Optimizing Message Aggregation for Parallel Simulation on High Performance Clusters

C. D. Pham
RHDAC
Université Claude Bernard Lyon I
FRANCE
cpham@lhpca.univ-lyon1.fr

C. Albrecht
Institut für Technische Informatik
Medizinische Universität zu Lübeck
GERMANY
albrecht@iti.mu-luebeck.de

Abstract

High performance clusters (HPCs) based on commodity hardware are becoming more and more popular in the parallel computing community. These new platforms offer a hardware capable of a very low latency and a very high throughput at an unbeatable cost, making them attractive for a large variety of parallel and distributed applications. With adequate communication software, HPCs have the potential to achieve a level of performance similar to massively parallel computers. However, for parallel applications that present a high communication/computation ratio, it is still essential to provide the lowest latency in order to minimize the communication overhead. In this paper, we are investigating message aggregation techniques to improve parallel simulations of fine-grain ATM communication network models. Even if message aggregation is a well-known solution for improving the communication performance of high latency interconnection networks, the complex interaction between message aggregation and the underlying communication software is often ignored. We show that message aggregation must carefully take into account the characteristics of the communication software to be efficient on an HPC. This methodology can be applied as a preliminary step to tune a message aggregation algorithm for a given combination of hardware architecture and communication software layer.

1. Introduction

Parallel simulation techniques have been proposed as a solution for reducing the simulation time and for increasing the system size of studied models. It usually assumes that the system to be simulated can be spatially partitioned into disjoint sub-systems. Each sub-system is simulated by an associated Logical Process (LP) on a dedicated processor. As the simulation is distributed, message passing is a common way to exchange information or events between the different LPs. The messages are time stamped by the sending LP according to its *Local Virtual Time*. At the receiving side, they are put in a local *Future Event List* to be processed. Synchronization of messages for causality constraints can either be performed conservatively [2] or optimistically [9]. Other methods to parallelize a simulation exist but their applications are limited [5]. Simulation of communication networks usually requires a large amount of events to be simulated as these events typically represent packets in the real system. High-speed network models emphasize this property as the number of messages increases dramatically. Therefore, parallel simulations are usually performed on parallel machines in order to benefit from the lowest latency available.

As technological advances have made distributed systems based on commodity hardware available at a very low price, several attempts to bring parallel simulation techniques on high performance clusters (HPCs), such as Myrinet [1], have been successfully conducted [3, 13, 11]. Although the speedups achieved on these HPCs can be very close to those obtained on state-of-the art parallel machines [11], and that for an unbeatable price, it is still desirable to optimize the communication between remote LPs in order to reduce the communication overhead. Parallel simulation of communication network models is an area where all possible optimizations on communication are generally welcome because these models typically have a high communication to computation ratio. These optimizations can be performed at the lowest level in the communication software [4] or in the application at a higher level. We are focusing on application-level optimizations and more precisely on a well-known optimization, called message aggregation, that groups several small messages for the same destination into a single larger one [8, 3]. The rational behind message aggregation is that it is less costly to send an

M byte message than to send n times an m byte message with $n * m = M$. In the case of fine-grain applications, message aggregation can become very efficient. In [3], the authors have performed extensive experiments on message aggregation with a Time Warp based simulator with MPI as the communication layer. Static and dynamic message aggregation policies were used on several application models. The parallel simulations were executed with 4 LPs on both an Ethernet-based network of workstations (NOW) of two 4-processor SMP and a single 4-processor SMP. Results show that application-level message aggregation can reduce the simulation time by around 30% in the best case on the NOW. The static aggregation strategy shows very close performances compare to the dynamic strategy for the interesting area of the speedup curve. Investigating the use of high performance clusters (3 Pentium Pro interconnected by Fast Ethernet and Myrinet), the authors in [13] have shown that message aggregation presents disappointing results for Myrinet and encouraging results for Fast Ethernet. The reason they mentioned is that the Myrinet cluster was exploited using the MPI-BIP communication software that provides a much smaller latency, hence less potential of improvements, than the TCPMPL layer (the authors' implementation of TCP/IP-based communication layer) developed on Fast Ethernet. Therefore, message aggregation is reported to behave badly on low latency environments.

The work described in this paper goes a step further in the investigation of message aggregation for parallel simulation on a Myrinet-based HPC. In particular, we show that the behavior of the lowest level of communication should be taken very carefully into account in order to achieve efficient message aggregation. Therefore, there is a tight relation between the application optimizations and the capabilities of the communication layer. In addition, as more nodes than the previous study in [13] are available for experimentations, scalability issues of message aggregation are also discussed. The remainder of the paper is organized as follows: Section 2 presents the high performance environment based on Myrinet hardware and the BIP and MPI-BIP communication software. Section 3 presents the high-speed communication network model used to test the message aggregation strategy. This application naturally exhibits very small computation granularity (processing time of an event) that dramatically stresses the communication layer. Section 4 presents describes the message aggregation strategy and the tuning methodology. Results are presented in Section 5 and the conclusion is given in Section 6.

2. The high performance cluster environment

Recently a number of high performance local area networks (LANs) have been proposed to achieve transfer of data over the gigabits rate. HPC based on these technologies can benefit from a much higher bandwidth and typically much smaller latencies than the previous generation of general purpose interconnection networks. Our experimental platform is based on Myrinet hardware [1]. The Myricom LAN was chosen for its performance over the Gbits/s, its affordable price and its software openness. There are several features that makes this kind of technology much more suitable than a typical commodity network:

- the hardware provides an end-to-end flow control that guarantees a reliable delivery and alleviates the problem of implementing in software the reliability on top on an lossy channel. As message losses are exceptional, it is possible to use algorithms that focus on very low overheads in the normal case,
- the interface card has a general purpose processor to handle most of the communication activity without interrupting the main processor. Efficient overlapping of communication/computation and collective operations can then be performed.
- the interface card has up to one megabyte of buffer memory. With the network being as fast as the PCI bus, this memory isolates transfers on the network from transfers on the PCI bus, so that they will not be limited by the slowest one at all times.

2.1. BIP

The BIP (Basic Interface for Parallelism) communication layer [12] is used for message exchanges between nodes of the HPC. The BIP communication layer provides only a protocol with low-level functionalities. Although specialized parallel applications could interface directly with it, it is intended as the base of other protocol layers like IP, and higher level APIs like the well established MPI [7]. Highly optimized, the raw communication performance for BIP is about $5\mu s$ latency one-way with a distinction between small and large messages at 256 bytes as illustrated by figure 1 with a ping-pong test. The distinction between small and large messages can be clearly seen by a jump in the latency curve. More details on that distinction will be given later on. The low latency allows applications to scale more by allowing the use of a finer grain of computation. This is particularly important for parallel simulation of communication networks as it will be explained in the next section.

2.2. MPI-BIP

MPI-BIP is an interface to the MPI APIs based on the MPICH implementation [6]. Although it adds additional overheads compared to BIP, this MPI interface allows the

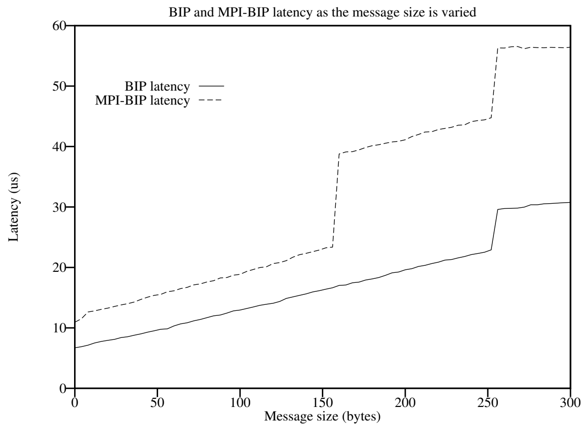

Figure 1. BIP and MPI-BIP latency as the message size is varied.

immense MPI user community to benefit from the small latency offered by the Myrinet hardware. One important feature over BIP is the embedded flow control that alleviates the user from the hassle of low-level operations needed in native BIP. MPI-BIP is the interface we use in all the experimentations described in this paper. Figure 1 shows the latency of MPI-BIP where three interesting parts can be identified.

In the first part, for message size up to 156 bytes, the MPI-BIP latency compared to the one of BIP is about 1.7 times higher. Then we can see a jump in the latency for message size starting at 160 bytes where the MPI-BIP latency is about 2.1 times greater than the BIP latency. This second part is explained by a change in how MPICH handles messages (the threshold for the strategy change is implementation dependent): (i) for a message size from 0 to 156 bytes, MPICH asynchronously sends the message with the MPI header (the envelope in the MPI terminology), (ii) for a message size greater than 156 bytes, MPICH first sends the header, then waits for an acknowledgment and then sends the data. This is call the rendez-vous mode [8]. The second jump in the latency that occurs at 256 bytes is due to the BIP distinction between small and large messages. The jump with MPI-BIP is slightly greater than the jump with BIP. The fact that the jump occurs at 256 bytes, and not before as can be expected due to the additional MPI header, is because the header has already been sent since MPICH is already in the rendez-vous mode for a 256 byte message. Nevertheless, the latency provided by MPI-BIP on Myrinet remains much smaller than the latency of more traditional combination of communication layers and interconnection networks, e.g. TCP/IP and Ethernet, TCPMPL and Fast Ethernet [3]. Note that the latency of MPI-BIP depends in a large part on the processor speed as the network part of the latency is very small. For instance, as the processor speed is increased, the latency of MPI-BIP is decreased (this is less the case with BIP since BIP grabs very few cycles of the main board processor). The values shown in figure 1 have been obtained on a network of Intel Pentium Pro 200 MHz. Tests on Intel Pentium II 333 MHz show that the MPI-BIP latency can be 20% smaller.

2.3. The experimental environment

The experimental cluster consists of 12 nodes interconnected by three 8-ports Myrinet switches. Each node is an Intel Pentium Pro 200 MHz with 64MBytes of RAM and a 440FX chipsets. The myrinet board contains a LANai 4.1 and 256Kbytes of memory. The test-bed is provided by the LHPC (Laboratoire pour les Hautes Performances en Calcul), a cooperation between ENS-Lyon and Matra Système Information.

3. The parallel simulation application

The application we consider is a communication network model based on the Asynchronous Transfer Mode (ATM) technology. The network model we consider consists of 78 ATM switches and traffic sources. The sources periodically open and close connections to random destinations. Cells are sent by traffic sources and are transfered to the destination by the ATM switches. The system under study includes a routing algorithm with the introduction of link cost and dynamic routing functions that provide load-balancing routing features. The simulation of a routing algorithm implies to simulate (i) the mechanism that consists in constructing and updating the routing tables and (ii) the flow of cells that is handled by the network [10]. The parallelization process of the network model consists in spatially partitioning the entire network in n regions and to assign each region to an LP on a processor. Cell transfers from one region to another are represented by time stamped messages exchanged between the LPs. The message synchronization is performed with conservative rules [2] because the application exhibits good lookahead with the link propagation delay.

The life cycle of the simulated objects in the parallel simulator basically consists in processing internal messages and generating messages. Internal messages are taken from a local global event list and processed in timestamp order. Generated messages that have to go off the processor are sent to the destination using the communication system, otherwise they are directly inserted as new internal events. The destination is identified by a couple ($node$, ic) where ic is the logical input channel for the receiver. External messages are received in the communication layer buffers and periodically inserted in the local global event list to become internal events. The ic value of each message is used to compute a safe window for event processing, according to the conservative rules. When an LP has no more eligible internal messages to process (i.e., no more messages) or it has reached the upper bound of the time window, it would send out a null-message and would get all external events from the communication buffers.

From the parallel simulation perspective, this application presents a very small granularity and requires a lot of small messages to be exchanged between processors (as opposed to more traditional high-performance computing applications where the computation part is larger). It is not unusual to send several millions of messages per minute of wall clock time! These messages are either model messages (cell exchanges between ATM switches), or synchronization messages (null-messages from the conservative kernel). Therefore the application typically requires very low latencies from the communication system.

4. Message aggregation

Message aggregation aims at reducing the communication overhead by sending in a single larger message several small messages. There is a trade-off on how long to keep aggregating in order to avoid useless waiting as a side-effect at the receiver side. As shown in [3], it is certainly not desirable to keep an aggregate too long. However, in our application, as in most fine grained applications, the polling for new external events is done periodically and not after every event processing. More precisely, our conservative kernel computes a safe upper bound time window for event processing and only polls for remote messages at the end of an event processing cycle. In this case, the aggregation at the sender side, that obviously delays the delivering of events, will hopefully have a limited side-effect on the application.

4.1. Limitation on the size of the aggregate

It has been shown that the overall latency for sending a message can be split in a constant part and a size-dependent part. As mentioned before, message aggregation is motivated by the fact that the constant part is relatively high making several small messages much more costly than a single large one. For our application, the regular message size is 42 bytes. A simple ping-pong test gives $15.07\mu s$ for the latency (software+hardware). Figure 2 illustrates the gain in latency between a theoretical unaggregated `send` and a theoretical aggregated `send` based on the MPI-BIP latency evolution depicted previously in figure 1. The difference between the two curves clearly shows the weight of the constant part.

Aggregating 2 messages already saves 39% of overhead. More than 50% could be saved with 3 messages. When aggregating 4 messages, the gain goes down significantly.

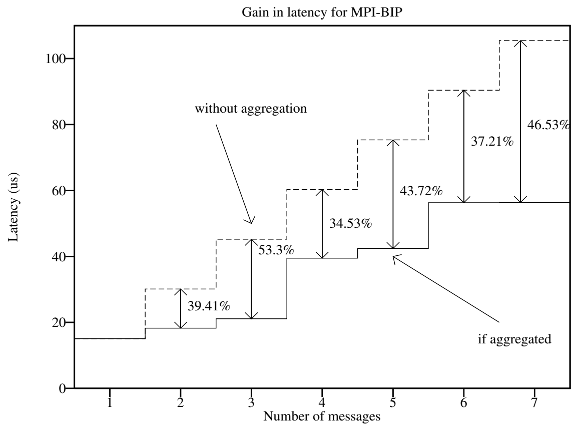

Figure 2. Gain in latency for MPI-BIP.

The explanation is given by the evolution of the message latency of MPI-BIP (figure 1) since the size of a 4-message aggregate is greater than 156 bytes. One could imagine a strategy that would construct aggregates with a size that would give the maximum gain (3, 5, 7 and above for example). A timer would send the aggregate if the ideal size has not been reached in a given time window. The main reason not to do so must be taken from the internal behavior of the communication layer: MPI-BIP switches from the MPICH "small message" send strategy to the MPICH send strategy with rendez-vous for message size greater than 156 bytes as described previously. The rendez-vous mode adds an additional synchronization overhead if the reception is posted asynchronously, which is the case in the conservative kernel (external messages are taken from the reception buffer only at the end of the time window). It is not really desirable to post a reception call at the end of every message processing nor it is desirable to make the sending code more complex (asynchronous sends are possible but then periodic checking that the send buffer is free is overwhelming). Figure 3 shows the time spent in an `MPI_Send` call (time needed for the call to returns, not necessarily the time needed to sent the data) when the `MPI_Recv` call is posted asynchronously after a δ time delay expressed in μs, which is a typical behavior in most parallel simulators.

From figure 3, it can be seen that for a message size less than 156 bytes, the time spent in the `MPI_Send` call is independent from the posting time of the corresponding `MPI_Recv` call because MPICH uses asynchronous send with buffer copying. Therefore the call overhead is only proportional to the copying time (about $10\mu s$ for 156 bytes) and the send call returns very quickly. For a message size greater than 156 bytes, the send call overhead jumps to $34.35\mu s$ for 160 bytes of data in the best case (synchronous ping-pong, $\delta = 0$). As more asynchronism is added by increasing δ, the jump between 156 bytes and 160 bytes

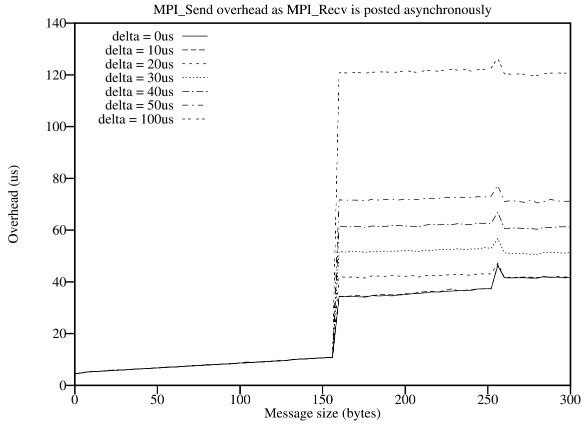

Figure 3. MPI_Send overhead as δ is varied.

increases dramatically. After the jump, the send call overhead remains approximately constant because no intermediate copying is involved in the rendez-vous mode. The small peak at 256 bytes comes from BIP and is not related to the MPICH implementation, but we do not have a satisfying explanation yet (the normal BIP distinction between small and large message would normally produce a jump, not a peak). Figure 3 is important because an efficient overlapping of computation and communication depends on the send call overhead: the faster the send call returns, the faster new computations can be started. With a computation grain of about 5 μs for our application on a Pentium Pro 200MHz, a δ of 50 μs would mean that the application polls the communication layer every 10 messages in the best case. As polling every 20 messages (δ=100 μs) is not unusual in such applications, using an aggregate size greater than 156 bytes dramatically reduces the computation/communication overlapping. Infrequent polling can worsen the situation.

Therefore, in order to achieve the maximum of performance (always asynchronous send to minimize the send call overhead), it is important to keep the size of an application aggregate below the critical 156 bytes threshold. In this way, the posting time of reception calls has no impact on the performances. Given this constraint, it is not possible to use an age criterion for the sending of an aggregate as proposed in [3]. Instead, a maximum aggregate size $S_{max} = 156$ is used to trigger the sending of an aggregate. Details of rules for constructing and sending aggregates are given below.

4.2. Static message aggregation strategy

Based on the results obtained in [3] where it can seen that dynamic aggregation does not improve significantly the performances, we perform message aggregation with a static aggregate size. Let S_{max} be the maximum size allowed for an aggregate: messages from the application are stored in temporary buffers, one per destination processor. When a new message needs to be sent, the final aggregate size S is computed and checked against S_{max}. If $S > S_{max}$ then the current aggregate is sent and the new message initializes a new aggregate. If $S < S_{max}$ then the new message is added to the current aggregate. Then, if $S > S_{max} - min\_event\_size$ the aggregate containing the new message is sent immediately to avoid useless waiting as there will be no more space left for any additional message. Events of size equal to min_event_size can be either a null-message or a model routing control message.

Rules for sending aggregates are broken when the conservative kernel sends null-messages: these special messages immediately trigger the sending of an aggregate, whatever the aggregate size is because we want to propagate the simulation time as fast as possible. This is very similar to the handling of anti-messages performed in [3]. Note that since null-messages are sent at the end of every event processing cycle, an aggregate is guaranteed to be sent when the LP has no more events to process. In other word, an aggregate is guaranteed to be sent after a finite time that may not be optimal but in practice, as the maximum size of the aggregate is relatively small (156 bytes), there is very little waiting. With a larger aggregate size, a security timer would have been necessary to limit the waiting at the receiver side.

5. Experimental results

In this section, we present the performance of the message aggregation technique presented so far. Since BIP is single-user, there is no interferences with other processes during the runs.

5.1. Performance of message aggregation

Table 1 presents the message statistics collected during a run. *total sent* is the number of physical messages sent: *un.* is for the unaggregated version while *w/A* is with message aggregation. *efficiency* shows how many messages (in percentage) have been saved with the aggregation technique.

	2 pr.	4 pr.	6 pr.	8 pr.	10 pr.
total sent un.	1300337	3643078	3420847	5101153	5378893
total sent w/A	758595	1820562	1588672	2293524	2488476
efficiency	41.6%	50%	54%	55%	54%
null-msg w/A	568620	1344526	1153182	1520551	1751671
null w/0	83.95%	64.18%	54.97%	52.63%	56.90%
null w/1	5.55%	14.86%	16.80%	20.08%	17.18%
null w/2	5.50%	11.35%	15.91%	16.10%	13.99%
null w/3	4.98%	9.59%	12.29%	11.16%	11.92%

Table 1. Message aggregation statistics.

Using message aggregation can reduce the number of physical messages by half in most cases. Table 1 also shows the number of null-messages sent and the size repartition of the aggregates containing a null-message (remember that a null-message makes the current aggregate to be sent immediately; not all aggregates contain a null-message). On 2 processors, message aggregation is a bit less efficient because there is a higher number of null-messages sent alone (83.95%). This is explained as follows: 2 processors means only one external communication channel between the 2 LPs. In this situation, the number of events processed per cycle is reduced because the event execution window is smaller (external events come only from 1 processor). It is possible to avoid this but it would suppose a special optimization for 2-processor executions. On more than 2 processors null-messages sent alone are still predominant but as the event execution window is larger (more external sources) their number decreases significantly.

Figure 4 compares the parallel simulation execution time of the unaggregated send version to the aggregated one as the number of processors is varied. Message aggregation can efficiently reduce the simulation time by up to 42% when 10 processors are used. Figure 5 shows the size repartition of the aggregates. As no more than 156 bytes aggregates can be sent, the maximum number of messages in an aggregate is 4: 3 regular messages (42 bytes each) + 1 small message (24 bytes). The small message can be either a null-message or a model routing control message.

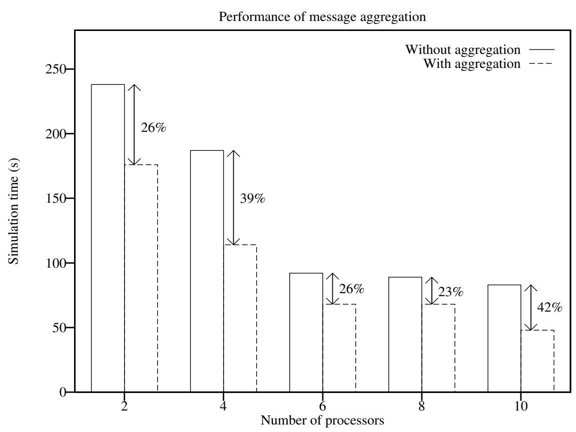

Figure 4. Performance of message aggregation as the number of processors is varied.

The proportion of 1-message aggregates is high in the 2-processor execution because of the high number of null-messages sent alone as explained previously (a 1-message aggregate is always a null-message). For the other cases, the proportion of 1-message and 3-message aggregates are approximately the same. It is interesting to note that 100% of 4-message aggregates are composed of 3 regular messages + 1 null-message. 3-message aggregates with no null-message are sent after an extra delay because there is no way to know in advance if the next message would make the current aggregate too big.

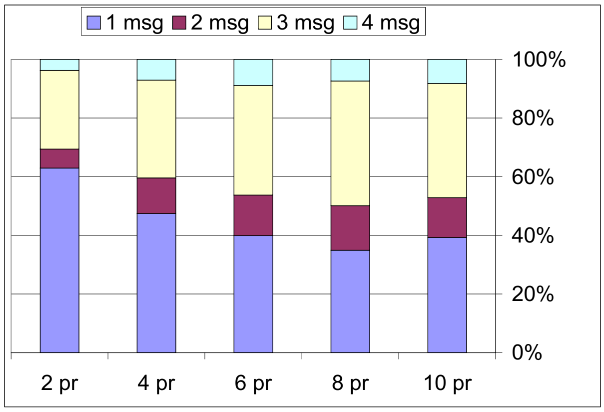

Figure 5. Repartition of aggregate size in number of messages.

5.2. Reception buffer size, flow control and their impact on the parallel simulator

Regarding scalability when the problem size increases, preliminary experiments with a network model of 156 switches have shown that the communication sub-layer must be scaled accordingly (the sequential version also requires a machine with 120 MBytes of memory whereas the parallel version could use 64 MBytes nodes). For instance, if we keep the same number of processors for a larger problem size, i.e. large number of ATM switches, the number of messages sent from one processors to another can increase dramatically. At the receiving side, the MPICH reception buffers could be rapidly saturated resulting in a tight flow control and additional buffering at the sender side that can dramatically increase the simulation time. This observation on a 156-switch model leads us to think that the same problem may have happen in the 78-switch model as well but without being noticed. Therefore we started looking at the MPICH flow control a bit closer.

The standard MPI-BIP distribution defines a maximum number of nodes of 48 with only 6 reception buffers per nodes (1 buffer = 1 message regardless of its size). Table 2 presents the message statistics of message aggregation when the reception buffer is increased. Due to memory limitations of MPI-BIP, we could not increase the number of buffers to more than 32 when a maximum of 8 processors are used (8x32) and to more than 24 when a maximum of 10 processors are used (10x24).

The efficiency of message aggregation is a bit better than the previous case and this will be explained later on. One

	2 pr.	4 pr.	6 pr.	8 pr.	10 pr.
total sent un.	852890	2549018	2761616	3940195	4257933
total sent w/A	368800	963202	1110277	1506972	1656210
efficiency	57%	62%	(59%)	(62%)	(61%)
null-msg w/A	142980	248727	493663	516347	619830
null w/0	83.57%	62.25%	53.83%	49.94%	52.66%
null w/1	5.02%	13.34%	17.04%	17.99%	16.79%
null w/2	4.16%	12.52%	14.23%	17.90%	15.64%
null w/3	7.23%	11.87%	14.88%	14.15%	14.89%

Table 2. Message aggregation statistics - MPICH 8x32 and 10x24.

thing that must be noted is that increasing the reception buffer size has the effect of reducing the number of null-messages in the simulation runs. This phenomenon that we did not expect can be explained as follows: a small number of reception buffers means that less remote messages are available when an LP polls the communication layer after having processed all its eligible events. Since less messages are available, the next cycle of event processing will contain a small number of events. Then, as null-messages are sent after each event processing cycle, more null-messages will be sent (note that this is not a problem of the parallel simulator kernel). With a larger number of reception buffers, more messages can be processed per cycle so null-messages are sent less frequently. As the traffic of null-messages is reduced, the parallel simulation performs better.

Figure 6 compares the parallel simulation execution time of the unaggregated send version to the aggregated one as the number of processors is varied. We can see from the figure that the execution time of the unaggregated version on a small number of processors (2 and 4) can decrease significantly when the number of reception buffers has been increased. This is mainly due to the smaller proportion of null-messages and because these cases were the most affected by the tight flow control (more simulation objects per processor and still only 6 reception buffers). Message aggregation appears less profitable in term of execution time than the previous case (when the number of reception buffers were left at 6) because the aggregated version shows nearly identical performance regardless of the reception buffer size. This even if the number of physical messages is reduced by about 50% on 2 and 4 processors, and by about 30% in the other cases. For instance, message aggregation on 4 processors shows only $1s$ of improvement for the simulation time! Here, the unaggregated version achieves the same level of performance than the aggregated one in the 48x6 case.

Figure 7 shows the size repartition of the aggregates for the 8x32 and 10x24 case. Once again, the maximum number of messages is 4 but the number of 3-message aggregates is by far predominant compared to the other aggregate size. This is because aggregates can reach their maximum

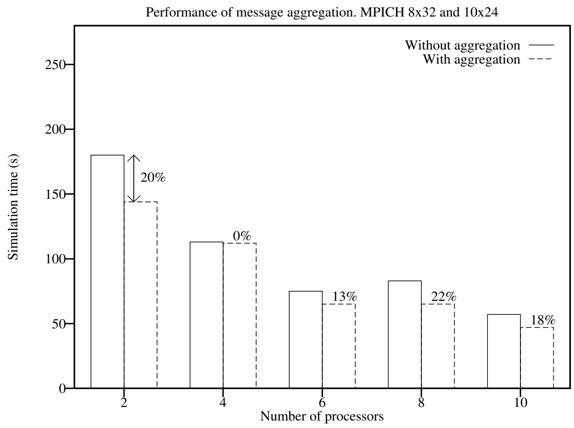

Figure 6. Performance of message aggregation - MPICH 8x32 and 10x24

size without being broken by a null-message. This explains the higher efficiency of message aggregation. Once again 100% of 4-message aggregates are composed of 3 regular messages + 1 null-message. Also, less than 10% of 3-message aggregates contain a null-message while more than 90% contain the maximum number of regular messages.

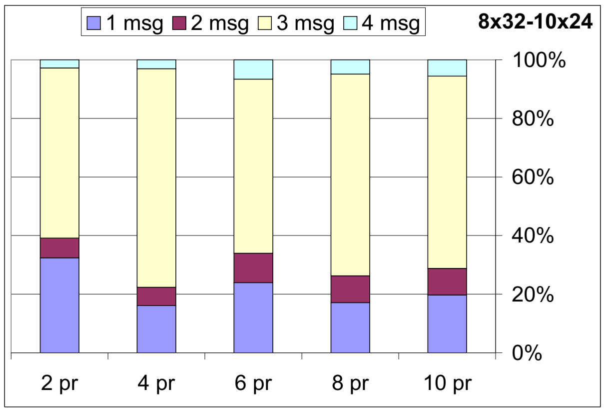

Figure 7. Repartition of aggregate size in number of messages.

5.3. Message aggregation or not?

With a small number of reception buffers, message aggregation is very efficient and can significantly reduce the simulation time. In this case, message aggregation counterbalances the effect of a small number of reception buffers by allowing more simulation events to be contained in a small number of physical messages. When the number of reception buffer increases, the unaggregated parallel simulator performs much better and message aggregation is less profitable. However, relying on the buffer availability is not a scalable solution for communication as pointed in [8]. Message aggregation is found almost insensitive to the reception buffer size and we believe that when clusters with 32 or 48 nodes, or more, will be commonly used, message aggregation will then offer a scalable solution to gracefully optimize the communication on the HPC.

6. Conclusions

In this paper, we have studied the performance of message aggregation on a low latency message passing environment. The idea was to look carefully at the low-level communication features to optimize the message aggregation strategy and to avoid aggregate sizes that present very high overhead. This method can be applied as a preliminary step to tune the message aggregation algorithm for a given combination of hardware architecture and communication software layer. The results show that message aggregation can be efficient even on a low latency architecture if the design of the aggregation strategy takes care of the lower communication layer behavior. In general, message aggregation also reduces the need of large reception buffers and therefore allows for more scalability. This issue is very important as we aim at simulating large-scale ATM network models containing hundreds of nodes.

High performance clusters have many features that make them already very promising. One of the features that is becoming popular is the generalization of communication card with embedded general processor (such as Myrinet or Gigabit-Ethernet). Message aggregation can be an easy way to increase the amount of parallelism since buffering is already performed. The next step would be to execute the sending operation of large messages by the communication card processor in order to achieve a higher level of parallelism for fine grain parallel applications.

References

[1] Boden, N. J., Cohen, D., Feldermann, R. E., Kulwik, A. E., Seitz, C. L., Seizovic, J. N., and Su, W. K. MYRINET: A Gigabit per second Local Area Network. *IEEE-Micro*, 15:29–36, February 1995.

[2] Chandy, K., Misra, J.: Distribution Simulation: A Case Study in Design and Verification of Distributed Programs. *Trans. on Soft. Eng.*, **5**(5) 440–452.

[3] Chetlur, M., Abu-ghazaleth, N., Radhakrishnan, R., Wilsey, P. A.: Optimizing Communication in Time Warp Simulators. *In Proceedings of PADS'98*, Banff, Canada, pp64-71.

[4] Chetlur, M., Sharma, G. D., Abu-Ghazaleh, N., Rajasekaran, U., Wilsey, P. A.: An Active Layer Extension to MPI. *In proceedings of the EuroPVM/MPI '98, (LNCS 1497)*, September 1998, pp97-104.

[5] Ferscha, A., Tripathi, S. K.: Parallel and Distributed Simulation of Discrete Event Systems. CR-TR-3366, Dept. of Comp. Science, Univ. of Maryland.

[6] Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. TR, Argonne National Laboratory, 1996.

[7] Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface. *MIT Press*, Cambridge, MA, 1994.

[8] Gropp, W., Lusk, E.: Tuning MPI Programs for Peak performance. `http://www.mcs.anl.gov/mpi`.

[9] Jefferson, D. R. : Virtual Time. *ACM Trans. on Prog. Lang. and Sys.*, **7**(3) (July 1985) 405–425.

[10] Pham, C. D., Brunst, H., Fdida, S.: Conservative Simulation of Load-Balanced Routing in a Large ATM Network Model. *In Proceedings of PADS'98*, Banff, Canada, pp142-149.

[11] Pham, C. D.: High Performance Clusters: A Promising Environment for Parallel Discrete Event Simulation. *In proceedings of PDPTA'99*, Las Vegas, USA, June 28-July 1, 1999.

[12] Prylli, L., Tourancheau, B.: BIP: A New Protocol Designed for High Performance Networking on Myrinet. *In Parallel and Distributed Processing, IPPS/SPDP'98*. **1388** of *LNCS*, pages 472–485. Springer-Verlag.

[13] Rajasekaran, U. K. V., Chetlur, M., Sharma, G. D., Radhakrishnan, R., Wilsey, P. A.: Addressing Communication Latency Issues on Clusters for Fine Grained Asynchronous Applications - A Case Study. *In Proceedings of the Intl. Workshop on Personal Computer based Network of Workstations (PCNOW'99)*.