

Finding a Tradeoff between Host Interrupt Load and MPI Latency over Ethernet

Brice Goglin¹, Nathalie Furmento²

¹ INRIA, ² CNRS

LaBRI – 351 cours de la Libération, F-33405 TALENCE – FRANCE

¹ Brice.Goglin@inria.fr — ² Nathalie.Furmento@labri.fr

Abstract—Achieving high-performance message passing on top of generic ETHERNET hardware suffers from the NIC interrupt-driven model where coalescing is usually involved. We present an in-depth study of the impact of interrupt coalescing on the OPEN-MX performance. It shows that disabling coalescing may not be relevant for most metrics except small-message latency. Two new coalescing strategies are then presented so as to efficiently support both latency-friendly and coalescing-friendly workloads thanks to the NIC looking at OPEN-MX messages and streams before deciding when to raise interrupts.

The implementation of these strategies in the firmware of MYRI-10G NICs shows that OPEN-MX is now able to achieve a low small-message latency, a high large-message throughput, and a satisfying message rate without having to manually tune the coalescing delay depending on the benchmark. Real application performance evaluation further shows that our modifications even improve the NAS Parallel Benchmark IS execution time by 7-8 % thanks to our NIC firmware raising up to 20 % of additional interrupts at the correct time.

I. INTRODUCTION

High-performance networking in clusters relies on advanced hardware features that vendors implement in their NICs. For instance, zero-copy, i.e. the ability to read events and data from user-space on the sender side and deposit it directly in the target application buffers on the receiver side, enables high-throughput communication. It also enables application-directed polling of incoming events without any intervention of the operating system. These features require advanced hardware support that is presently not available in generic ETHERNET hardware.

OPEN-MX [1] is a message passing stack implemented on top of the ETHERNET software layer of the LINUX kernel. It provides high-performance communication over any generic ETHERNET hardware using the wire specifications and the application programming interface of *Myrinet Express* [2]. While being compatible with any legacy ETHERNET NIC, OPEN-MX performance suffers from the lack of the aforementioned hardware features.

One key-point of the OPEN-MX receive stack is that the existing NIC and driver model enforces a interrupt-driven model. The host processors are indeed notified of newly received packets only when interrupts are raised, which implies an large software overhead. The usual way to work around this problem is *Interrupt Coalescing* which coalesces several notifications within a single interrupt.

While working well for the throughput of TCP-like communication flows, interrupt coalescing causes small-message latency to increase significantly. OPEN-MX benchmarking may thus require proper tuning of the coalescing delay depending on the communication pattern. We propose in this article to study the actual impact of coalescing strategies on OPEN-MX performance, from the latency, throughput and message-rate points of view.

The rest of this paper is organized as follows. Section II provides background information about interrupts in receive stacks and OPEN-MX, and it details our motivations. Section III explains two approaches to improve interrupt notification by adding some knowledge of OPEN-MX messages and streams in the NIC coalescing heuristics. Experiments shown in Section IV emphasize the performance impact of the existing coalescing strategies and of our proposal on various micro-benchmarks and on application performance. We discuss related work and propose future research directions in Sections V and VI, respectively.

II. BACKGROUND AND MOTIVATIONS

We introduce in this section the existing techniques for dealing with interrupts in ETHERNET hardware, high-performance computing and the reason why this needs to be studied in the context of OPEN-MX.

A. Ethernet hardware and Interrupts

The design of ETHERNET hardware in the last decades was mostly driven by TCP/IP communication. These network interfaces usually serve many communication flows between multiple hosts. The user expectations in this context focuses on the throughput and the equity between flows. Several optimizations such as *Transmit Segmentation Offload* or *Large Receive Offload* have been proposed as ways to improve TCP performance on generic hardware. The design of the receive stack remains nonetheless very simple. The NIC just tries to deposit a stream of packets in the host memory as quickly as possible, and then notifies the host processor of their arrival. This notification relies on a interrupt-based model, which leads to the question of when to raise each interrupt.

Interrupts are logical signals that I/O devices may send to processors to force them to process some event. This interrupt suffers from a large hardware and software overhead (several microseconds) due to the need to switch from the

current processor execution context into a dedicated interrupt handling context. It is thus important to reduce the amount of interrupts so as to prevent interrupt processing from consuming all CPU time, and so as to improve the overall system availability. This so-called *Receive Livelock* problem [3] led to the design of new operating system receive stacks, such as LINUX NAPI [4], which try to find a tradeoff between early packet processing and interrupt load.

On the hardware side, the well-known *Interrupt Coalescing* optimization delays the interrupt up to when a certain amount of packets has been received or when a timeout expires. The host is thus able to process several packets at once instead of being interrupted multiple times. While reducing the host interrupt load, coalescing however increases communication latency since the host may not process a packet before the coalescing timeout expires. Reducing the coalescing delay is an obvious solution for improving latency but significantly increases the host load under high traffic. However, it has been shown that large coalescing delays improve performance regardless of the metrics [5]. Additionally, coalescing must be carefully tuned to improve TCP performance [6]. It is thus interesting to look at this problem in the context of high-performance computing where both latency and host load are important metrics.

B. The OPEN-MX Stack

The OPEN-MX stack aims at providing high-performance message passing over any generic ETHERNET hardware. It exposes the *Myrinet Express* API (MX [2]) to user-space applications. Many existing middleware projects such as MPICH2-MX [7] and OPEN MPI [8] run successfully unmodified on top of it. OPEN-MX is also interoperable with hosts running the native MX stack over ETHERNET (MXoE).

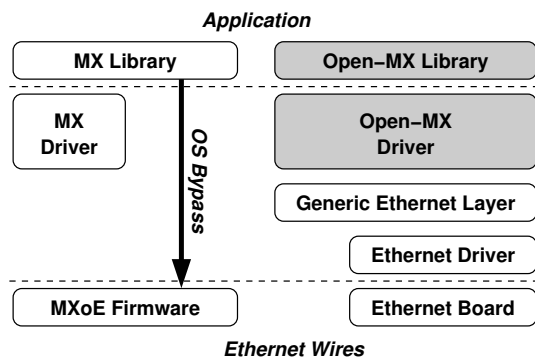


Fig. 1. Design of the native MX and generic OPEN-MX software stacks.

OPEN-MX was initially designed as an emulated MX firmware in a LINUX kernel module [1]. This way, legacy applications built for MX benefit from the same abilities without needing the MYRICOM hardware or the native MX software stack (see Figure 1). However, the features that are usually implemented in the hardware of high-speed networks are obviously prone to performance issues when emulated in software. Indeed, portability to any ETHERNET hardware

requires the use of a very simple common low-level programming interface to access drivers and NICs.

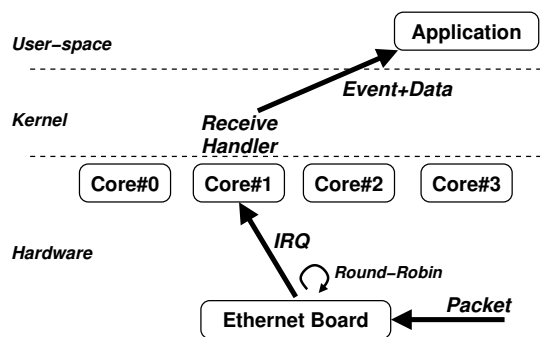


Fig. 2. Path from the NIC interrupt up to the application receiving the event and data.

As any implementation on top of the LINUX kernel ETHERNET software interface, the OPEN-MX receive stack is *Interrupt-driven*. Incoming packets are processed by the OPEN-MX-specific *Receive handler* when an interrupt is raised by the NIC (see Figure 2). Indeed, this handler is invoked when the *Bottom Half* of the ETHERNET driver processes an interrupt. The corresponding event and data are then passed to the user-space application through a shared memory ring.

The whole receive stack is thus prone to latency issues due to interrupt coalescing techniques since delaying an interrupt in the NIC postpones the corresponding packet processing in the host. It raises the question of finding a tradeoff between low latency and limited host interrupt overhead. Moreover, since interrupts are usually scattered across all processor cores by the hardware chipset in a round-robin manner, each interrupt may require OPEN-MX data structures to be evicted from other processor caches and fetched back into the local cache. Thus, the more interrupts, the more cache-line bounces between cores.

C. Motivation and Objectives

Achieving high-performance with OPEN-MX requires an efficient delivery of incoming packets up to user-space applications. Relevant metrics in high-performance computing vary from raw latency and message rate (as measured by usual benchmarks), up to the overall host interrupt load (involved in the overall application performance). High-speed networks such as INFINIBAND [9] or MYRI-10G [10] rely on application-directed polling and the ability of the NIC to deposit packets directly in the application buffers. These features are not available in generic ETHERNET hardware and drivers. OPEN-MX has to rely on the existing interrupt-driven model. It is thus important to adjust the usual ETHERNET NIC interrupt behavior in order to satisfy OPEN-MX latency and host interrupt load requirements.

The benchmarking guidelines provided with OPEN-MX recommend to disable interrupt coalescing when latency is the critical metric. The first objective of this paper is to

study the actual impact of tuning interrupt coalescing on other metrics such as message rate as well as real application performance. The second objective is to **design a new interrupt coalescing strategy that suits OPEN-MX requirements**. The idea is to automatically support both low latency communication for small messages, and high throughput and low host overhead for large communication patterns and large messages.

The notion of *Message* makes MPI communication patterns differ significantly from usual TCP flows. For instance, it makes sense to preferably raise interrupts at the end of messages instead of when a static timeout expires. We thus propose to add *Markers* in OPEN-MX packets to help the NIC decide when an interrupt should be raised. We then want to integrate this idea within a *Stream-aware* coalescing model that tries to avoid early interrupts if another OPEN-MX packet is expected in the near future.

These ideas have to be implemented inside the OPEN-MX wire protocol and in a ETHERNET NIC firmware since the existing hardware does not offer such abilities. Indeed, *Interrupt Coalescing* has been designed for TCP communication patterns. The TCP/IP protocol does not provide the NIC with any way to find out the internal structure of a communication flow (the actual structure is only known by the application). For this reason, existing ETHERNET hardware and operating systems such as LINUX only provide very basic ways to tune interrupt coalescing: setting the maximal coalescing delay, or the maximal amount of packets that may be received before raising an interrupt. This model suits TCP communication since periodic and coalesced interrupts enable high-throughput for continuous and regular flows. However, applying this model to structured communication flows such as OPEN-MX raises the question of how to adapt it to message passing. Indeed, different parallel application phases may mix few or lots of small or large messages, and thus mix cases where interrupt coalescing requirements differ significantly.

III. DESIGN OF A OPEN-MX-AWARE INTERRUPT COALESCING STRATEGIES

We describe in this section the design of new interrupt coalescing strategies which try to automatically support low latency for small messages as well as high throughput and low host overhead for large messages and large communication patterns.

A. Overview of OPEN-MX communication patterns

Once OPEN-MX establishes a connection between two peers, each MPI message is transferred using one of the existing formats in the wire specification of MXoE (*Myrinet Express over Ethernet*): Up to 128 bytes (*Small* messages), a single packet is sent eagerly. Up to 32 kiB (*Medium* messages), a stream of several fragments is sent eagerly. The size of these fragments depends on the fabric *Maximum Transmission Unit* (MTU). For these messages, an interrupt would obviously be preferred when the last fragment arrives since the application would then be able to process the entire message at once.

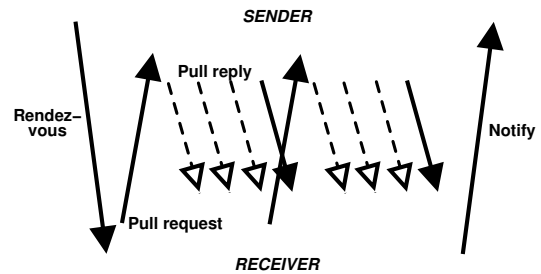


Fig. 3. Timeline of packet exchange during a large *Pull* OPEN-MX message. Plain lines with black arrows are packets that OPEN-MX marks as needing an immediate interrupt.

Large messages (more than 32 kiB) are much more complex: The sender first sends an explicit *Rendezvous* packet. When a matching receive request has been posted, the receiver initiates a *Pull* to retrieve data from the sender. This strategy is similar to a RDMA *Get* operation. It is performed by requesting up to 32 fragments at once, and then requesting the next 32 fragments while the previous ones are being received (see Figure 3). Once the transfer is done, an explicit *Notify* message is sent back to the sender to complete the operation. This complex protocol is sensitive to latency since multiple round-trips are involved, both for the *Rendezvous/Notify* and for requesting data fragments. It is thus important to mark the following packets as requiring special care from the NIC since processing them early may improve the overall performance: *Rendezvous*, *Pull* request, last fragment of a *Pull* reply, *Notify*.

This shows how message passing over OPEN-MX exhibits structured communication flows where some packets have to be privileged by the NIC so as to both reduce small-packet latency and improve the throughput of large patterns. TCP streams cannot be treated as cleverly since there is no way to determine any structural information such as *Latency-sensitive* packets.

B. Latency-Sensitive Packets

Having identified how the OPEN-MX message structure appears within packets and may be used by the NIC to decide when to raise interrupts, we can now present the implementation of our new coalescing techniques. The idea behind OPEN-MX-aware interrupt coalescing is to detect *Latency-sensitive* packets and preferably raise interrupts as soon as they have been transferred into host memory. We implemented this model in OPEN-MX by marking packets as latency-sensitive in the sender driver (where the actual splitting of user-space messages into fragments is performed). Packets are marked by adding a special flag in the OPEN-MX header.

On the receive side, the NIC must be able to recognize this marker before applying its interrupt coalescing heuristics. We modified the `myri10ge` ETHERNET firmware of MYRI-10G NICs to do so. When a new packet is processed, the firmware checks the OPEN-MX-specific *Latency-Sensitive* marker flag. The interrupt cannot however be raised immediately since the packet must be deposited in the host memory (by DMA) before the interrupted processor may actually process it. The

Algorithm 1 OPEN-MX Coalescing: Interrupt coalescing based on marking *Latency-Sensitive* packets.

```

if a packet arrives then
  Create packet Descriptor
  if Packet is Marked then
    Mark packet Descriptor
  end if
  Submit DMA to host memory
end if
...
if a DMA completes then
  Find corresponding packet Descriptor
  if Descriptor is Marked then
    Raise Interrupt
  end if
end if
...
if Interrupt coalescing timeout expires then
  Raise Interrupt
  Reset coalescing timeout
end if

```

firmware thus marks the packet descriptor and checks it when its DMA completes (see Algorithm 1). If the descriptor was marked as latency-sensitive, an interrupt is raised immediately instead of waiting for the usual coalescing timeout to expire.

This implementation will be referred to as “OPEN-MX Coalescing”. Thanks to it, the NIC now raises an interrupt when receiving a small packet, the last fragment of a medium message, the last fragment of a pull request during a large message, or a performance-sensitive control packet (for instance a *Rendezvous*). Any packet not matching one of the above criteria is treated normally, with a interrupt coalescing timeout, which means that IP connections and OPEN-MX management packets are unaffected by our firmware modifications.

This model enables the coalescing of all fragments within a single medium message, or all fragments replying to a single *Pull* request. It brings interesting cache properties since all these fragments will be processed at once by the same processor which received the interrupt. Indeed, these fragments refer to the same OPEN-MX descriptors in the driver (communication channel, pull descriptor, ...). Processing all these fragments consecutively on the same core is thus expected to increase cache hits. On the contrary, usual interrupt coalescing splits the packet flow randomly and thus scatters across different cores the processing of related packets (interrupts are usually raised to processors in a round-robin manner), causing cache-line bounces between cores.

C. Stream-aware Model

We described in the previous section a OPEN-MX coalescing mechanism that raises interrupts when needed, especially at the end of a stream of related fragments. We expect this model to achieve better small-message latency and large-message throughput. But it also limits the message rate since

there is basically one interrupt per small message. We now look at how to improve message rate by reducing when possible the need to raise an interrupt for each small packet.

Supporting small-message streams in a clever way requires to carefully detect such streams before actually raising interrupts. This idea looks like predicting the future of the incoming traffic but it is actually not that hard. Indeed, as explained in the previous section, the interrupt is not raised immediately since the packet first has to be deposited in host memory by DMA. We thus implemented a *Stream-aware* variant of our OPEN-MX coalescing strategy by looking at the future incoming traffic during the DMA processing time. If no other packet arrives before the DMA completes, the requested interrupt is actually raised. If some packets arrived, the interrupt is *Deferred* so as to wait for the corresponding DMAs to also complete (see Algorithm 2).

Algorithm 2 *Stream* Coalescing: Interrupt coalescing, with deferring of interrupt in case of a stream of packets.

```

if a packet arrives then
  Create packet Descriptor
  if Packet is Marked then
    Mark packet Descriptor
  end if
  Submit DMA to host memory
end if
...
if a DMA completes then
  Find corresponding packet Descriptor
  if no other DMA is pending then
    if Descriptor is Marked or DeferredInterrupt is set then
      Raise Interrupt
      Clear DeferredInterrupt
    end if
  else if Descriptor is Marked then
    Set DeferredInterrupt
  end if
end if
...
if Interrupt coalescing timeout expires then
  Raise Interrupt
  Clear DeferredInterrupt
  Reset coalescing timeout
end if

```

This implementation will be referred to as “*Stream Coalescing*”. This way, in case of a stream of small packets, the interrupt is deferred to after the last packet so that all packets are processed at once by the host processor. However, if a single small packet is received, the interrupt is still raised early and the host processes the packet as soon as possible. Moreover, the initial interrupt coalescing delay is still available (for non-OPEN-MX packets or non-marked packets). In case of very long streams, the interrupt may at most be deferred until the coalescing timeout expires.

This model also has the advantage of helping support for packet disorder. Indeed, since only the last fragment of medium messages or pull replies is marked, a mis-ordered fragment may cause the interrupt to be raised before all fragments are actually received. If the missing fragments arrive immediately after the marked one, our new *Stream* coalescing model will detect them and defer the interrupt a bit. All fragments will thus still be reported at once to the host. If the missing fragments are significantly late, the interrupt may be raised before they arrive. But the message transfer time would have been disappointing anyway because of these mis-ordered and delayed packets.

IV. PERFORMANCE EVALUATION

We now look at the impact of interrupt coalescing strategies on OPEN-MX performance. We first study the existing coalescing techniques (timeout based) and try to understand how their possible configurations impact on the OPEN-MX latency, throughput and message rate. Then, we look at the new techniques that we designed specifically for OPEN-MX and explain how they improve performance on microbenchmarks and real applications.

A. Experimentation Platform

We implemented the OPEN-MX and *Stream* coalescing techniques that were presented in Sections III-B and III-C in the `myri10ge` firmware 1.4.41 of MYRICOM MYRI-10G ETHERNET interfaces [10]. Modifying the firmware to support our OPEN-MX coalescing model required the addition of less than 20 lines of code (in the main incoming packet processing routine and in the write DMA completion routine). The *Stream* coalescing patch added about 20 other lines of code.

The experimentation platform is composed of two dual-socket INTEL quad-core XEON processors (*Clovertown* E5345 2.33 GHz) running OPEN MPI 1.3.0 over OPEN-MX 1.0.901, on top of LINUX kernel 2.6.26 and `myri10ge` driver 1.4.4. The ETHERNET fabric and OPEN-MX have been configured with a *Maximum Transmission Unit* (MTU) of 1500 bytes so that interrupt-related effects appear more significantly. Indeed, the smaller the packets, the higher the interrupt and processing costs per byte. However, it has to be noted that a larger MTU (9000-bytes jumboframes) would exhibit the same behavior for small messages (where the MTU does not matter) and for proportionally-larger messages.

B. Impact of Interrupt Coalescing

We now detail the actual impact of the existing interrupt coalescing delay on the OPEN-MX performance.

1) *Message Rate*: Figure 4 presents the maximal rate of a unidirectional stream of 128 bytes messages (i.e. *Small* messages) between two OPEN-MX processes, depending on interrupt coalescing, interrupt binding, and on the core sleeping state. The default configuration of modern machines is to scatter interrupts across all cores in a round-robin manner,

to use a high interrupt coalescing delay ($75 \mu\text{s}$ on MYRI-10G NICs) and to let cores go to sleep when idle¹. This configuration is able to receive up to 433k messages per second, disabling interrupt coalescing ($0 \mu\text{s}$ delay) reduces the rate by more than a factor of two.

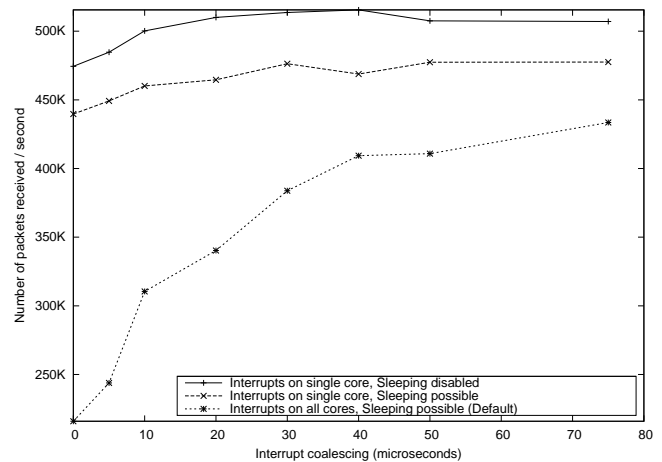


Fig. 4. Message rate of a stream of 128 bytes OPEN-MX messages.

Then, letting cores go to sleep when idle appears to have a huge overhead since disabling sleeping significantly improves the message rate. It means that if the target core is sleeping when an interrupt is raised (for instance because the MPI process running on this core is waiting for an I/O to complete), several microseconds may be needed before the interrupt is actually processed.

Finally, processing interrupts on different cores causes cache-line bounces between cores and thus increases the overhead, as explained at the end of Section III-B. However, for load balancing purpose, interrupt processing should remain distributed across cores, hence increasing the overhead.

Both these arguments justify the idea of coalescing interrupts. Indeed, the less interrupts, the less cores have to be woken up, and the more packets are processed at once by each core, reducing the number of cache-line bounces per packet.

2) *Interrupt Overhead*: To further study the impact of interrupt coalescing and binding, we measured the per-packet overhead of interrupt processing. The stream is now composed of a million of explicitly invalid 128 bytes packets. All of them are dropped immediately by the OPEN-MX receive handler so that only the low-level receive stack is involved.

The observed overhead is 965 ns per packet when always raising an interrupt (coalescing disabled) while it drops by roughly 20% when coalescing is enabled, down to 774 ns. It shows that coalescing indeed reduces the host interrupt overhead.

Moreover, by binding interrupts on a single core, the overhead drops slightly, by roughly 40 ns. We assume it is related to a cache miss being removed in the low-level receive

¹Modern INTEL XEON processors quickly enter the C1E sleep state when idle.

stack thanks to all packets being processed on the same core. We expect that a lot more cache misses would have been observed if the OPEN-MX receive handler had processed these packets instead of dropping them. It confirms that raising many interrupts causes cache-related problems since interrupts are usually scattered across multiple cores.

3) *Ping-pong*: Figure 5 presents the throughput of an OPEN-MX ping-pong benchmark. As expected, it shows that interrupt coalescing significantly disturbs small-message latency since it increases from about $10 \mu\text{s}$ up to $75 \mu\text{s}$. Indeed, each ping-pong iteration uses a single packet and the host has to wait for the whole coalescing delay to expire ($75 \mu\text{s}$ on MYRI-10G NIC by default). However, large-message throughput increase when interrupt coalescing is enabled since many packets are transmitted and thus causing the lower per-packet interrupt overhead to become important.

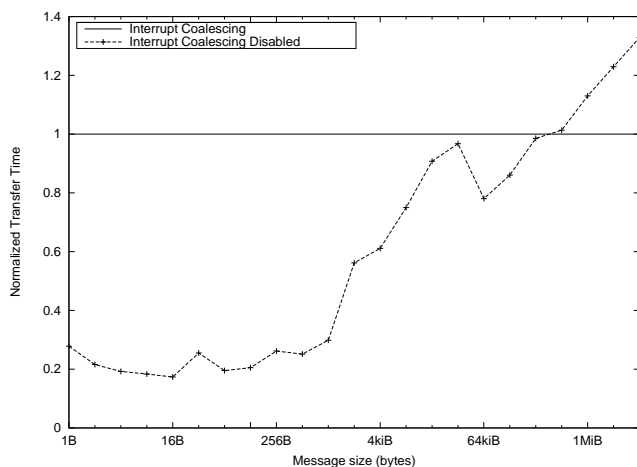


Fig. 5. Relative transfer time of a ping-pong depending on the interrupt coalescing.

4) *Summary*: Results presented in this section confirm that disabling interrupt coalescing should only be recommended when small-message latency is critical. Other metrics such as message rate and large-message throughput benefit significantly from coalescing. Indeed, coalescing reduces the host interrupt load and improves locality during interrupt processing since more packets are processed at once on the same core, thus avoiding cache-line bounces. Moreover, many interrupts may cause sleeping cores to wakeup more often, thus increasing the host overhead even more.

C. OPEN-MX and Stream Coalescing

In the previous section, we described the impact of the interrupt coalescing delay on the OPEN-MX performance. We now look at the new OPEN-MX and *Stream* coalescing strategies that we proposed in Sections III-B and III-C.

1) *Ping-pong*: Figure 6 presents the normalized message transfer time during a OPEN-MX ping-pong between our experimentation hosts. It shows that our OPEN-MX coalescing

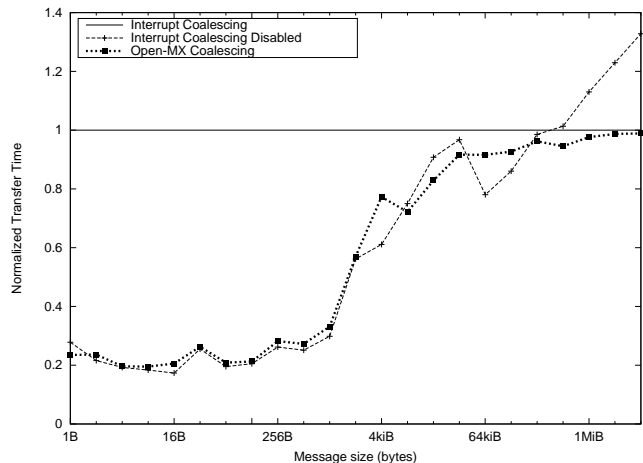


Fig. 6. Relative transfer time of a ping-pong depending on the interrupt coalescing.

modifications in the `myri10ge` firmware enable both a low small-message latency (as if interrupt coalescing was enabled) and a high large-message throughput (as if interrupt coalescing was using a large delay). As explained in Section III-B, marking small-message packets as *Latency-Sensitive* enables low latency without requiring interrupt coalescing to be disabled. And marking only the last fragment of medium messages or the last fragment of a stream of pull reply packets prevents the host load from increasing (if there were too many interrupts), while it also enforces one interrupt to be raised on time (at the end of the message).

The performance of the *Stream* coalescing modification is not presented here since this optimization is not involved during a ping-pong. It basically brings the same performance improvement as the first OPEN-MX coalescing modification.

2) *Message Rate*: Table I presents the message rate between 2 OPEN-MX hosts measured on the receiver side (where interrupts matter). It first confirms again that disabling interrupt coalescing dramatically reduces message rate, not only for small messages (by a factor of 2) but also for large messages (by 26%).

TABLE I
MESSAGE RATE DEPENDING ON THE MESSAGE SIZE AND COALESCING TECHNIQUE.

Message Size	Interrupt Coalescing Strategy			
	Default	Disabled	OPEN-MX	<i>Stream</i>
0 B	490k	252k	423k	435k
32 kiB	14507	6476	14533	14691
1 MiB	452	334	451	447

The OPEN-MX coalescing firmware brings significant improvements over disabling coalescing, even for small messages. The reason is that some other packets are actually transferred (for instance some ACKs, up to 20% of the traffic) but these packets are not marked as *Latency-Sensitive*. So even if many small packets need an immediate interrupt, the overall

number of interrupts is reduced and the message rate thus increases.

As expected, the *Stream* coalescing firmware improves the message rate for small messages. We observed that it actually reduces the number of interrupts by about a factor of 2. It does not however bring a significant benefit for medium and large messages since these messages are already optimized cleverly in our first firmware modification: only the last packet of medium messages is marked, and only the last packets of each 32-packet pull reply are marked. Moreover, the *Stream* coalescing modification requires more work in the NIC and may thus limit performance under high traffic.

In the end, OPEN-MX is now able to approach the message rate of the usual coalescing timeout, especially for non-small messages.

3) *Anatomy of Interrupts during Large Messages:* We now take a deeper look at the behavior of our OPEN-MX coalescing by studying the impact of each marked packet during a large message transfer. Table II presents the transfer time of a large message and the corresponding amount of raised interrupts (on both sides). The 234 kiB size is intermediate between small messages (where only latency matters) and large messages (where throughput and streams matter). As explained in Section III-A, this message requires 5 pull requests, and each request gets 32 pull reply packets in return. A total of 162 packets are thus exchanged.

TABLE II
IMPACT OF INTERRUPT COALESCING STRATEGIES ON THE TRANSFER TIME OF A 234 kiB MESSAGE.

Coalescing Strategy	Transfer Time	Interrupts
Disabled	705 μ s	\approx 92.4
Timeout 75 μ s	762 μ s	\approx 14.4
OPEN-MX	708 μ s	\approx 13.7

Such a basic communication pattern does not suffer much from interrupt overhead since no process is actually using any single core. Thus, the best coalescing delay configuration for this micro-benchmark is to disable it, even if generates more than 6 times more interrupts per message (92 instead of 14).

Our modified firmware is able to achieve almost the same performance (708 μ s instead of 705 μ s) while keeping a very small amount of interrupts. We even need a bit less interrupts than the usual coalescing strategy. We assume that our modification raises interrupts exactly on time, while the usual coalescing may sometime miss packets and thus need another additional interrupt later.

Looking at which marked OPEN-MX packet actually helps reducing the overall message transfer time, we observe that marking the initial *Rendezvous* packet is critical since it decreases the time by 20 μ s. Marking the *Notify* packet does not however appear critical. This surprising result may be caused by some timing coincidence in this specific micro-benchmark. Indeed, nothing in the wire protocol justifies why this packet could arrive late without disturbing the overall performance. Marking pull requests and the last pull replies

respectively decreases the transfer time by 5 and 2 μ s. They appear less critical than the *Rendezvous* message because the driver tries to pipeline 4 requests at the same time. So even if one of them completes later because of interrupt coalescing, the wire may still be used by the already started next requests.

4) *Packet Mis-ordering:* We now study the impact of the *Stream* coalescing firmware modification on mis-ordered packets by looking at the transfer time of a 32 kiB medium message (23 packets). We simulated packet mis-ordering by moving the packet mark from the last fragment to an earlier one. A mis-ordering degree X thus means that packet $N - X$ was marked instead of N . Table III shows that the *Stream* coalescing firmware indeeds help performance in the mis-ordered case since the overall transfer time is reduced.

TABLE III
IMPACT OF PACKET MIS-ORDERING ON THE TRANSFER TIME OF 32 kiB MEDIUM MESSAGES.

Transfer Time	Correct Order	Mis-Ordering Degree	
		1	3
OPEN-MX Coalescing	156 μ s	177 μ s	177 μ s
<i>Stream</i> Coalescing	156 μ s	171 μ s	174 μ s

However, the success rate of the optimization is limited to 30 % when a single packet is mis-ordered, and drops to 15 % when 3 packets are mis-ordered. Indeed, the firmware appears to be able to defer interrupts by 3 or 4 packets but it fails to do so as soon as a small delay between packets appears. One way to improve this result would be to look deeper in the future traffic before deciding whether the interrupt should be raised. However, there is no easy way to do so in the *myriloge* firmware since only the processing time of a DMA is available as such a timeout.

This results significantly limits the scope of our *Stream* optimization since delays between packets often occur in loaded fabrics.

5) *Summary:* Results presented in this section first show that our OPEN-MX-specific coalescing strategy is able to achieve a low small-message latency (as if coalescing was disabled) and a high large-message throughput (as if coalescing was enabled). Moreover, our *Stream*-aware modification reduces the impact on message rate. This modification however cannot efficiently deal with a high packet disorder.

In the end, OPEN-MX is now able to achieve satisfying performance for different metrics without having to tune interrupt coalescing manually before running the benchmark.

D. NAS Parallel Benchmarks

We now look at real application performance by comparing the execution of NAS Parallel Benchmarks [11] depending on the interrupt coalescing strategy. 8 processes were used per node (one per core).

Table IV first shows that disabling interrupt coalescing never helps performance. It actually increases the execution time of the benchmarks whose network traffic is the highest (IS, FT and CG), up to 11.6 % for IS class C. This result confirms

TABLE IV
EXECUTION TIME (IN SECONDS) OF THE NAS PARALLEL BENCHMARKS WITH 16 PROCESSES ON 2 NODES, DEPENDING ON THE INTERRUPT COALESCING STRATEGY. ONLY NON-NEGLIGIBLE SPEEDUP PERCENTAGES ARE SHOWN. THE MOST SIGNIFICANT ONES ARE BOLD.

NAS	Coal.	Disabled	OPEN-MX	<i>Stream</i>
bt.C.16	271.2	272.8	273.3	272.6
cg.C.16	90.04	91.50 (-1.6 %)	90.76	90.70
ep.C.16	31.30	31.45	31.49	31.36
ft.C.16		Not enough memory		
ft.B.16	24.24	24.86 (-2.5 %)	24.21	24.20
is.C.16	32.75	37.03 (-11.6 %)	30.51 (+7.3 %)	31.96 (+2.5 %)
is.B.16	21.98	22.97 (-4.4 %)	20.32 (+8.2 %)	21.76 (+1.1 %)
lu.C.16	203.8	202.4	203.2	206.6 (-1.4 %)
mg.C.16	43.91	43.63	43.75	43.72
sp.C.16	549.1	551.1	546.8	546.1

that disabling coalescing only helps small-message latency and should not be recommended for other workloads. Table V emphasizes this result by showing that disabling interrupt coalescing leads to 22 times more interrupts and thus much more host overhead.

Our OPEN-MX coalescing scheme shows comparable performance to the regular interrupt coalescing for most tests, except the large-message intensive IS where the execution time is even reduced by 7-8 %. It confirms that our proposed strategy has the advantages of the usual coalescing while improving large-message throughput thanks to interrupts being raised on time. However, our *Stream* coalescing optimization shows disappointing results since the IS performance gain mostly disappears. We feel that this is caused by the *Stream* detection not being as efficient as expected as shown in Section IV-C4. The overhead of this optimization in the firmware may thus reduce the NIC throughput more than it actually helps the overall performance.

TABLE V
TOTAL NUMBER OF INTERRUPTS GENERATED DURING THE EXECUTION OF THE NAS PARALLEL BENCHMARKS WITH 16 PROCESSES ON 2 NODES, DEPENDING ON THE INTERRUPT COALESCING STRATEGY.

NAS	Coal.	Disabled	OPEN-MX	<i>Stream</i>
is.C.16	86.4k	1.93M ($\times 22$)	100.5k (+16 %)	101.6k (+17 %)
is.B.16	22.4k	496k ($\times 22$)	26.7k (+19 %)	27.2k (+21 %)

Table V also shows that our firmware optimizations add 15-20 % interrupts on the IS benchmark while reducing the execution time by 7-8 %. It confirms that raising a bit more interrupts when really needed actually enhances performance by waking up the processing host on time.

V. RELATED WORK AND DISCUSSION

Modifying network interfaces to help MPI performance has been the subject of many research projects. High-speed networks such as INFINIBAND [9], MYRI-10G [10], and the upcoming QsNET III [12] are well-known examples of hardware that were designed specifically for high-performance computing by adding support for many dedicated features in the NIC. ETHERNET hardware usually offers less features but

its programmability still often enables interesting HPC development by adding message passing abilities in the firmware of advanced NICs [13]. More recently, iWARP introduced some standardized RDMA abilities in several advanced ETHERNET NICs [14]. These ideas remain however expensive and require intrusive software support in the operating system.

OPEN-MX tries however to render high-performance MPI available to commodity hardware by exposing the popular interface of *Myrinet Express* and by not enforcing specific hardware support in the NICs. In this work, we proposed the addition of very simple features in NICs to significantly improve MPI performance over ETHERNET. In contrary to advanced features such as RDMA which requires important resources and abilities in the NIC, our few dozens lines of code should be easy to add to most ETHERNET firmwares. We feel that proposing such *Stateless* features is an easy way to get better support for high-performance computing in commodity hardware.

Interrupt coalescing has been widely studied in the last decade since the *Receive Livelock* problem appeared with the increasing network traffic. New ideas for designing operating system ETHERNET receive stacks have been proposed [3]. In the LINUX kernel, the *New Driver API* (NAPI [4]) now takes care of the tradeoff between efficient packet processing and host interrupt load by allocating a polling budget to each driver instance. On the hardware side, most modern NICs support interrupt coalescing, now offering efficient TCP/IP receive stacks. However, these ideas do not apply to message passing where the communication flow has much more structure than a basic TCP stream. And specialized high-speed networks did not solve the problem since their NICs directly deposit packets in the target application and thus do not need interrupts to notify the host of packet arrival. Our work tries to enhance the existing coalescing techniques by adding some knowledge of message passing protocols inside the NIC firmware.

VI. CONCLUSIONS AND FUTURE WORK

The long-awaited convergence between specialized high-speed networks for clusters and usual ETHERNET technologies raises the question of how to translate software innovations from the former onto the latter. Existing ETHERNET hardware offer limited features and thus make message passing performance hard to achieve. Fast notification of packet arrival is one of the areas where high-speed networks brought software innovations such as depositing packets in the application memory. However their adoption in the existing ETHERNET model is dramatically constrained by the interrupt-driven model. Indeed, raising interrupt is useful for improving latency while coalescing interrupts is critical to reduce the host load.

We proposed in this article an in-depth study of the existing interrupt coalescing strategies in the context of high-performance message passing with OPEN-MX. Coalescing has been designed to help TCP/IP throughput but it is also known to disturb small-message latency. We showed that disabling coalescing indeed improves the OPEN-MX latency and may thus be relevant for benchmarking purpose. However

other metrics such as throughput or message rate and real application obtain higher performance thanks to coalescing. Indeed, coalescing reduces the host interrupt load and also reduces the risk of having to wakeup a sleeping idle core.

We then designed an OPEN-MX-specific coalescing strategies that relies on having the sender mark *Latency-Sensitive* packets such as small messages or the last fragment of a stream. We also proposed a *Stream* coalescing strategy that tackles message rate performance by coalescing interrupts in case of consecutive latency-sensitive packets. These ideas have been implemented in the `myri10ge` firmware of MYRI-10G NICs. Performance evaluation first shows that a ping-pong benchmark may now achieve a low small-message latency (as if coalescing was disabled) and high large-message throughput (as if coalescing was enabled). Meanwhile, message rate remains satisfying, and the NAS Parallel Benchmark performance does not decrease and even increases by up to 8% for the communication intensive IS benchmark.

We showed that the improvement is related to the NIC raising, at the right time depending on incoming packets, only up to 20% additional interrupts. This solution is a good tradeoff between basic timeout-based coalescing techniques (where few interrupts are raised at almost random times), and disabling coalescing (where up to 22 times more interrupts overload the host processors). Moreover, non-OPEN-MX traffic (such as TCP/IP) is not disturbed by our modification since the new coalescing techniques only look at marked packets.

The *Stream* coalescing modification unfortunately shows its limitations as soon as the traffic becomes irregular. It cannot always handle a high-degree of packet mis-ordering due to the difficulty of looking at the upcoming traffic long in advance. Moreover, it slightly decreases NAS performance due to more work in the NIC. Since message rate is usually not a critical metric for small clusters targeted by OPEN-MX, we expect that our first OPEN-MX coalescing mechanism will be more relevant.

We have also discussed the problem of interrupt affinities for some caches. Indeed, processing incoming packets involves many accesses to some shared communication channel descriptors in host memory. It may cause cache-line bounces when many interrupts are raised to multiple cores. We are thus looking at adding OPEN-MX-aware *Multiqueue* support [15] to solve this issue by attaching each communication channel processing to a single core. This is another example of *Stateless* hardware support that requires very few resources and could be added in most existing NIC firmwares. We feel it is a good way to show hardware vendors that helping message passing performance is easy and does not require complex support such as RDMA or TOE in the NIC.

We are also looking at *Adaptive Coalescing* which changes the interrupt coalescing delay dynamically depending on the traffic. Indeed, a small traffic may be treated with one interrupt per packet (thus improving latency), while a large throughput really requires a large coalescing delay. This feature is however available in very few ETHERNET drivers so far. Our early tests with an experimental `myri10ge` adaptive coalescing support

show that it helps microbenchmarks but cannot help real applications as well as our firmware modifications do. We assume this is related to the fact that tuning coalescing depending on the past traffic only works for regular communication patterns. Therefore, we are studying the idea of combining adaptive coalescing with our firmware modifications that exploit the knowledge of the message structure in the communication flow.

ACKNOWLEDGMENTS

We would like to thank Hyong-Youb Kim, Andrew J. Gallatin, and Loïc Prylli from Myricom, Inc. for helping us when modifying the `myri10ge` firmware of MYRI-10G boards.

REFERENCES

- [1] B. Goglin, "Design and Implementation of Open-MX: High-Performance Message Passing over generic Ethernet hardware," in *CAC 2008: Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2008*. Miami, FL: IEEE Computer Society Press, Apr. 2008. [Online]. Available: <http://hal.inria.fr/inria-00210704>
- [2] *Myrinet Express (MX): A High Performance, Low-Level, Message-Passing Interface for Myrinet*, Myricom, Inc, 2006, <http://www.myri.com/scs/MX/doc/mx.pdf>.
- [3] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, pp. 217–252, 1997.
- [4] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond softnet," in *Proceedings of the 5th annual Linux Showcase & Conference*, Oakland, CA, Nov. 2001.
- [5] K. Salah, "To Coalesce or Not To Coalesce," *International Journal of Electronics and Communications*, vol. 61, pp. 215–225, 2007.
- [6] M. Zec, M. Mikuc, and M. agar, "Estimating the impact of interrupt coalescing delays on steady state tcp throughput," in *Proceedings of the 10th SoftCOM Conference*, Split, Croatia, Oct. 2002.
- [7] "MPICH-MX and MPICH2-MX Software," <http://myri.com/scs/download-mpichmx.html>.
- [8] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, Sep. 2004, pp. 97–104.
- [9] "Infiniband architecture specifications," InfiniBand Trade Association, 2001, <http://www.infinibandta.org>.
- [10] "Myricom Myri-10G," <http://myri.com/Myri-10G/>.
- [11] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks," *The International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, Fall 1991.
- [12] J. Beecroft, D. Hewson, F. Homewood, and D. R. and Ed Turner, "The Elan5 Network Processor," International Supercomputing Conference (ISC'07), Dresden, Germany, Jun. 2007.
- [13] P. Shivam, P. Wyckoff, and D. K. Panda, "EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing," in *Proceedings of Supercomputing ACM/IEEE 2001 Conference*, Denver, CO, Nov. 2001, p. 57.
- [14] M. J. Rashti and A. Afsahi, "10-Gigabit iWARP Ethernet: Comparative Performance Analysis with Infiniband and Myrinet-10G," in *Proceedings of the International Workshop on Communication Architecture for Clusters (CAC), held in conjunction with IPDPS'07*, Long Beach, CA, Mar. 2007, p. 234.
- [15] Z. Yi and P. P. Waskiewicz, "Enabling Linux Network Support of Hardware Multiqueue Devices," in *Proceedings of the Linux Symposium (OLS2007)*, Ottawa, Canada, Jun. 2007, pp. 305–310.