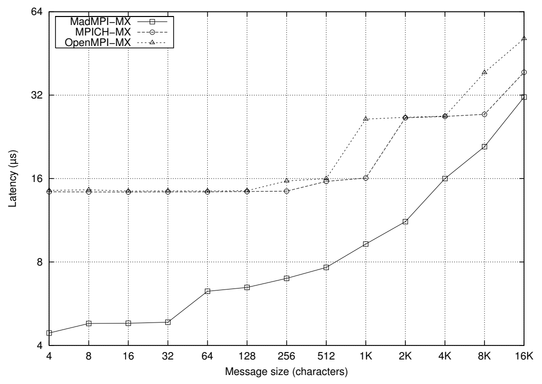

Figure 6. 8-seg. ping-pong - Latency over Myri-10G.

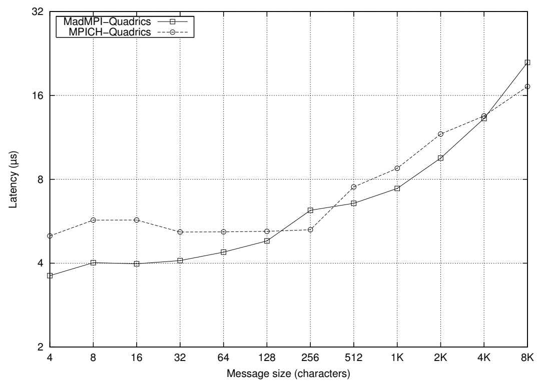

Figure 8. 8-seg. ping-pong - Latency over Elan/Quadrics.

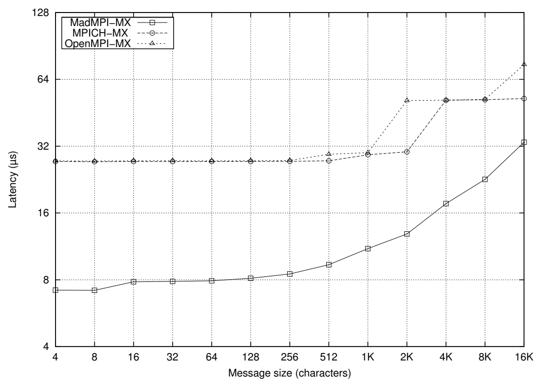

Figure 7. 16-seg. ping-pong - Latency over Myri-10G.

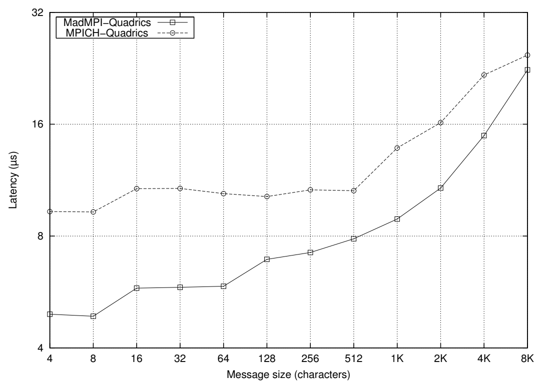

Figure 9. 16-seg. ping-pong - Latency over Elan/Quadrics.

5.3 Optimization of derived datatypes

We now present the performance of our optimization mechanisms when using MPI derived datatypes. We use a ping-pong program which exchanges arrays of a given indexed datatype. The datatype describes a sequence of two data blocks, one small block (64 bytes) followed by a large data block (256 KBytes).

In order to process a derived datatype communication request, MPICH copies all the data fragments into a new contiguous buffer and sends the obtained buffer in an unique transaction, using the *rendezvous* protocol if necessary. Data are received in a temporary memory area before being dispatched to their final destination [6]. This behaviour is certainly optimized when dealing with a small overall data size as the `memcpy` operations for each the data blocks will cost less than the multiple communication operations. However, the cost of a memory copy operation being proportional to the size of the data, this behaviour is no longer optimized when dealing with bigger blocks.

OPENMPI (version 1.2) currently follows the same policy for small messages (less than 8 KBytes). Large datatypes are transmitted using gather/scatter communication primitives, but no particular optimization is done to possibly rearrange segments and aggregate the smaller ones.

On the opposite, MAD-MPI uses an algorithm which generates an individual communication request for each block, allowing the underlying communication layer to perform any appropriate optimization. We used a scheduling strategy which aggregates all the small blocks (using messages reordering) with the *rendezvous* requests of the large blocks, hence the large blocks are directly received at their final destination, and the whole transfer is made with a zero-copy technique.

This strategy decreases significantly the time transfer of indexed datatypes build upon small and large fragments. On our example (see Figures 10 and 11), we can observe a gain of about 70 % in comparison with MPICH and about 50 % with OPENMPI over MYRI-10G and until about 70 % versus MPICH over QUADRICS.

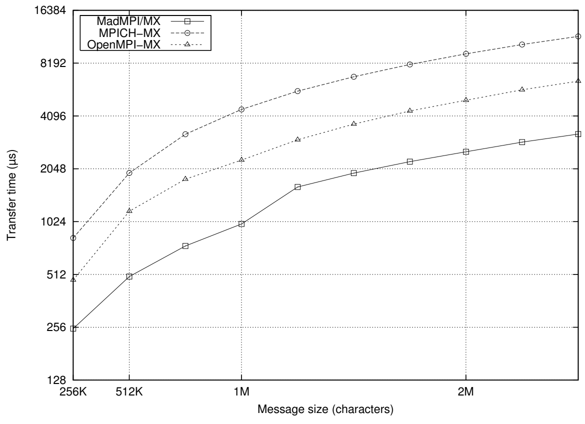

Figure 10. Indexed datatype over Myri-10G.

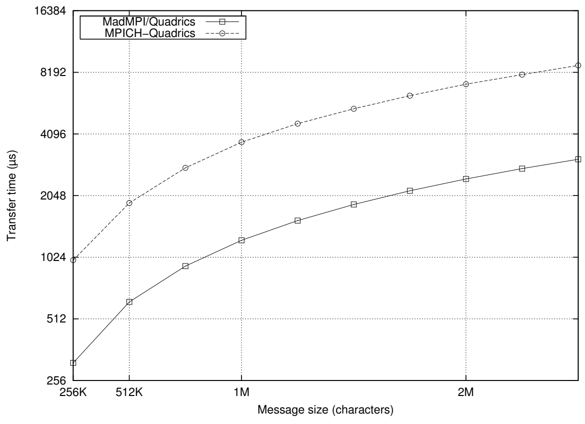

Figure 11. Indexed datatype over Elan/Quadrics.

6 Related works

This section briefly presents the recent research efforts in designing communication supports for high performance networks.

MPICH2-NEMESIS [3] is to our knowledge the current best implementation of the MPI standard both in term of bandwidth and latency. Nevertheless, its implementation currently focusses on the performance of individual transfers. Thus, no message reordering or multiplexing is used.

PMV2 [11] is used as the multi-network communication subsystem for the YAMPII [8] MPI implementation by the same team. As such, it also focuses on achieving low latency, as does MPICH2-NEMESIS.

VMI 2 [10] is a low-level communication library dedicated to address fault tolerance issues. It provides multi-rails capabilities on top of homogeneous and heterogeneous networks. It also implements some NIC scheduling but does not provide multiplexing of communication flows. It is therefore mostly oriented towards improving bandwidth.

MADELEINE 3 [1] is the preceding major version of NEWMADELEINE. It already was performing some very basic request optimizations such as aggregating packets without inter-dependencies. However, MADELEINE 3 was only sending header-less packets by design, making it impossible to implement non-deterministic sender actions such as opportunistic multi-flow aggregation or packet reordering.

7 Conclusion and Future Work

In this paper, we present a new low-level communication engine for high speed networks. NEWMADELEINE features a programmable packet scheduler that can use a wide range of "*just-in-time*" optimization strategies. Data segments can be aggregated into the same physical packet even if they belong to different logical channels (e.g. different MPI communicators). If needed, they can be reordered (to maximize the number of aggregation operations), or even sent out-of-order. Large data segments can also be split on the sending side (and later reassembled on the receiving side) into several chunks that may be sent through different networks.

To demonstrate the relevance of using such a communication engine for building communication libraries such as MPI implementations, we developed a simple optimization strategy that aggressively tries to aggregate small packets (data and control) whenever possible. The experiments conducted over MYRI-10G and QUADRICS networks show that the gain obtained when transferring complex (i.e. non contiguous) messages within MPI applications can be substantial. We also showed that the overhead of the NEWMADELEINE optimizer is small when running applications with no opportunity for communication optimizations.

We are currently designing more powerful optimization strategies to efficiently exploit multiple, heterogeneous physical networks within the same application. The NEWMADELEINE architecture is particularly well suited to the implementation of greedy load-balancing strategies over multiples network interface cards. In the short term, we also plan to port a full featured MPI implementation such as MPICH2 [3] or OpenMPI [5] on top of NEWMADELEINE, to evaluate the impact of aggressive communication optimizations in real applications.

References

[1] O. Aumage, L. Boug, A. Denis, L. Eyraud, J.-F. Mhaut, G. Mercier, R. Namyst, and L. Prylli. A Portable and Efficient Communication Library for High-Performance Cluster Computing. *Cluster Computing*, 5(1):43–54, 2002.

[2] O. Aumage, E. Brunet, G. Mercier, and R. Namyst. High-performance multi-rail support with the newmadeleine communication library. In *HCW 2007: the Sixteenth International Heterogeneity in Computing Workshop*, Long Beach, California, USA, March 2007.

[3] D. Buntinas, G. Mercier, and W. Gropp. Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem. In *The Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06)*, pages 521–530, Washington, DC, USA, 2006. IEEE Computer Society.

[4] N. Furmento and G. Mercier. Optimisation Mechanisms for MPICH-Madeleine. Technical Report 0306, INRIA, July 2005. Also available as LaBRI Report 1362-05.

[5] R. L. Graham, T. S. Woodall, and J. M. Squyres. Open MPI: A Flexible High Performance MPI. In *The 6th Annual International Conference on Parallel Processing and Applied Mathematics*, 2005.

[6] W. Gropp, E. Lusk, and D. Swider. Toward faster packing and unpacking of mpi datatypes. `http://www.mcs.anl.gov/mpi/mpich1/papers/index.html`.

[7] M. Inc. Myrinet EXpress (MX): A High Performance, Low-level, Message-Passing Interface for Myrinet, 2003. `http://www.myri.com/scs/`.

[8] Y. Ishikawa. YAMPII Official Home Page. `http://www.il.is.s.u-tokyo.ac.jp/yampii`.

[9] Q. Ltd. Elan Programming Manual, 2003. `http://www.quadrics.com/`.

[10] S. Pakin and A. Pant. VMI 2.0: A Dynamically Reconfigurable Messaging Layer for Availability, Usability, and Management. In *The 8th International Symposium on High Performance Computer Architecture (HPCA-8)*, 2002.

[11] T. Takahashi, S. Sumimoto, A. Hori, H. Harada, and Y. Ishikawa. PM2: High performance communication middleware for heterogeneous network environments. In *SC'00*, pages 52–53, 2000.