A multicore-enabled multirail communication engine

Elisabeth Brunet [#1], Francois Trahay [#2], Alexandre Denis [*3]

[#] *University Bordeaux 1/LaBRI,*
351 cours de la Liberation F-33405 Talence, France
[1] elisabeth.brunet@labri.fr
[2] francois.trahay@labri.fr

[*] *INRIA Bordeaux – Sud-Ouest/LaBRI,*
351 cours de la Liberation F-33405 Talence, France
[3] alexandre.denis@labri.fr

***Abstract*—The current trend in clusters architecture leads toward a massive use of multicore chips. This hardware evolution raises bottleneck issues at the network interface level. The use of multiple parallel networks allows to overcome this problem as it provides an higher aggregate bandwidth. But this bandwidth remains theoretical as only a few communication libraries are able to exploit multiple networks. In this paper, we present an optimization strategy for the NEWMADELEINE communication library. This strategy is able to efficiently exploit parallel interconnect links. By sampling each network's capabilities, it is possible to estimate a transfer duration *a priori*. Splitting messages and sending chunks of messages over parallel links can thus be performed efficiently to reach the theoretical aggregate bandwidth. NEWMADELEINE is multithreaded and exploits multicore chips to send small packets, that involve CPU-consuming copies, in parallel.**

I. INTRODUCTION

During the past few years, the typical architecture of clusters has greatly evolved due to the impressive emergence of multicore chips. The increase of the number of cores per machine – 16 cores per node is now commonplace – is expected to continue and will soon lead to cluster nodes made up of hundreds of cores. To avoid the potential bottleneck caused by many cores accessing a single network interface card, some clusters feature multiple physical networks to provide each computing node with several network interface cards. These "multirail" clusters also allow to increase the maximum available bandwidth between peers. The T2K OpenSupercomputer installed in Kyoto [1], which is composed of 16-core nodes interconnected with a 4-link Infiniband network, is a good illustration of such multirail architectures.

Even within multirail clusters, the number of NICs did not grow up as quickly as the number of cores, so each network interface card (NIC) still has to be shared by several cores. While most programming environments simply assign each communication flow to a single network link, we believe it is more relevant to adopt a more dynamic approach by multiplexing all communication flows on top of the multirail network. The idea is to maximize network utilization by both dynamically balancing packets over the underlying links and transferring single packets over multiple links when possible. Depending on the state and capabilities of the underlying networks, multiple packets with the same destination may be aggregated and handled by a single core, or they may be sent in parallel by different cores over separate NICs.

The multirail-enabled communication library that we present in this paper is able to exploit multicore architectures to make communication flows progress in parallel. By sampling networks and by analyzing every NIC's state, it is possible to efficiently send data across several parallel links to reach the minimum transfer time.

The remaining of this paper is organized as follows. In Section 2, we discuss the issues raised when transferring data across multiple networks and we present our sampling-based communication system. In Section 3, some highlights of the implementation are described. Section 4 gives a performance evaluation of our multirail implementation. Finally Section 5 draws a conclusion and plans for future works.

II. EFFICIENT MULTIRAIL COMMUNICATION

With the increase of the number of cores per node in clusters, the inter-node communication subsystem becomes a bottleneck and communication intensive applications may not scale on such architectures because of limited bandwidth. Multiple parallel networks provide higher bandwidth and allow to overcome this scalability issue [2].

Efficiently exploiting parallel rails obviously profits to applications that communicate through small messages: data packets can be spread across the available networks, increasing the message rate. Applications that massively transfer large messages may also benefit from multiple links since it is possible to split a message to send it across several links in parallel. But in order to fully exploit multirail architectures and to reach the theoretical aggregate bandwidth, specific algorithms have to be designed at the communication library level.

A. Using multiple NICs

A basic support for multirail networks can easily be achieved by monitoring the state of each NIC: when a message has to be sent, any available network is used to transmit the

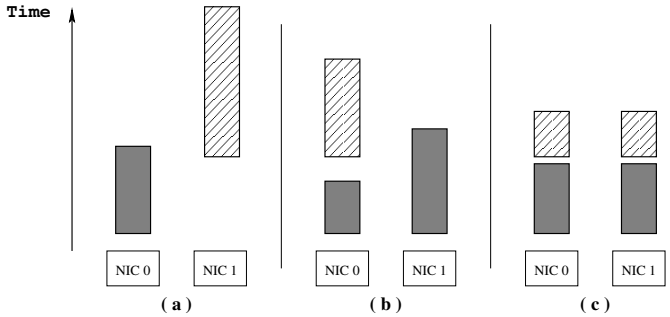

Fig. 1. (a) each message is sent on one network. (b) messages are split into chunks of equal size over the available networks. (c) messages are split into chunks of equal transfer time.

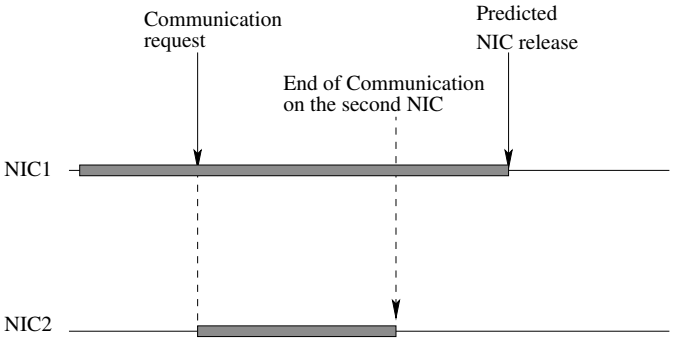

Fig. 2. Using predictions to select NICs.

data. This method does not use all the capabilities of the multirail network: it requires at least as many simultaneous communication flows as parallel networks to reach the maximum available bandwidth. Even if the global bandwidth is arisen, each communication flow transfer time is the same as if there were a single NIC. Moreover networks may be under-utilized if the application does not provide enough communication flows as depicted in Figure 1 (a).

In order to fully use available networks and to minimize transfer times, it is necessary to split messages and to send the resulting chunks in parallel. The main issue here is to determine how to split a message and to select a set of networks.

Equally splitting a message may be a good solution if all the networks share the same capabilities (*i.e.* same bandwidth, same latency, etc.), but as soon as networks are heterogeneous, this may lead to degraded network bandwidth utilization: some chunks are sent slowly whereas others are sent rapidly as shown on Figure 1 (b).

In order to reach the maximum network bandwidth, messages have to be split in such a way that the time required to send each chunk of a message is equal. This way, each chunk transfer will end at the same time, minimizing the transfer time of the whole message as the Figure 1 (c) illustrates. The split ratio of the chunks depends on network's capabilities. If two networks are involved in the transfer of a message, the fastest one will have to send more data than the other one. For example, OpenMPI [3] computes a ratio by comparing the maximum available bandwidth of each network. This method permits to achieve good performance for large messages, but suffers from a lack of precision as different network technologies do not behave the same way: a split ratio for a 8 MB message may not fit a 256 KB message.

Another issue raised when transferring data in parallel is that a NIC may become idle a few moments after the split ratio is chosen. This NIC will not be used because this would require to stop the pending transfer, compute another split ratio and then re-send the data. As a result, the misknowledge of networks' workload may lead to a potential underutilization of the links.

By knowing precisely how networks behave, a finner split ratio could be computed so that, independently from the size of the messages, all chunks end at the same time. The workload of networks can also be estimated with a more precise knowledge of networks' capabilities. The computation of the split ratio can thus take into account NICs that are currently busy but that will be idle soon.

B. Adapting the split ratio with sampling

In order to efficiently exploit all available networks, it is important to know their actual properties such as the underlying communication paradigm (*e.g.* message passing or RDMA), the availability of gather/scatter operations, etc. Beyond, a more valuable knowledge is the ability to predict how long a communication will take, and therefore to foresee when a NIC should become idle.

Aiming transfer time minimization, and given such predictions, it is possible to determine the best message scheduling. For instance, it could be interesting to either dispatch messages over all available NICs or simply a subset of those. It could also be worth delaying a transfer while some NICs that especially fit the considered transfer are busy.

Considering a cluster interconnected with two networks, the first step is to draw up which NICs should participate to the communication. As illustrated in Figure 2, NIC1 is typically discarded provided that NIC2 is expected to become free before NIC1. If several NICs are selected, the split ratio is determined by dichotomy. The algorithm begins by splitting the packets in two chunks of equal size. It then compares the predicted transfer time required by each network. For each interface, the time remaining before it becomes idle is added to its predicted transfer time. This dichotomy process is repeated until a split ratio where both transfer durations are equivalent is found.

C. A high-performance parallel communication engine

Figure 3 compares the transfer times obtained with a basic greedy balancing of the messages – when a NIC becomes idle, it looks after the next communication – with the ones where all the messages are sent sequentially over a single network. It clearly shows that it is not interesting to perform the transfers of eager packets – the packets which do not require a preliminary *rendezvous* handshake – through multiple

networks. Basically, it is more efficient to aggregate the messages and to send them over the fastest available network instead of using the entire set of network resources [4].

This low performance is mainly explained by the excessive use of the CPU due to the numerous PIO transfers from the host to the NIC memory – and in the other way if some receptions occur simultaneously (see Figure 4a). Indeed, on a single core, the PIO transfers are serialized; so the NICs can not perform their communication requests in parallel. Considering the increase of the number of cores, an opportunity to bypass this technical constraint is to exploit those cores in order to perform the transfers in parallel on different cores. This would discharge the core where the communication is initiated as in Figure 4c. On the temporal diagram on Figure 4, the case (a) corresponds to the greedy algorithm, the case (b) to the case where the communication is aggregated in order to be sent over the fastest available network, and the case (c) to the offloading of the PIO transfers onto another core. In the last case, there is an initialization time introduced by the detection of the available cores, and by the actual setup of the copy on the remote core. It is however worth noting that the cost of this synchronization overwhelms the PIO transfer time of tiny messages on a single core, thus making offloading counterproductive in that case.

III. Highlights of the implementation

In this Section, we present the implementation of our efficient multirail communication strategy. It has been implemented in the NewMadeleine communication library using the PIOMan I/O manager.

A. NewMadeleine with the PIOMan I/O Manager

a) NewMadeleine*:* Our communication library for high performance networks is called NewMadeleine [5]. It is available over MX/Myrinet, Verbs/InfiniBand, Elan/QsNet, and TCP/Ethernet. It aims at applying dynamic scheduling optimizations on multiple communication flows such as reordering, aggregation, multirail distribution, etc. Its running

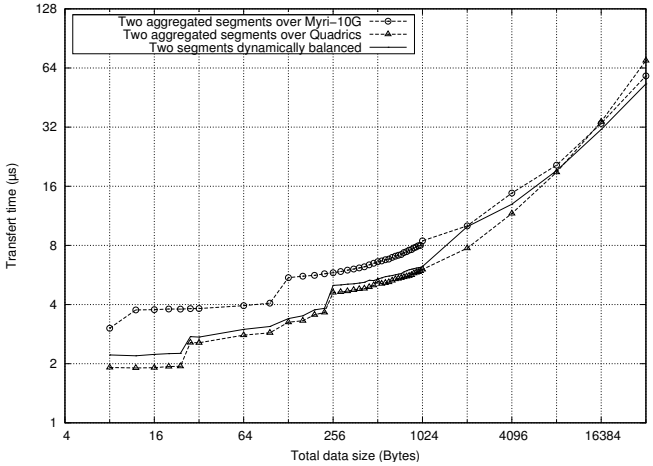

Fig. 3. Performance of the greedy balancing strategy.

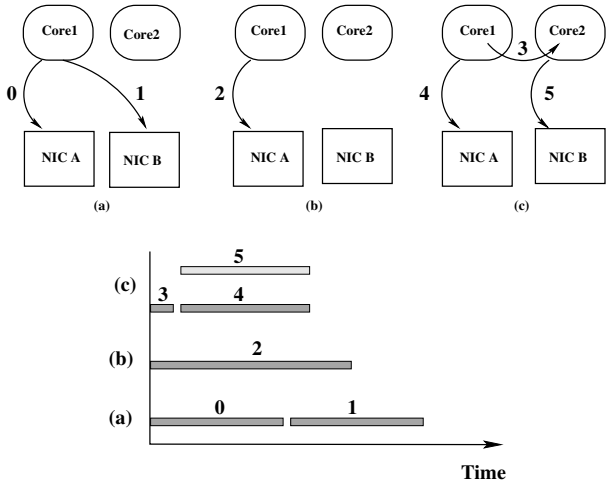

Fig. 4. PIO transfer combinations.

behavior is totally synchronized with the activity of the underlying NICs, in contrast to more classical communication libraries which are totally driven by the application. NewMadeleine has a 3-layers architecture as depicted in Figure 5. The application enqueues packets into a list and immediately returns to computing. The packet scheduler is only activated when a NIC becomes idle in order to feed it.

b) Marcel*:* Marcel [6] features a two-level thread scheduler that achieves the performance of a user-level thread package while being able to exploit SMP machines. The architecture of Marcel was carefully designed to support a high number of threads and to efficiently exploit hierarchical architectures. Marcel extensively relies on the concept of *tasklets* [7]. Tasklets have been introduced in operating systems to defer treatments that cannot be performed within an interrupt handler. Tasklets have a very high priority, meaning that they are executed as soon as the scheduler reaches a point where it is safe to let them run.

c) PIOMan*:* The progress engine of the PM2 software suite is PIOMan [8]. It performs as an event detector. It aims at providing the other software components with a service that guarantees a predefined level of reactivity to I/O events. PIOMan is able to balance the event detection processing over the whole machine and thus works closely with the Marcel

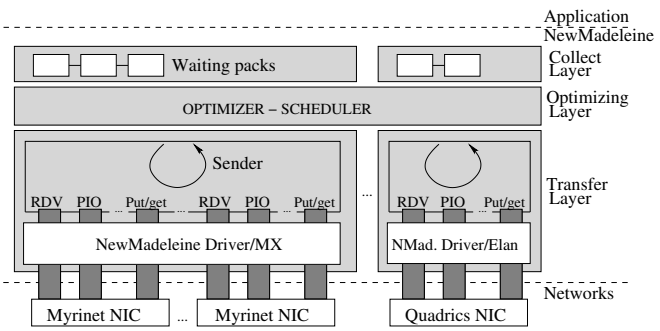

Fig. 5. NewMadeleine architecture.

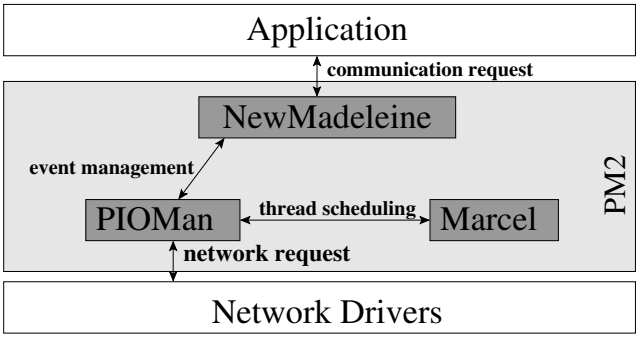

Fig. 6. The PM2 software suite.

thread scheduler which provides information on the running threads and the available CPUs. This way, PIOMAN is able to choose the most appropriate method (polling or interrupt-based blocking call) depending on the context (number of computing threads, available CPUs, etc.) to ensure a high level of reactivity. PIOMAN uses the tasklet mechanism provided by MARCEL to execute detection methods on the most suitable CPUs. MARCEL also schedules PIOMAN on some triggers (CPU idleness, context switches, timer interrupts, etc.) so as to ensure a fast detection of communication events. PIOMAN is generic and is not bound to any particular network or communication library; however, in this paper we focus on the use of PIOMAN by NEWMADELEINE.

B. Implementation of the strategy

Thanks to its layered architecture depicted in Figure 5, the way to describe how a communication has to be performed only depends on the selected strategy. Thus, the features proposed in this article are mainly organized around the implementation of a new NEWMADELEINE optimization strategy which actually is a *plug-in* called to gather the data requests and interrogated by the lower layer in order to know what to do at the appropriate time.

In the article context, the strategy is invoked at several critical moments: when a NIC becomes idle in order to select a new packet to send, when a *rendezvous* request has just been received and just before managing the emission of an eager packet. In this latter case, the strategy is interrogated and determines, knowing the NICs and cores activity and from the performance profile of the network drivers, what is the best combination of data transfers. So, as the goal is to provide parallel sending over different cores, the strategy splits the data in $\min\{$`number of idle NICs`, `number of idle cores`$\}$ chunks at most, each of them is then sent over a different NIC from a different core.

C. Network samplings

All the strategy decisions are based on the possibility to predict the duration of a transfer on each of the NICs. Instead of simply relying on the usual bandwidth and latency parameters provided by the vendors, an accurate profile of each NIC is performed at the initialization of NEWMADELEINE. Such a profile is measured with the help of a set of benchmarks that were designed for that purpose. For instance, as latency and bandwidth depend on the data size, those parameters are measured for various sizes (*e.g* powers of 2).

With this sampling, NEWMADELEINE estimates the transfer time of a communication given its size for each of the NICs, using the following algorithm. First, the strategy accesses the results of the sampling measurements through structures initialized at the launch of NEWMADELEINE. Second, the sampled sizes that are the closest to the message size are retrieved, for instance using a logarithm in the case of power of 2 samples. Finally, the estimated transfer time is computed by the mean of a linear interpolation.

Such sampling measurements can also be used to determine other parameters such as *rendezvous* threshold for various NICs.

D. Sending simultaneously small packets

In a previous paper [4] we noticed that using a single core to transmit small messages over several networks in parallel was not efficient. This is due to the CPU-consuming copies involved when sending eager packets. The NEWMADELEINE communication library being multithreaded, it is able to exploit several cores to simultaneously send chunks of message over different NICs as depicted in Figure 7.

Once the distribution of the data across parallel networks has been determined by the strategy, each message chunk is registered. Important information (data pointer, message size, chosen network, etc.) is stored in a to-be-sent list and idle cores are signaled that some requests need to be sent. The application can then resume its computation.

In case a thread is being scheduled on an idle core while the strategy computes the split ratio, a signal is sent in order to preempt the thread and to let the packet submission occur. As remote cores detect the registered requests, callbacks are executed: one of the requests is selected and the corresponding data is sent over the given network.

This method is costly because of the communication between the strategy and the remote cores. This communication has been evaluated to 3 μs – 6 μs if a thread has to be preempted by a signal. Transmitting tiny eager packets in parallel is thus inappropriate, but this mechanism appears to be useful to send medium-sized eager messages.

IV. EVALUATION

In this Section, we present preliminary results obtained by using the multithreaded NEWMADELEINE communication library. Our experiments have been carried out on a set of two dual dual-core Opteron boxes. The experiments we have conducted are using a combination of an Elan/QsNetII Quadrics network [9] and a MX/Myri-10G Myricom network [10].

A. Distributing data among heterogeneous networks

To evaluate the ability of NEWMADELEINE to distribute data across heterogeneous networks, we use a classical ping-pong program and we measure the obtained bandwidth. The

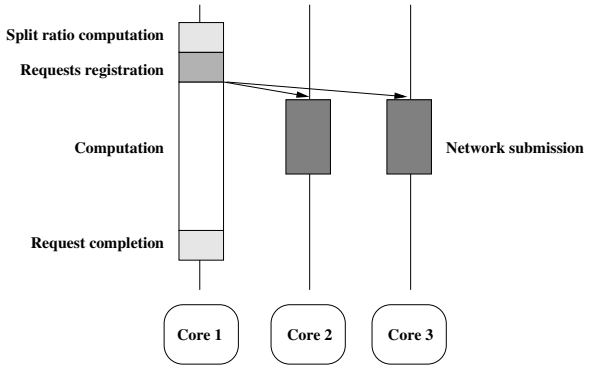

Fig. 7. Sending eager packets over parallel networks

Figure 8 shows that by sending the whole message through Myri-10G, a 1170 MB/s bandwidth is reached whereas sending the message through Quadrics permits to reach 837 MB/s. In order to fully exploit the available networks, it is necessary to split the message and to transmit each chunk of message on a separate network.

By splitting the message in two chunks which have the same size (Iso-split), the measured bandwidth arises up to 1670 MB/s, which is far from the theoretical aggregate bandwidth of approximately 2 GB/s. This is due to the performance differences between Myri-10G and Quadrics: for example, when the application sends a 4 MB message, a 2 MB chunk of message is sent over Myri-10G in approximately 1730 μs while another 2 MB chunk is sent through Quadrics in 2400 μs. The Myri-10G network is thus unused for 670 μs.

Taking into account the actual performance of networks allows us to compute a more precise split ratio: as the Myri-10G network is faster than Quadrics, the message is split in such a way that Myri-10G sends more data than Quadrics. The sampling-based hetero-split reaches a maximum bandwidth of 1987 MB/s which is very close to the theoretical maximum bandwidth. This higher bandwidth is due to a more equilibrated workload: for example, when the application sends a 4 MB message, a 2437 KB chunk of message is sent through Myri-10G in 1999 μs whereas a 1757 KB chunk is sent over Quadrics in 2001 μs that considerably minimizing the transfer time.

B. Using idle cores to send small messages across several networks

In a previous work [4], we noticed that splitting eager messages to send the resulting chunks through several networks was not efficient as these packets require CPU intensive memory copies. Sending chunks of a message over different networks would boil down to a serialized data transfer. By exploiting multicore architectures, it is though possible to send chunks of messages in parallel as explained in Section II-C.

Unfortunately, the implementation of this technique is still too costly because of synchronization issues. Yet, an optimized implementation would hopefully achieve better results. To estimate the potential benefits of this method, we perform

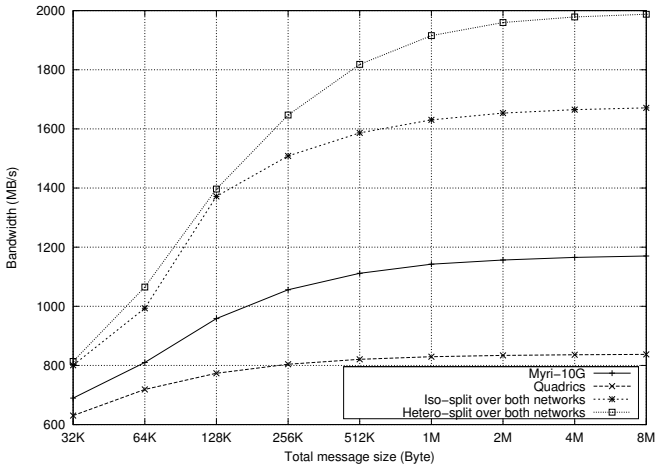

Fig. 8. Message splitting - Bandwidth

a latency benchmark that measures the performance of both Myri-10G and Quadrics networks. In order to estimate the results that could be obtained by splitting messages and by using idle cores to submit eager message chunks, we proceed as described in equation (1): a split ratio is computed and a transfer duration T_D is determined for each network. The offloading cost T_O – which corresponds to the time elapsed between the computation of the split ratio and the beginning of the submission to a network – is then added to the maximum computed time. T_O has been evaluated to 3 μs that are mainly due to communication between cores and synchronization.

$$T(size) = T_O + max(T_D(size.ratio, N_1), \\ T_D(size(1 - ratio), N_2)) \quad (1)$$

Figure 9 presents the results of this estimation. Splitting small messages (*i.e.* smaller than 4 KB) appears to be costly because of the initialization of the tasklet. But as messages become larger, sending chunks in parallel becomes more and more interesting and permits to reduce by up to 30% the transfer duration.

V. CONCLUSION

The development of multicore systems in clusters leads to a bottleneck at the interconnect subsystem level. The use of independent parallel networks permits to increase the communication performance and to bypass the problem by providing an enhanced cumulated bandwidth. But this aggregate bandwidth remains theoretical as few communication libraries are able to exploit efficiently multiple networks in parallel.

We proposed a model able to transmit messages over several parallel links efficiently by evaluating networks capabilities. The implementation of this model within the NEWMADELEINE communication library estimates transfers duration, predicts when a NIC will become idle and thus is able to split messages to send them across parallel networks so that the transfer duration is minimized. The experiments conducted over Myri-10G and Quadrics exhibit very good performance

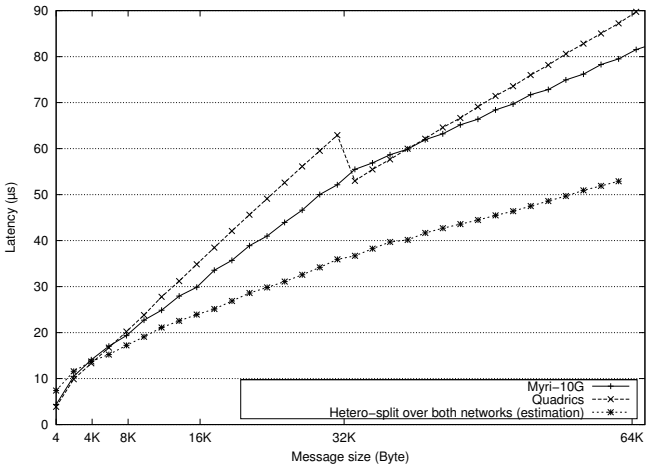

Fig. 9. Splitting small messages - Latency

as NEWMADELEINE almost reaches the theoretical aggregate bandwidth.

The multithreaded communication library that implements our model exploits multicore machines to process parallel PIO transfers. Despite the overhead problem we face in this preliminary implementation, an estimation of a future implementation's behavior shows that it should be possible to reduce latency by up to 30 % in the small message case. Thus, the multithreading subsystem of the NEWMADELEINE communication library has to be improved to reach this performance level.

In the short term, we plan to integrate NEWMADELEINE in the MPICH2-Nemesis [11] software stack so as to use the multirail capabilities and the multithreaded communication system within the widespread MPI implementation. This will allow us to further experiments and enhance our techniques onto a wide range of applications.

REFERENCES

[1] H. Nakashima, "T2k open supercomputer: Inter-university and interdisciplinary collaboration on the new generation supercomputer," *Informatics Education and Research for Knowledge-Circulating Society, 2008. ICKS 2008. International Conference on*, pp. 137–142, Jan. 2008.

[2] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits, "Using multirail networks in high-performance clusters," *Cluster Computing, 2001. Proceedings. 2001 IEEE International Conference on*, pp. 15–24, 2001.

[3] 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.

[4] 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, held in conjunction with IPDPS 2007*, Long Beach, California, USA, March 2007. [Online]. Available: http://hal.inria.fr/inria-00126254

[5] O. Aumage, E. Brunet, N. Furmento, and R. Namyst, "Newmadeleine: a fast communication scheduling engine for high performance networks," in *CAC 2007: Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2007*, Long Beach, California, USA, March 2007, also available as LaBRI Report 1421-07 and INRIA RR-6085. [Online]. Available: http://hal.inria.fr/inria-00127356

[6] Runtime Team, LaBRI-Inria Bordeaux – Sud-Ouest, "Marcel: A POSIX-compliant thread library for hierarchical multiprocessor machines," 2007, http://runtime.futurs.inria.fr/marcel/.

[7] M. Wilcox, "I'll do it later: Softirqs, tasklets, bottom halves, task queues, work queues and timers," in *Linux.conf.au*. Perth, Australia: The University of Western Australia, January 2003.

[8] F. Trahay, A. Denis, O. Aumage, and R. Namyst, "Improving reactivity and communication overlap in mpi using a generic i/o manager," in *EuroPVM/MPI*, ser. Lecture Notes in Computer Science, F. Cappello, T. Herault, and J. Dongarra, Eds., vol. Recent Advances in Parallel Virtual Machine and Message Passing Interface, no. 4757. Springer, 2007, pp. 170–177. [Online]. Available: http://hal.inria.fr/inria-00177167

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

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

[11] D. Buntinas, G. Mercier, and W. Gropp, "Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem," *Parallel Computing, Selected Papers from EuroPVM/MPI 2006*, vol. 33, no. 9, pp. 634–644, Sep. 2007.