High Throughput Intra-Node MPI Communication with Open-MX

Brice Goglin

High Throughput Intra-Node MPI Communication with Open-MX

Brice Goglin
INRIA Bordeaux – LaBRI – France
Brice.Goglin@inria.fr

***Abstract*—The increasing number of cores per node in high-performance computing requires an efficient intra-node MPI communication subsystem. Most existing MPI implementations rely on two copies across a shared memory-mapped file.**

Open-MX offers a single-copy mechanism that is tightly integrated in its regular communication stack, making it transparently available to the MX backend of many MPI layers. We describe this implementation and its offloaded copy backend using I/OAT hardware. Memory pinning requirements are then discussed, and overlapped pinning is introduced to enable the start of Open-MX intra-node data transfer earlier.

Performance evaluation shows that this local communication stack performs better than MPICH2 and Open MPI for large messages, reaching up to 70 % better throughput in micro-benchmarks when using I/OAT copy offload. Thanks to a single-copy being involved, the Open-MX intra-node communication throughput also does not heavily depend on cache sharing between processing cores, making these performance improvements easier to observe in real applications.

I. Introduction

The emergence of multicore architectures increases the need for high-performance local inter-process communication. While hybrid OpenMP-MPI models have been proposed as a solution to programming clusters of shared-memory nodes [3], most existing applications still use MPI for both intra- and inter-node communication since it is considered easier to program with. Most MPI implementations thus provide a dedicated local communication backend that is transparently used whenever local inter-process communication is involved.

Local MPI communication have been the topic of many research works in the past since multiprocessor architectures have been widely used in high-performance computing in the last decade. It led to the design of carefully optimized shared memory implementations to achieve very low latency, such as the Nemesis subsystem in MPICH2 [2]. This shared-memory model that many other MPI layers also use (such as Open MPI [5] or MVAPICH [4]) is based on the sender copying data in a shared file and the receiver copying back from it. While being efficient for small message latency, this model raises the problems of memory bandwidth saturation and cache pollution for large messages since multiple memory copies are involved. To overcome these issues, one solution consists in entering the operating system to perform a single-copy for large messages. This idea has been studied in multiple software stacks such as MVAPICH [14] but it raises the problems of pinning memory and integrating a kernel subsystem within a user-space middleware.

This research is supported by a collaboration between INRIA and Myricom, Inc.

We present in this paper the implementation and optimization of this model in the Open-MX communication stack. Open-MX [6] enables existing applications that were programmed for Myricom's widely used *Myrinet Express* (MX) interface [11] to work on any regular Ethernet hardware. Open-MX may also transparently rely on the local loopback interface, providing a tight integration of both intra-node and inter-node communication to any legacy MPI or MX application. We introduce in this stack a dedicated single-copy based implementation bypassing the local interface and we describe its port on Intel I/OAT hardware which enables memory copy offload. This model achieves high-throughput local communication without relying on shared caches between processing cores. We then discuss memory pinning requirements in our model and propose the overlapping of memory pinning in order to start data transfer earlier and thus reduce the overall communication time.

The remaining of this paper is organized as follows. We present the existing intra-node communication models and Open-MX as well as our motivations in Section II before describing the Open-MX optimized intra-node communication subsystem in Section III. Section IV presents a performance evaluation of Open-MX local communications and optimizations as well as comparison with two famous shared-memory MPI implementations. Before concluding, related works are discussed in Section V.

II. Background and Motivations

In this section, we first describe the traditional shared-memory-based intra-node MPI communication model. Then we introduce Open-MX and its local communication model, before discussing the drawbacks of these solutions and presenting motivations for this work.

A. Intra-Node MPI Communication

Running parallel applications on multiprocessor nodes (and nowadays multicore processors) involves many local communications between processes. Their performance is as critical to the whole application performance as network communication matters to applications running on multiple nodes.

Many MPI implementations offer optimized intra-node communication. Most of them, including MPICH2,

OPEN MPI and MVAPICH, are implemented as shared-memory. This model is based on a memory-mapped shared file where the sender writes commands and copies data while the receiver reads commands and copies data back from. This idea is easy to implement, it does not require any system call in the critical path, and it enables multiple optimization to achieve very interesting small message latency. For instance, the NEMESIS subsystem in MPICH2 reaches 300 ns latency [2] thanks to carefully tuned shared data structures and atomic operations.

However, the main drawback of this idea relies in its use of two memory copies per message, which implies twice the amount of memory and cache access. Moreover, since caches are not shared among all cores on existing architectures, many communication patterns may not benefit from all these optimizations. Besides, since memory copies are processed by the host CPU, the host overhead may be significantly high for large messages. This is the main reason why a single-copy based model looks interesting at least for large messages.

B. Open-MX Local Communication

The OPEN-MX [6] stack aims at providing high-performance message passing over any generic ETHERNET hardware. It exposes the *Myrinet Express* (MX) API and ABI to user-space applications and it is also interoperable with hosts running the native MX stack over ETHERNET. Existing middlewares such as MPICH2-MX or OPEN MPI already successfully run unmodified on top of it. Thanks to these features many legacy applications may be executed on top of OPEN-MX with regular ETHERNET hardware.

The OPEN-MX stack may easily be used for intra-node communication since the loopback interface of the operating system is available as well as regular ETHERNET hardware. This implementation has the advantage of being tightly integrated and very easy to use on any hardware while other similar optimizations in MVAPICH would require an INFINIBAND interface and the addition of a specific kernel module [14]. Therefore, OPEN-MX appears as an interesting framework for testing intra-node communication optimization.

Moreover, OPEN-MX is able to improve the receive performance by offloading memory copy on INTEL *I/O Acceleration Architecture* (I/OAT) [7]. Such a feature is known to help intra-node communication [14] by reducing the host overhead and improving large message throughput. However, it involves some memory pinning since physical pages have to be passed to the I/OAT hardware.

C. Motivation and Objectives

Given that shared-memory-based implementations are expected to suffer when no cache is shared between the computing cores or when the message does not fit in the cache, we propose in this paper an extensive study of the single-copy model. The OPEN-MX stack has the advantage of offering a tight integration between regular network communication and intra-node communication. However, given the characteristics of the latter, it still offers a large room for improvement. Therefore, our objectives in this paper are:

- Integrating an optimized intra-node communication model in the OPEN-MX driver without exposing the complexity to user-space ;
- Supporting I/OAT memory copy offload to reduce CPU overhead and cache pollution, and to improve performance ;
- Reducing the need to pin memory and trying to optimize it since it is known to be expensive [15].

III. OPTIMIZING OPEN-MX INTRA-NODE COMMUNICATION

In this section, we describe OPEN-MX single-copy based intra-node communication model and its optimization. We then discuss memory pinning and present how to remove or overlap it.

A. Optimized Single-Copy based Model

Given that OPEN-MX implements the MX protocol on top the generic ETHERNET layer in the LINUX kernel, all application requests are passed to the OPEN-MX kernel driver. Outgoing messages are copied from the application buffers into a *Socket Buffer* that the underlying ETHERNET driver will send. Incoming messages are stored in socket buffers by the driver before OPEN-MX copies their data into the application receive buffers.

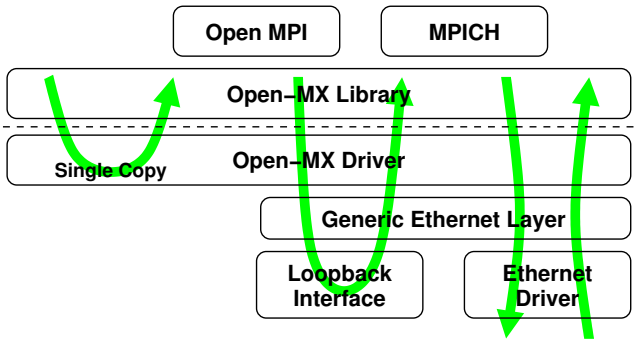

Fig. 1. Design of Open-MX intra-node communication stack.

Instead of relying on the loopback interface to send and receive local messages (stored inside socket buffers), the OPEN-MX driver **bypasses the ETHERNET layer** and directly copies messages from the source application buffers into the target application receive buffers. This model is described by Figure 1.

The first point to optimize regards the **threshold between small and large messages** models in OPEN-MX. Indeed, in the MX wire protocol specification, small messages are sent eagerly and buffered by the receiver. Large messages are actually transfered after a *Rendezvous*, so that no additional copy is needed. The threshold is empirically set to 32 kB but it has to be revised for local communication since the latency and bandwidth are much different. We thus implemented in OPEN-MX a tunable rendezvous threshold that we empirically

set to 4 kB for local communication, which means the eager protocol is much less often used than in regular networking. We will not discuss the eager protocol anymore in this article and we now focus on our single-copy mechanism for large messages.

The second optimization is about **I/OAT copy offload**. The I/OAT hardware (a DMA engine) is available to kernel subsystems. They may submit multiple offloaded copies and poll for their completion later [9]. We added in OPEN-MX the ability to replace regular memory copies with offloaded copies on I/OAT. It has the advantage of not polluting any cache, not using the host CPU, and being faster than regular copies for large messages. However, it has a startup overhead that prevents if from being used for small messages, and it is slower than a regular copy within the cache. OPEN-MX thus uses a tunable threshold to switch its single-copy mechanism from regular to offloaded copies. Given the current processor cache sizes, we empirically set this threshold to 2 MB and let the administrator change it depending on the machine cache size. Table I summarizes OPEN-MX local communication strategies and default thresholds.

Message size	Strategy
<= 4 kB	Eager
4 kB - 2 MB	Rendez-vous + Single-Copy
> 2 MB	Rendez-vous + Offloaded Single-Copy

TABLE I
SUMMARY OF OPEN-MX LOCAL COMMUNICATION STRATEGIES.

B. Removing or Overlapping Memory Pinning

The single-copy mechanism involves a process copying from/to another process address-space. Due to page faults not being possible in another address space, this strategy requires pinning of the other process memory. The OPEN-MX implementation deports the actual copy in the receiver process after the rendezvous. So we have to make sure that the source buffer in the sender process is pinned in physical memory before the receiver can copy from it. Similarly, I/OAT offloaded copies imply passing physical addresses to the DMA engine hardware, which means both source and destination memory buffers have to be pinned as well.

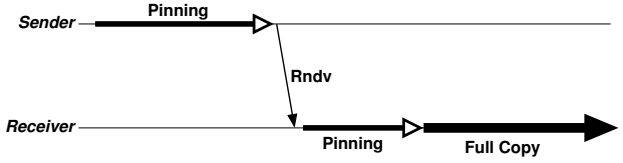

Fig. 2. Single-Copy communication.

All this memory pinning is actually enabled by default on both sides since OPEN-MX networking communications always require it and local communication only differs during the actual copy. It is one severe drawback of this model since memory pinning is known to be expensive (up to 25 % of the whole communication time [15]), and it justifies our work on optimizing it. Figures 2 and 3 summarize the single-copy model in the regular and offloaded cases.

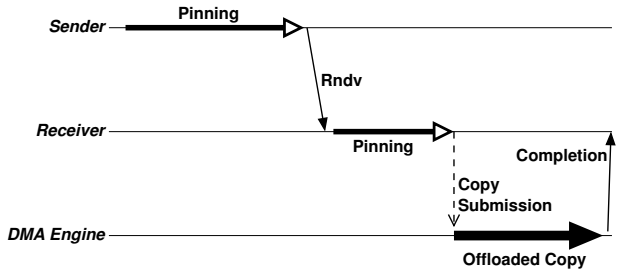

Fig. 3. Offloaded Single-copy communication.

The usual way to reduce the observed memory pinning overhead is to cache pinned buffers to reduce the need to pin/unpin again in case of multiple communications reusing the same buffers (*Pin-down Cache* [13]). This solution is very interesting for benchmarks performance and for applications reusing the same buffers multiple times. But, it reveals some strong technical issues due to the need to detect address space modification to invalidate cached buffers that are freed or unmapped by the application.

We propose in this article to hide the cost of memory pinning by overlapping it with the actual memory copy. This idea may actually be raised thanks to OPEN-MX memory pinning being much more flexible than in high-speed networks. Indeed, optimizing pinning in high-speed networks would require some complex synchronization between the NIC (which processes the actual data transfer), the driver (which takes care of memory pinning), and the application (which submits requests to the NIC through *OS-bypass*). The OPEN-MX model is much simpler since all these components are managed by the kernel driver, making synchronization much easier.

The overall idea of the model is to pin part of a data buffer, start copying from/to this partial buffer, and then pin another chunk. It enables actual overlapping since pinning and copying are not performed by the same process or hardware component. We designed the optimized memory pinning in OPEN-MX to be:

- **Deferred**, to overlap the rendezvous handshake ;
- **Chunked**, so that other processes can start copying from a partially pinned buffer ;
- **Progressively-Agressive**, so that some small chunks are pinned first in order to start copying early, while large chunks at the end reduce the overall pinning overhead ;
- **On-Demand**, so that the receive buffer is not uselessly pinned in the non-offloaded case ;

All these ideas but the latter are mostly about rescheduling pinning and copying in order to reduce the overall application waiting time. The model is summarized on Figures 4 and 5, and Table II. Additionaly, overlapping I/OAT copy offload with memory pinning prevents the processors from being idle while the copies are being processed.

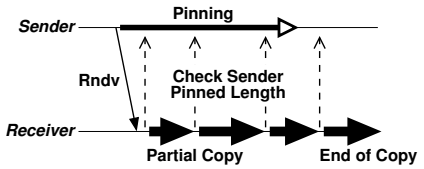

Fig. 4. Overlapped Pinning in Single-Copy communication.

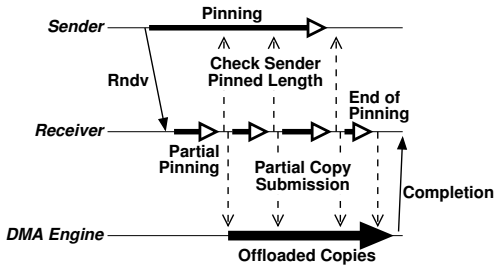

Fig. 5. Overlapped Pinning in Offloaded Single-Copy communication.

IV. Performance Evaluation

We now present a performance evaluation of the Open-MX intra-node communication implementation. We look at the impact of our optimizations, and compare Open-MX performance with Open MPI and MPICH2.

All tests were run on a dual quad-core Xeon 2.33 GHz "Clovertown" machine. MPICH2 1.0.7 (with the Nemesis channel) and Open MPI 1.2.6 were used with their default shared-memory configuration. Open-MX performance was measured by using its MX ABI compatibility which enables loading the Open-MX library as the MX backend in Open MPI[1]

A. Basic Design Performance and Tuning

Figure 6 presents the impact of our basic optimizations in Open-MX on the Intel MPI Benchmarks (IMB [10]) PingPong performance. Using the loopback interface for local communication results in poor throughput. Bypassing the Ethernet layer to enable our single-copy mechanism gives about 30 % better performance for large messages (1300 MB/s) and up to 170 % for 512 kB messages (reaching almost 5 GB/s) since the cache is used much better. The throughput drops significantly after 1 MB messages because the L2 cache size is 2 MB.

[1]The `mpiexec` launcher was used with `--mca pml cm` in order to directly map MPI functions onto the MX ABI.

		Pinning Optimization
Regular Copy	Sender	Overlapped with Copy
	Receiver	Removed since Unused
Copy Offload	Sender	Overlapped with Copy
	Receiver	Overlapped with Copy

TABLE II
Summary of Memory Pinning Optimizations in Open-MX intra-node communications.

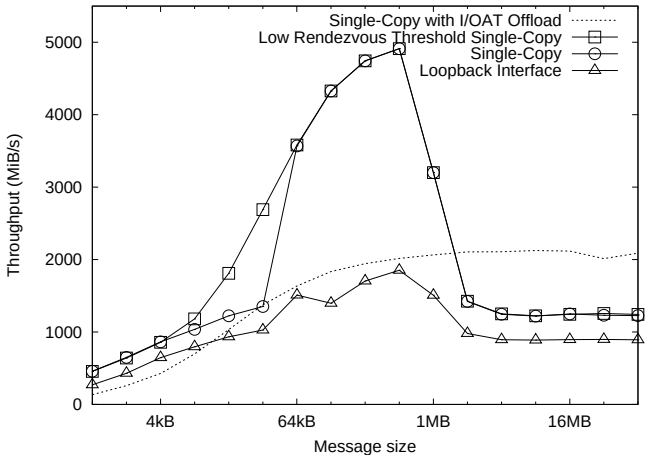

Fig. 6. Open-MX basic optimizations performance with IMB PingPong.

The Figure also shows how important it is to switch to the rendezvous protocol earlier since its throughput is higher than the eager protocol between 4 kB and 32 kB. This is the reason why we empirically chose 4 kB as the threshold for local communication.

Finally, we also observe that offloading memory copies onto the I/OAT DMA engine results in very good performance (up to 2.1 GB/s) for large messages without ever going down since no cache is involved. However, it remains slower than the non-offloaded case when messages fit in the cache, explaining why we decided to enable offload only after 2 MB.

B. Cache Sharing

We now look at the impact of process binding and cache sharing on the throughput. Given that our machine is based on two quad-core processors that contain two independent dual-core subchip with a shared L2 cache, three cases have to be studied:

- **Shared L2 Cache:** 2 processes are running on the same dual-core subchip, using the same shared L2 cache.
- **Shared Die:** 2 processes are running on the same quad-core die but not on the same subchip. No cache is shared, but the link to the memory controller is still shared.
- **Different Dies:** 2 processes are running on different dies. Nothing is shared.

Figure 7 presents the IMB PingPong throughput depending on process binding. We observe that Open-MX performance does not decrease much when there is no shared cache between the processing core. The performance even drops later (after 1 MB instead of 512 kB messages) since each L2 cache is only used by a single process. Also, the offloaded copy performance does not depend on process binding at all since it does not involve any cache.

Figures 8 and 9 show similar behaviors for MPICH2 and Open MPI. When sharing a cache, they are faster than Open-MX up to 512 kB. Their performance then drops faster since two copies are involved (the cache is filled faster). Very large message performance is similar to Open-MX in the

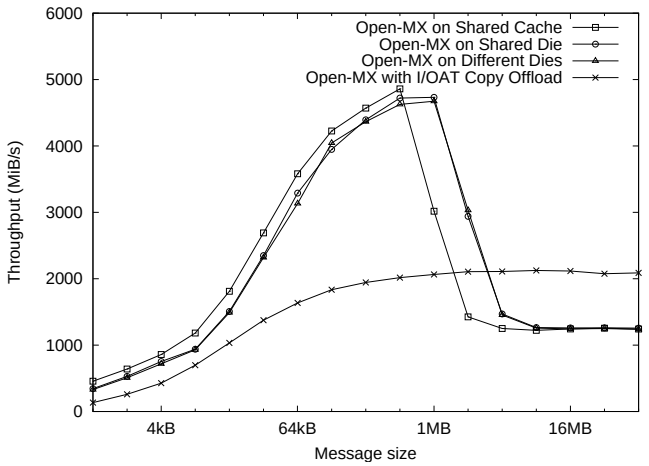

Fig. 7. IMB PingPong performance over OPEN-MX depending on process binding.

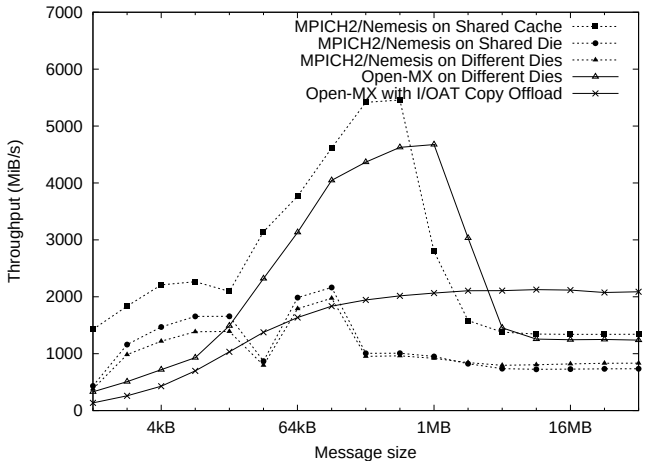

Fig. 8. Compared IMB PingPong performance over MPICH2-NEMESIS and OPEN-MX depending on process binding.

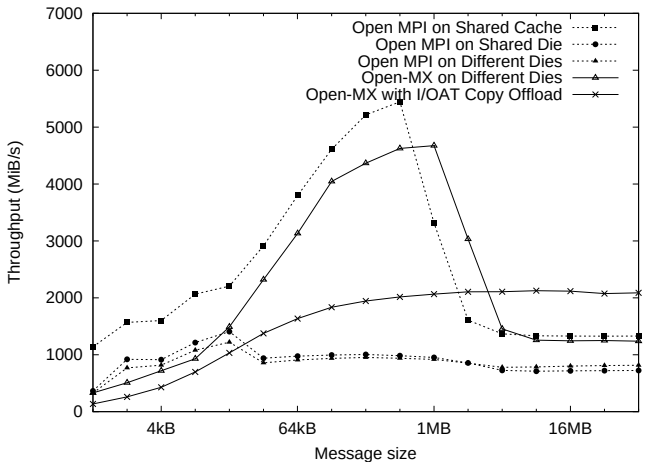

Fig. 9. Compared IMB PingPong performance over OPEN MPI and OPEN-MX depending on process binding.

non-offloaded case, and obviously 35 % below OPEN-MX if I/OAT hardware is available since only OPEN-MX may benefit from it.

When no cache is shared, MPICH2 and OPEN MPI performance drops significantly, from more than 5 GB/s to 1-2 GB/s for medium messages. For very large messages, the throughput saturates below 800 MB/s while OPEN-MX I/OAT copy offload remains above 2 GB/s. Also, we observe that binding processes on the same die decreases large message performance because a single link to the memory controller is used. All these results reveals how using 2 memory copies kills the performance due to the cache not being shared and the memory bus saturating.

C. Overlapped Memory Pinning

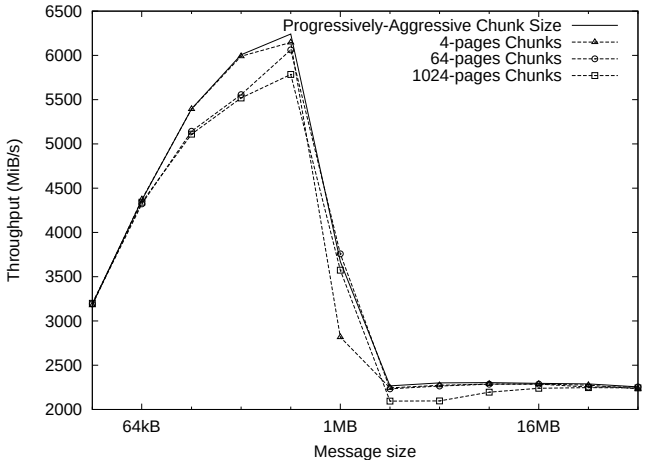

Fig. 10. Impact of chunk size during OPEN-MX overlapped pinning on IMB PingPong performance.

Figure 10 presents the impact of optimizing memory pinning depending on how we split the buffer into chunks. It shows that using large chunks decreases performance since the overlapping of memory pinning has to wait until the first chunk is pinned. Using small chunks gives 10 % better performance since OPEN-MX may then overlap earlier. However, we expect this strategy to add an extra overhead for very large messages since the pinning cost is not linear. Therefore, OPEN-MX now uses a progressively-aggressive chunk size, from 1 page for the first chunk, up to 1024 pages per chunk. It reaches the same performance than always using small chunks without risking of increasing the CPU overhead for very large messages.

Figure 11 compares the impact of optimized overlapped pinning on the IMB PingPong performance (with copy offload enabled after 2 MB) with the impact of the *Pin-down Cache*. It shows that OPEN-MX is now able to achieve the same throughput for very large messages (2300 MB/s, 10 % above the non optimized case) thanks to overlapped pinning. This fully-optimized OPEN-MX performance is 70 % above MPICH2 and OPEN MPI if a cache is shared between processing cores, and up to 3 times better if not. For smaller messages, our work improves performance by up to 27 %

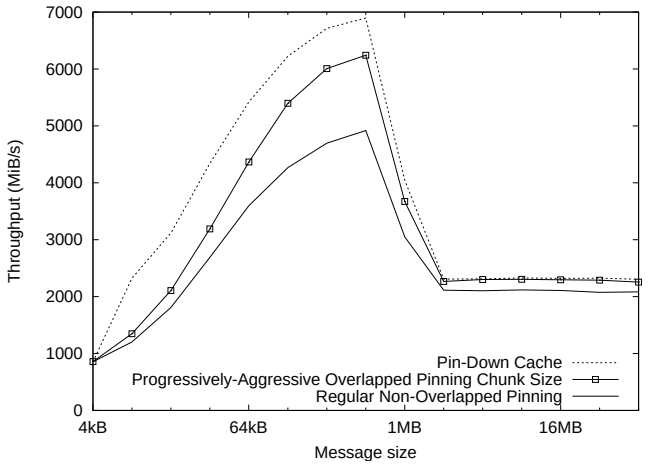

Fig. 11. Performance comparison of IMB PingPong over OPEN-MX with pinning cache, progressively-aggressive overlapped pinning, and regular non-overlapped pinning.

without being able to reach the 45 % improvement of the pin-down cache since the startup overhead of overlapped pinning cannot be avoided.

It has to be noted that apart from being technically difficult to implement safely, the pin-down cache is only guaranteed to improve performance when the application reuses the same buffer many times, while our overlapped pinning always helps.

D. Compared Scalability

Figure 12 presents the compared scalability of OPEN-MX, MPICH2 and OPEN MPI shared memory implementations by looking at IMB Alltoall execution time when running 2, 4 and 8 processes on our machine. We expect OPEN-MX to scale better since it does not rely on shared caches and does not saturate the memory bus as well.

We observe that the more processes are involved, the faster OPEN-MX passes OPEN MPI and MPICH2 performance (from 64 kB with 2 processes down to 8 kB for 8 processes). The performance improvement is also bigger with the number of processes, up to 5 for 64 kB messages with 8 processes.

It also has to be noted than I/OAT copy offload brings a very interesting performance improvement (up to 33 %) even before the 2 MB threshold that we empirically set. This reveals the strong advantage of this hardware on loaded configurations where saving some CPU cycles and not polluting caches too much may become critical. We are thus looking at reducing the copy offload threshold further.

The impact of optimized memory pinning on this test goes up to 10 % and is even negative in some cases. We think that it is caused by Alltoall patterns loading the machine much more and thus revealing the fact that overlapped pinning does not reduce the overhead but only reschedules it.

E. NAS Parallel Benchmarks

We finally look at the NAS parallel benchmarks [1] performance. Table III summarizes the performance improvement brought by the OPEN-MX implementation on several cases.

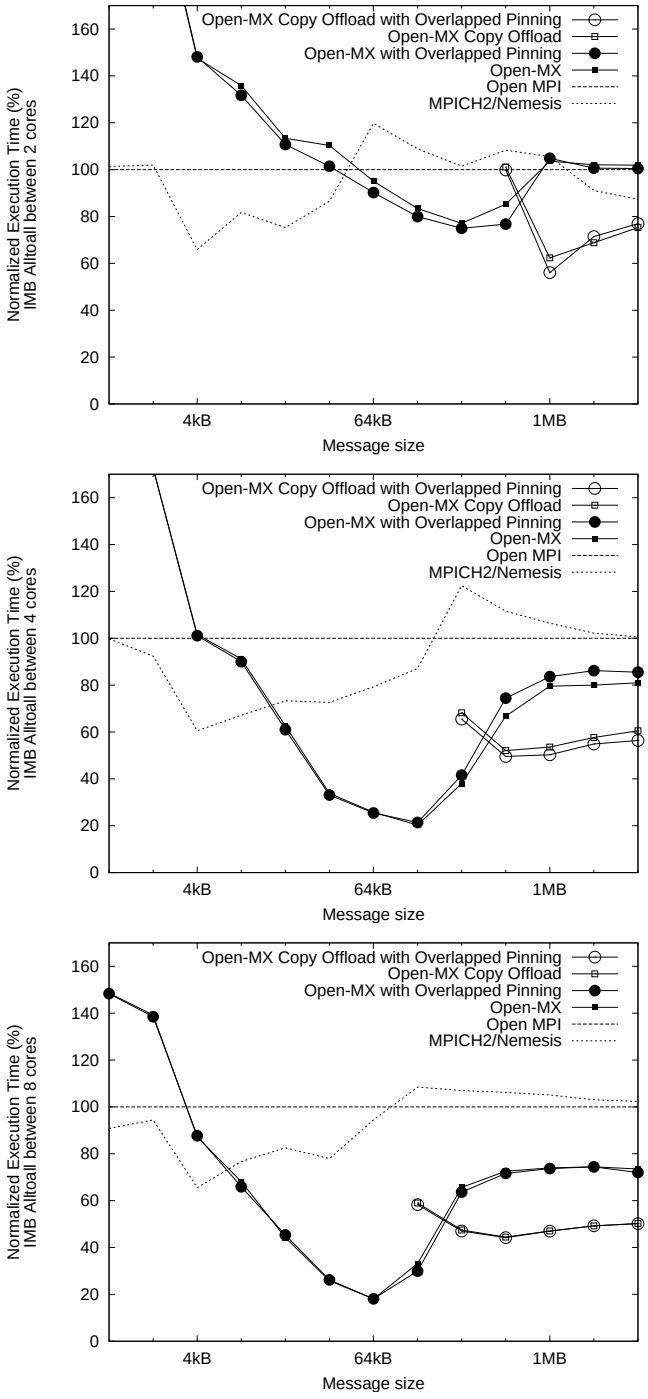

Fig. 12. IMB Alltoall normalized execution time over MPICH2-NEMESIS, Open MPI and Open-MX when increasing the number of processing cores.

We did not observe any significant difference between OPEN-MX and OPEN MPI or MPICH2 in other cases. These numbers were gathered by binding 4 processes on the same die but we observed similar relative behaviors with other bindings.

Benchmark	OPEN MPI	MPICH2	OPEN-MX
sp.B.4	304.01 s	306.55 s	301.83 s
ft.B.4	49.07 s	49.10 s	46.73 s
is.C.4	11.20 s	11.19 s	10.20 s

TABLE III
NAS PARALLEL BENCHMARKS EXECUTION TIME.

We observe that OPEN-MX brings 1-2 % improvement on SP and 3 % on FT. The more interesting results comes from IS, which is known to rely on many large messages, where OPEN-MX is 10 % faster thanks to our optimizations. Looking in detail at the impact of our optimizations, we note that overlapped memory pinning brings about 3 % improvement while I/OAT copy offload obviously brings the biggest part (7 % on average).

V. RELATED WORKS

Optimizing intra-node MPI communication has been the topic of many research works. MPICH2 [2], OPEN MPI [5], and MVAPICH [12] all rely on a shared memory-mapped file to achieve very good small message latency. MPICH2 pushed this model further by optimizing cache misses and using atomic operations to reach a 300 ns latency on modern machines. We showed in this paper that the throughput of this two-copy model however heavily relies on a cache being shared by the processing core. These implementations cannot use a single-copy mechanism since it requires kernel support while they are implemented as user-space libraries. OPEN-MX tightly integrates this support within its main network communication stack, making it easily available to all applications thanks to its MX compatibility.

MVAPICH has been modified to use a single copy mechanism with I/OAT copy offload support [14] but this support remains an ad-hoc extension. The OPEN-MX implementation is very simple to administrate and exploit, without requiring the administrator to load an additional kernel driver to extend a user-level MPI implementation. The single-copy support in MVAPICH also relies on basic memory pinning which appears to be very expensive. It justifies our work on hiding pinning through overlap.

Using multiple DMA channels simultaneously[2] may improve copy throughput by up to 40 % [15]. However, OPEN-MX assigns a single channel per message and only relies on multiple channels to handle multiple outstanding messages. This strategy reduces the management cost without much decreasing the overall performance since we expect modern multicore machines to use many processes at the same time and thus have many outstanding messages use all the available DMA channels.

[2]There are 4 or 8 independent DMA channels on current INTEL I/OAT hardware.

The overhead of memory pinning has been often worked around using a pin-down cache [13] to reuse formerly pinned pages. This model however requires tracing of address space modifications to maintain the cache validity, which means some system calls have to be intercepted in user-space [15] or the kernel has to be modified [8] to do so. It is also not guaranteed to improve performance since the application has to reuse the same buffer multiple times before the impact of the cache becomes visible. Our overlapped memory pinning does not always bring the same performance improvement but it still improves performance a bit even if the application does not reuse its memory buffers multiple times.

VI. CONCLUSION

As the number of cores increases in high-performance computing nodes, intra-node communication performance becomes more and more important. Many research works have been proposed to improve small message latency, but the generally used shared memory-mapped file model reveals a large dependency on a cache being shared by the processing cores. A single-copy model through the operating system implies less cache pollution, memory bus saturation, and CPU overhead, but it requires a kernel implementation.

We presented in this article several optimizations for this single-copy local communication model. They have been implemented in the OPEN-MX stack[3] which offers a transparent integration of networking and local communication and which is easily available on any hardware. We first described how we bypass the ETHERNET layer in the OPEN-MX driver to implement the single-copy mechanism without exposing the complexity to user-space. Thanks to this model being implemented in the LINUX kernel, it may also offload memory copies onto I/OAT hardware. We then explained how this strategy had to be tuned for local communication to benefit from the performance of the rendezvous strategy and copy offload earlier. We finally presented an overlapped memory pinning mechanism which enables the OPEN-MX single copy strategy to start copying while the involved buffers are still being pinned.

This implementation offers similar performance to shared-memory models such as OPEN MPI and MPICH2 but it does not suffer of no cache being shared between the processing cores. Offloading memory copies onto the I/OAT hardware enables a 70 % throughput improvement for large messages. Finally, overlapping memory pinning brings from 10 to 25 % throughput improvement, making the need for the pin-down cache less important. In the end, OPEN-MX throughput for large messages is now between 1.7 and 3 times above MPICH2 and OPEN MPI depending on cache sharing. When no I/OAT hardware is available, the improvement goes up to 80 % if no cache is shared.

This article only discusses large message throughput. We are now looking at improving small messages in OPEN-MX as well. The current kernel-based model still achieves a high

[3]OPEN-MX is available for download at http://open-mx.org.

latency due to a system call being involved. We expect having to switch to shared-memory model for small messages to approach MPICH2 latency.

We are also looking at applying overlapped memory pinning to network communication as well. However, we expect a smaller performance improvement since the network bandwidth is lower and thus the pinning overhead is relatively lower.

Finally, we still plan to improve our copy offload model. First, our offload threshold is mostly decided by looking at the non-offloaded copy performance in the shared-cache case. However, in the non-shared-cache case, this threshold could well be reduced. We are thus looking at checking whether processes share a cache before deciding to offload the copy or not. We are also trying to release the CPU when waiting for offloaded copies to complete. The I/OAT hardware does not send any interrupt that could wake up sleeping processes. But we expect to be able to estimate the completion time and let the process sleep a bit before going back to polling for I/OAT completions.

In the end, even if not polluting any cache appears to be a strong advantage of I/OAT copy offload, pre-warming the cache may be interesting since the application is probably going to use the data right after receiving it. For this reason, it may actually be good to use a regular memory copy for the beginning of large messages, and then switch to I/OAT copy offload.

REFERENCES

[1] 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*, 5(3):63–73, Fall 1991.

[2] Darius Buntinas, Guillaume Mercier, and William Gropp. Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: Proc. 13th European PVM/MPI Users Group Meeting*, Bonn, Germany, September 2006.

[3] Franck Cappello and Daniel Etiemble. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, Dallas, TX, November 2000.

[4] Lei Chai, Albert Hartono, and Dhabaleswar K. Panda. Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'06)*, Barcelona, Spain, September 2006.

[5] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[6] Brice 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, April 2008. IEEE Computer Society Press.

[7] Brice Goglin. Improving Message Passing over Ethernet with I/OAT Copy Offload in Open-MX. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 223–231, Tsukuba, Japon, September 2008. IEEE Computer Society Press.

[8] Brice Goglin, Loïc Prylli, and Olivier Glück. Optimizations of Client's side communications in a Distributed File System within a Myrinet Cluster. In *Proceedings of the IEEE Workshop on High-Speed Local Networks (HSLN), held in conjunction with the 29th IEEE LCN Conference*, pages 726–733, Tampa, Florida, November 2004. IEEE Computer Society Press.

[9] Andrew Grover and Christopher Leech. Accelerating Network Receive Processing (Intel I/O Acceleration Technology). In *Proceedings of the Linux Symposium*, pages 281–288, Ottawa, Canada, July 2005.

[10] Intel MPI Benchmarks. http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219847.htm.

[11] Myricom, Inc. *Myrinet Express (MX): A High Performance, Low-Level, Message-Passing Interface for Myrinet*, 2006. http://www.myri.com/scs/MX/doc/mx.pdf.

[12] Network-Based Computing Lab, The Ohio State University. MVAPICH: MPI for InfiniBand over VAPI Layer. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.

[13] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down cache: A Virtual Memory Management Technique for Zero-copy Communication. In *Proceedings of the 12th International Parallel Processing Symposium*, pages 308–315, April 1998.

[14] K. Vaidyanathan, L. Chai, W. Huang, and D. K. Panda. Efficient Asynchronous Memory Copy Operations on Multi-Core Systems and I/OAT. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'07)*, Austin, TX, September 2007.

[15] K. Vaidyanathan, W. Huang, L. Chai, and D. K. Panda. Designing Efficient Asynchronous Memory Operations Using Hardware Copy Engine: A Case Study with I/OAT. In *Proceedings of the International Workshop on Communication Architecture for Clusters (CAC), held in conjunction with IPDPS'07*, page 234, Long Beach, CA, March 2007.