High-Performance Message Passing over generic Ethernet Hardware with Open-MX

Brice Goglin

High-Performance Message Passing over generic Ethernet Hardware with Open-MX

Brice Goglin

INRIA Bordeaux - Sud-Ouest – LaBRI
351 cours de la Libération – F-33405 Talence *– France*

Abstract

In the last decade, cluster computing has become the most popular high-performance computing architecture. Although numerous technological innovations have been proposed to improve the interconnection of nodes, many clusters still rely on commodity Ethernet hardware to implement message passing within parallel applications. We present Open-MX, an open-source message passing stack over generic Ethernet. It offers the same abilities as the specialized *Myrinet Express* stack, without requiring dedicated support from the networking hardware. Open-MX works transparently in the most popular MPI implementations through its MX interface compatibility. It also enables interoperability between hosts running the specialized MX stack and generic Ethernet hosts. We detail how Open-MX copes with the inherent limitations of the Ethernet hardware to satisfy the requirements of message passing by applying an innovative copy offload model. Combined with a careful tuning of the fabric and of the MX wire protocol, Open-MX achieves better performance than TCP implementations, especially on 10 gigabit/s hardware.

Key words: Message Passing, MPI, Ethernet, Copy Offload, Myrinet Express

1. Introduction

High-performance computing has become increasingly important over the last three decades. It has spread into many domains, from research to industry, from particle physics to seismology and automotive crash simulation. This trend has led to many technological innovations. Although the most powerful computing installations use custom supercomputer architectures [1], the current market is dominated by clusters. These machines are assembled from regular workstation nodes and the *Message Passing Interface* (MPI [2]) is the standard for communicating between nodes within parallel applications.

The interconnection between cluster nodes has been the subject of much research. Indeed this internode communication could have a dramatic impact on the overall performance of parallel applications. Numerous software and hardware optimizations have been proposed, and specialized fabrics and protocols such as InfiniBand [3] and Myrinet [4] have been designed to implement the MPI standard more efficiently. However these dedicated technologies remain expensive and only target a limited set of applications in which latency or network bandwidth is critical. As a consequence, most current clusters still rely on commodity networking technologies such as Ethernet [1]. Although these specialized and commodity interconnects are expected to converge in the near future, the gap between them in terms of performance and capabilities remains large.

It is not yet clear which features will be part of the future converged technology. Ethernet already appears as an interesting networking layer within local networks for various protocols such as FibreChannel [5] and ATA [6]. Several research projects are seeking to implement high-performance MPI stacks over generic

Ethernet hardware [7, 8, 9, 10, 11]. In this paper we introduce an open-source¡ message passing stack called OPEN-MX that addresses this issue while trying to remain compatible with the existing *Myrinet Express* stack (MX [12]) for Myricom fabrics. Thanks to this compatibility, OPEN-MX is immediately available to most existing MPI implementations, for instance OPEN MPI [13] and MPICH2 [14].

The MX stack has been designed for high-performance message passing over specialized HPC networking hardware. OPEN-MX aims to offer the exact same capabilities on top of generic Ethernet hardware. It also enables interoperability between Myricom specialized hardware and commodity Ethernet hardware. This feature has been experimented as a way to provide the networking layer for PVFS2 [15] in BLUEGENE/P systems. However, Ethernet-based protocols often suffer from the limited capabilities of standard Ethernet hardware. We therefore describe in this paper how OPEN-MX circumvents these limits and achieves high-performance message passing using innovative optimizations in the networking stack.

This paper is an extended revision of [16, 17] which introduced the OPEN-MX stack. It is organized as follows: Section 2 describes the past and current trends in HPC and its networking-based solutions. Section 3 then motivates our work and details its objectives and design. Actual performance issues with message-passing over Ethernet and our solutions in the OPEN-MX stack are then presented in Section 4 and evaluated in Section 5.

2. High-Performance Networking over Ethernet

Cluster computing emerged about fifteen years ago with the NOW [18] and *Beowulf* [19] projects which assembled hundreds of commodity workstations into larger computing systems. It has since resulted in the development of dedicated high-speed networking technologies that are now widespread in modern clusters. However, many clusters still use traditional networking technologies when communication performance is not critical. Here, we detail the main commodity and specialized networking solutions that have been proposed for both hardware and software.

2.1. High-Speed Networking in HPC

The widespread use of clusters in high-performance computing was eased by a reasonable performance/price ratio because most hardware components are now considered to be mainstream. However, assembling such a distributed system from many physically-separated machines quickly raised the problem of the cost of external communication between them. Indeed, the latency between two processors in the cluster could almost reach a millisecond, up to a hundred times slower than in a massively parallel supercomputer. This problem has led to the development of dedicated networking technologies so as to improve communication performance without requiring highly-specialized modifications of the host.

High-speed networks such as MYRINET [4] or more recently INFINIBAND [3] have been widely used in clusters for a decade, now representing about 30% of the TOP500 [1]. These technologies rely on scalable topologies to prevent congestion, advanced network interface controllers (NIC) that can offload communication management, and optimized message passing software stacks that bypass most of the traditionally-expensive communication layers. The innovations on the end-nodes include the following:

Zero-copy data transfer between the user-level application memory and the NIC through DMA (*Direct Memory Access*), so as to reduce latency and CPU copy overhead.

OS-bypass operations allowing the application to submit communication requests to the NIC without going across all the expensive operating system layers. Request completion may also be polled explicitly by the application because the NIC deposits packets and events directly in the user-space memory.

Overlap of communication with computation due to asynchronous interfaces and offloading of most of the communication processing into the NIC.

2.2. HPC with Commodity Networks

Communication-sensitive applications may obtain significant performance improvement due to dedicated high-speed network technologies [20]. However, because communication is not a major bottleneck for most applications, many clusters (more than half of the TOP500 [1]) still use commodity networking technologies such as gigabit/s Ethernet. Indeed, most parallel applications use an implementation of the *Message Passing Interface* (MPI [2]) on top of TCP, even though TCP/IP has often been recognized as being slow in the context of high-performance computing.

TCP/IP is known to have large overheads, especially due to its memory copies [21]. Its inability to benefit from high-speed technologies was demonstrated once multi-gigabit/s technologies emerged [22]. In the meantime, lighter protocols were designed to bypass the expensive traversal of all the operating system layers. *Active Messages* [23] proposed a message-driven interface that eliminates buffering beyond the requirements of network transport and offers some overlap capabilities. Another example is *Fast Messages* [24] which leverages high-performance messaging capabilities up to the MPI implementation.

The aforementioned custom protocols raised the need for advanced capabilities in the network interface to facilitate high performance message passing. Although some dedicated offloading capabilities were added to specialized high-speed networks, commodity hardware has been slowly improving, mostly only for IP networking. Most modern adapters may now offload checksum computations as well as transmit segmentation (TSO). Some advanced boards are even able to split the traffic into multiple queues so as to scatter the workload among multiple cores (*Multiqueue* [25]), or to gather multiple consecutive incoming packets to reduce the host overhead on the receiver side (*Large Receive Offload* [26]). The offloading of the entire TCP stack into the network interface was also proposed but was never accepted in practice due to many technical problems [27]. TCP/IP performance has dramatically improved in recent years thanks to these hardware improvements, now easily reaching line rates of 10 gigabit/s on modern servers. However, the gap between these commodity networks and dedicated message passing stacks has not disappeared, and it has led to further work on improving message passing on top of Ethernet technologies.

2.3. Convergence between High-Speed Networks and Ethernet

Specialized high-speed networks and commodity technologies diverged significantly in the last decade as they target different use cases, message passing or TCP/IP. Data centers now often use both technologies for different purposes, but they also often use a dedicated storage network such as FIBRECHANNEL. This has resulted in convergence between administrative (commodity) networking and storage networking solutions through the *FibreChannel-over-Ethernet* standard (FCoE [5]). Vendors have now been talking about the possible convergence of high-speed networks and commodity networks towards a single technology offering the best features and performance for multiple usages. However, it raises the question of how the existing software and hardware HPC innovations can be translated to mainstream technologies.

The re-implementation of protocols over the low-level Ethernet layer is a new trend that was initiated in the context of storage with *ATA-over-Ethernet* (AoE [6]) and has now spread into the HPC market. Myricom followed this path in 2005 when introducing the MYRI-10G hardware [28] which makes 10G Ethernet and the specialized network interoperable due to the XAUI-compatible physical port. It is then possible to use Myricom high-performance message passing stack (*Myrinet Express over Ethernet*, MXoE) on top of standard 10G Ethernet networks thanks to the addition of Ethernet routing headers to MX packets by the interface. QUADRICS QSNET III was following a similar path before the company closed down in 2009. INFINIBAND was also supposed to become interoperable with Ethernet because the MELLANOX CONNECTX interface is

able to switch from one protocol to another at startup [29]. However the incompatibility between these protocols at the physical level limits their interoperability in practice. The emergence of *Converged Enhanced Ethernet* may enable the actual wide deployment of INFINIBAND protocols over Ethernet (RoCEE [30]) and make the interoperability finally possible.

As no standardization effort towards convergence between Ethernet and HPC were actually proposed, many research projects attempted to improve message passing over Ethernet in the meantime. GAMMA [7] and PM-Ethernet [8] modify some low-level drivers and replace the existing IP stack so as to provide improved message passing stacks on some popular gigabit/s hardware. MULTIEDGE [11] uses a similar design on recent 1 and 10 gigabit/s hardware and thus achieves good bandwidth, but it does not offer small latencies. EMP [31] goes even further by modifying the firmware of some programmable boards to achieve better performance. Some projects such as MPI/QMP [9] and PM-Ethernet [10] focus on unmodified low-level drivers and try to improve message passing performance by using the aggregated throughput of a trunked Ethernet connection. However, they are not designed for single high-performance links such as modern MYRI-10G NICs. Moreover, all these projects, as well as some proprietary solutions such as PARASTATION [32], rely on custom MPI implementations instead of contributing their work to widespread implementations such as MPICH2 [14] or OPEN MPI [13], which are portable and considered stable and efficient.

The only attempt at standardization of high-performance message passing over Ethernet was iWARP which uses an IB-like model to provide RDMA semantics at the application level within the TCP stack. It achieves good performance on RDMA-enabled NICs [33]. A software-only implementation is also available for generic hardware [34] but its performance is significantly limited. Indeed iWARP was designed for RDMA-enabled NICs and thus suffers from additional memory copies when used on top of non-RDMA NICs. Moreover, iWARP is highly similar to a modified high-performance TCP/IP stack that targets long distance connections where the CPU overhead should be as low as possible, and it is not only limited to cluster computing.

Although many projects sought to improve message passing over Ethernet, most of them require intrusive hardware or software modifications, and do not provide as high throughputs and as low latencies as expected on modern 10G hardware.

3. Design of a Modern Message Passing Stack for Ethernet

In this section, we introduce the OPEN-MX open-source software stack which is available for download from `http://open-mx.org`. It targets the following objectives:

O1 To expose a standard and/or popular programming interface to ease interoperability, so that existing general-purpose MPI implementations might work on top of it.

O2 To work on any commodity/generic Ethernet hardware without requiring any specific hardware capabilities.

O3 Not to break the existing networking stack or drivers because they may be used for administration or storage simultaneously.

O4 To provide high-performance message passing to applications by efficient use of modern hardware, and to take advantage of advanced features if available, such as trunked Ethernet connections, scatter/gather, or copy offload.

3.1. Compatibility with Myrinet Express

The aforementioned objective **O1** of exposing a standard and/or popular interface in OPEN-MX required an in-depth analysis of current trends in high-performance computing. Many custom programming interfaces have been proposed for existing message passing stacks as explained in Section 2.3. However, none of them became popular or widely used because application developers in HPC tend to continue using the same interfaces. The *Message Passing Interface* (MPI [2]) is the current *de facto* standard. However, it covers more than only basic networking: for instance, it offers collective operations. A message passing driver does not need to handle these features that are already efficiently implemented and widely tested in general-purpose MPI layers such as OPEN MPI or MPICH2. One popular message passing driver that works underneath all popular MPI implementations is *Myrinet Express* (MX [12]). It exposes all necessary *Point-to-point* communication capabilities that are used by MPI as well as storage systems such as PVFS or LUSTRE.

We thus designed our OPEN-MX networking stack to offer the **MX programming interface** as well as its **binary interface**. It enables the linking of existing applications over OPEN-MX without rebuilding, for instance through OPEN MPI or PVFS. Secondly, we made OPEN-MX **wire-compatible** with MXoE, the Ethernet flavor of MX (see Section 2.3). It enables interoperability between any generic hardware running OPEN-MX and Myricom hardware running MX when they are connected to the same Ethernet fabric. This wire compatibility is being tested at the Argonne National Laboratory to provide a PVFS2 [15] transport layer between BLUEGENE/P [35] compute and I/O nodes. The compute nodes running OPEN-MX are connected through a BROADCOM 10 gigabit/s Ethernet interface to I/O nodes with a MYRI-10G interfaces running the native MXoE stack as depicted in Figure 1.

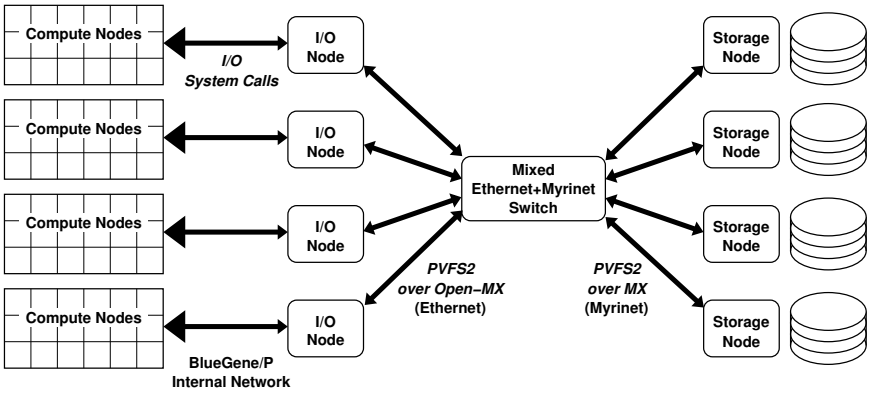


Figure 1: OPEN-MX in the BLUEGENE/P PVFS2 networking layer. OPEN-MX enables the removal of the IP layer for direct use of the native message passing capabilities of the fabric.

OPEN-MX implements the MX protocol which consists of three main methods for passing messages. *Small* messages (below 128 bytes) are optimized along the sender side by writing data in the NIC using a PIO and passing a single packet on the wire. *Medium* messages (from 129 bytes to 32 kB) are copied in a statically pinned buffer on both sides and the hardware transfers them through DMA and within multiple packets if necessary. *Large* messages (> 32 kB) are not sent through an eager protocol. They are processed through zero-copy after a rendezvous and memory pinning. The sender passes a window handle to the receiver which pulls data from it in 32 kB *blocks*. Multiple blocks may be requested in parallel, and each of them is transferred as a set of eight 4 kB-packets by default. The MX wire protocol only specifies the message types, lengths and fragmentation. It does not force OPEN-MX to use a similar PIO or DMA imple-

mentation. In the following sections, we detail how we translated this MX implementation into OPEN-MX, and which non-MX-compatible wire extensions have been introduced.

3.2. Emulating the MX stack in software

The *Myrinet Express* software stack has been designed for high-performance message passing [12]. Therefore, it exploits the capabilities of MYRINET and MYRI-10G hardware and firmware at the application level while providing low-latency and high bandwidth (2 μs and 1250 MB/s data rate). For this purpose, OS-bypass communication is used, as along with zero-copy for large messages. The operating system is not involved in the communication. Only the initialization phase and the memory registration of large messages require its assistance (see Figure 2). All the actual communication management is implemented in the user-space library and in the firmware. The MX firmware is also responsible for selecting packet headers depending on the fabric: Ethernet headers when talking to standard Ethernet switches (MXoE mode), or MYRINET specific headers when talking to Myricom switches (native MX mode).

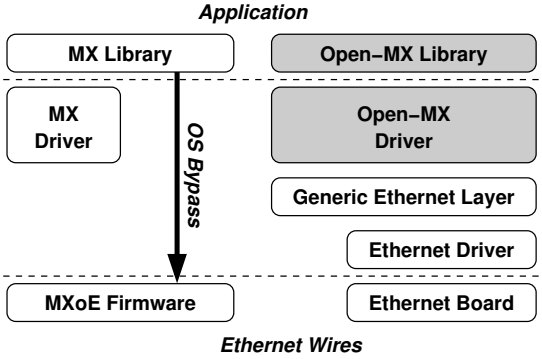


Figure 2: Design of the native MX and generic Open-MX software stacks.

We use a similar design for the OPEN-MX implementation while retaining the ability to diverge from this model later if necessary. However, OPEN-MX cannot rely on any specific hardware features because only the generic Ethernet software stack and hardware features are available. OPEN-MX is thus implemented on top of the Ethernet layer of the LINUX kernel and must emulate the native MX firmware (see Figure 2). Although it is portable across all existing Ethernet hardware that LINUX supports, this interface is simple as depicted by Figure 3. Thanks to this model, OPEN-MX works transparently on any generic NIC, without requiring any advanced feature, and it maintains the other existing stacks such as IP intact (objectives **O2** and **O3**).

The OPEN-MX stack is therefore composed of a user-space library (similar to the native MX one), and a kernel driver that replaces the MX NIC firmware by sending and receiving raw Ethernet messages (*Socket Buffers*, skbuff). Because these skbuffs are managed by the Ethernet layer in the LINUX kernel, OS-bypass is impossible. Whereas MX uses PIO from user-space, OPEN-MX relies on a system call which places the application data in a skbuff and passes it to the Ethernet driver for sending. Section 4 describes more specifically how socket buffers are used for data movement in OPEN-MX. Fortunately, the cost of system calls has decreased dramatically in recent years, making OS-bypass almost useless. Indeed, on modern AMD and INTEL processors, the basic cost of a system call is close to 100 nanoseconds. In earlier systems, it was often above 500 ns, making it undesirable on latency-critical paths.

The OPEN-MX user-space library provides the same features that are available in the native MX library. It first matches incoming messages against posted receives and makes the communication protocol progress in case of *rendezvous*. Secondly, it manages retransmission by acknowledging or resending messages if necessary. The OPEN-MX library also offers thread-safety to ease the development of middleware

```
/* Register receive handler function for incoming Open-MX packet type */
struct packet_type omx_pt = {
  .type = __constant_htons(ETH_P_OMX),
  .func = omx_recv,
};
dev_add_pack(&omx_pt);

/* Send Socket Buffer skb using Interface ifp */
skb_header->type = htons(ETH_P_OMX);
skb->dev = ifp;
dev_queue_xmit(skb);
```

Figure 3: Summary of the Ethernet programming interface in the LINUX kernel.

or applications with multiple threads concurrently accessing the networking stack, for instance PVFS2 or `MPI_THREAD_MULTIPLE` support in MPI implementations [36]. Finally, OPEN-MX also handles communication with other processes on the same host (*Intra-node communication*) in an optimized manner [37]. Even though the major MPI implementations already handle this special case, a few implementations, such as MPICH-MX do not. Because this feature must always be available to any application, OPEN-MX offers this ability and lets the MPI implementation bypass it when desired.

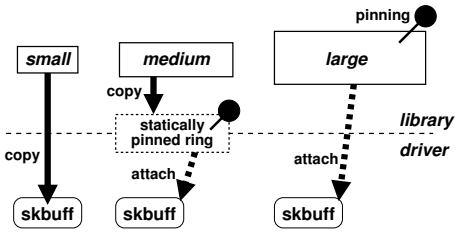
4. High-Performance Message Passing over Ethernet

High-performance interconnects yield high throughput data transfers due to a carefully optimized data path. For instance, memory copies are avoided as much as possible so as to improve bandwidth, to decrease CPU consumption, and to reduce cache pollution,. These zero-copy strategies rely on the capability of both hardware and driver to manipulate the host's physical memory in a flexible way. However, such advanced features are not available in generic Ethernet hardware. We now detail the implementation of the OPEN-MX send and receive stacks, the management of memory copies, and how the model may be tuned to achieve high performance.

4.1. Overall Implementation

As explained in Section 3.2, although MX directly drives the NIC, OPEN-MX uses system calls that manipulate both *Socket Buffers* and low-level Ethernet drivers. Modern networking hardware may read/write from non-contiguous memory regions (*Scatter/Gather*, objective **O4**). Therefore, the OPEN-MX driver may decide at runtime between copying the outgoing data in a newly-allocated contiguous buffer (usually for small messages), or directly passing references to the applications buffers to the NIC by attaching its pages to the skbuff (preferable for large messages). The default path from the application to the network through the user-space library and kernel driver is summarized in Figure 4. It uses as many memory copies as the native MX stack, i.e. one copy for small and medium messages and no copy for large messages However, due to the wide variety of hardware that OPEN-MX supports, it also enables the administrator to tune its behavior and thresholds depending on the underlying hardware capabilities and performance.

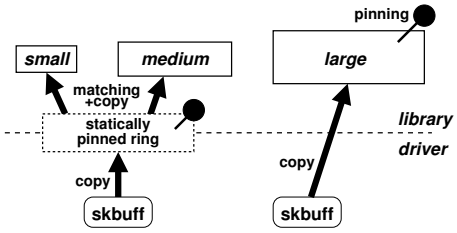
Although skbuff management is easy on the sender side, the receiver side of OPEN-MX raises several important issues. First, the OPEN-MX stack cannot decide where incoming data will be stored by the hardware. The Ethernet driver is responsible for allocating its own skbuffs and passing them to OPEN-MX once the NIC has filled them, which makes zero-copy impossible on the receiver side. This problem is actually



	Open-MX		MX	
	Library	Driver	Library	NIC
Small	×	Copy	PIO Copy	×
Medium	Copy	Attach+DMA	Copy	DMA
Large	Pin	Attach+DMA	Pin	DMA

Figure 4: Open-MX send strategies and comparison with MX. See Section 3.1 for details about message types.

not Open-MX-specific, and any other Ethernet-based protocol suffers from the same issue. It has resulted in the design of custom hardware, protocols and software stacks to avoid memory copies by replacing the TCP/IP implementation. For instance RDMA-enabled NICs and drivers, such as EMP [31] or iWarp [33], let the advanced hardware place the data in the right receive buffers. Unfortunately, this problem implies that the performance improvements achieved through these technologies cannot be obtained with standard Ethernet hardware. Secondly, the interrupt-driven model of the Ethernet stack causes the incoming packets to be processed by the *Bottom Half Interrupt Handler*, i.e. outside of the application's context. The NIC specific handler invokes the Open-MX receive handler to process the newly-received skbuff, which means that the target application page table is not necessarily available when processing the data.



	Open-MX		MX	
	Driver	Lib	NIC	Lib
Small	Copy	Copy	DMA	Copy
Medium	Copy	Copy	DMA	Copy
Large	Copy	Pin	DMA	Pin

Figure 5: Open-MX receive strategies and comparison with MX.

In the native MX receive stack, small and medium incoming messages are copied into a statically allocated user-space ring, and the user library can move the data to the application buffers after the matching. For large messages, the NIC already knows the final destination buffer because the data transfer occurs after a *rendezvous*. The Open-MX receiver side mimics these strategies by having the receive handler copy the data where the MX NIC would have placed it directly. The obvious drawback of this strategy lies in the additional memory copy that it involves, as shown in Figure 5. This model puts high pressure on both the CPU and the memory bus and thus limits the receiver side performance.

One way to avoid memory copies is to use virtual memory tricks to remap the source buffer in the target virtual address space. Such a strategy has been studied for a long time and zero-copy socket implementations have been proposed [38]. However, for this remapping to be possible in practice, virtual pages must be aligned in the same way in the source and destination buffers. It has been proposed to embed alignment constraints in the wire protocol as a way to circumvent this problem [39]. However, this solution does not apply to the existing general-purpose wire protocols that cannot include such alignment information. Also, remapping induces numerous pathological cases because modern operating systems rely heavily on multiple page table states, pages being shared, or memory pinning. And it induces a non-negligible page-table synchronization overhead on modern manycore platforms. This makes remapping difficult and expensive in many cases, although it is indeed a powerful method to improve performance and reduce the CPU load. Because it seems hard to avoid memory copies without dedicated support in the NIC and protocol, we now

introduce a solution based on reducing the impact of these copies by offloading them, as planned in objective **O4**.

4.2. Opportunities for I/O AT copy offload in OPEN-MX

INTEL's *I/O Acceleration Technology* (I/O AT) is a set of hardware features tailored to improve networking performance in data centers [40]. One interesting feature is the ability to perform asynchronous memory copies in the background because a DMA engine is integrated in the memory chipset (see Figure 6). It enables memory copies to be overlapped with neither CPU usage nor cache pollution, in contrast to the usual `memcpy` method. Such hardware has already been available in all modern INTEL servers for several years. The DMA engine programming interface of the LINUX kernel [41] is mainly used by the TCP/IP receiver stack. It offloads memory copies while the user process sleeps in the `recv()` system call until there is enough data to receive. The CPU usage is reduced and the network throughput is improved for various applications such as PVFS file transfers [42]. To the best of our knowledge, no other networking stack is yet using I/O AT copy offload.

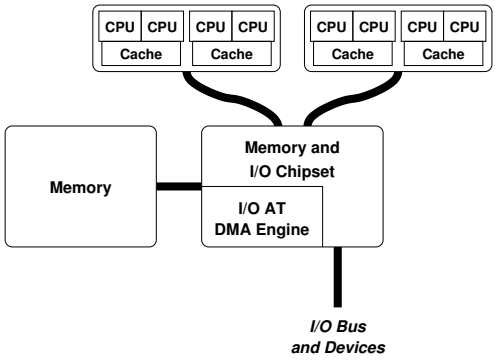


Figure 6: Description of the I/O AT hardware architecture in our dual quad-core XEON hosts.

In addition to improving performance and reducing CPU overhead, offloading memory copies with I/O AT also has the advantage of reducing cache pollution. This is critical for large messages because copying several megabytes can easily pollute the entire cache of the processor when using standard `memcpy` strategies. Indeed, it has been proven that offloaded copy improves concurrent memory operations [43]. However, this cache pollution cannot always be considered as problematic because it may actually preload data that the receiver application will soon read. For this reason, it may actually be interesting to use `memcpy` for small messages, or for the beginning of larger messages, and then switch to I/O AT. However, this would require `memcpy` to occur on a core that shares a cache with the target application, which is as of today hardly predictable.

The I/O AT copy offload mechanism appears as an interesting solution to improve the OPEN-MX receive stack by offloading memory copies between incoming skbuffs and the target data buffers without requiring advanced features from the NIC. We expect it to suit the current OPEN-MX large message receive stack and to actually enable 10G throughput on 10G hardware as planned in objective **O4**. The basic idea is to replace any existing memory copy (`memcpy`) in the OPEN-MX receive path with the submission of an asynchronous copy to the I/O AT kernel subsystem.

Once all submitted copies have been completed, the corresponding OPEN-MX event is deposited in user-space as if `memcpy` were complete. However the I/O AT hardware cannot directly notify the LINUX kernel of the completion of some asynchronous copy requests because the programming model does not offer any

interrupt-based completion callback. Therefore the submitter has to explicitly poll the hardware to determine whether some requests have been completed yet before assuming that the data transfer is finished. In the end, the model becomes interesting as soon as the overhead of submitting offloaded copy requests and checking their completion is faster than the corresponding `memcpy` operation. For this reason, it is important to submit as many chunk copies as possible before waiting for their completion so as to maximize the overlap abilities of the model. The OPEN-MX receive stack only has to wait for copy completion when it must notify the user library of the arrival of some packets or messages:

Small and Medium messages: Each packet is delivered to user-space as an independent event. It thus requires its payload to be copied before the event is delivered. The payload is limited to the MTU. Thus, usually less than 9000 bytes may have to be copied before waiting for copy completion. This case does not offer much room for overlap.

Large messages: A single event is notified to user-space when the whole message has been received. Therefore, copy completions only have to be polled once *all* packets have been received and their copies have been requested, offering greater room for overlap when the message is large.

4.3. Asynchronous Copy Offload of Large Message Receive

The OPEN-MX receiver side processes incoming packets within its specific receive callback, which is invoked by the bottom half interrupt handler of the operating system. The packet headers are first decoded to find the corresponding user endpoint and application. Then the payload is copied into the target buffer. A single event is reported to user-space for large messages when the last packet is received (see Figure 7).

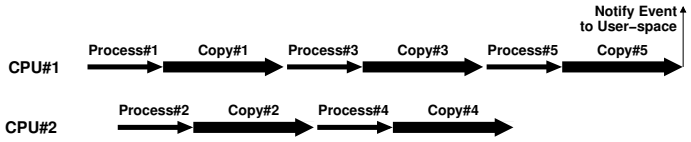


Figure 7: Timeline of the receipt of a 5-packets large message on 2 processors without I/O AT support. The header of each packet is first decoded (*Process#n*) then the payload is copied (*Copy#n*) before releasing the CPU and being able to process another incoming packet. The last packet callback notifies user-space of the completion of the receipt.

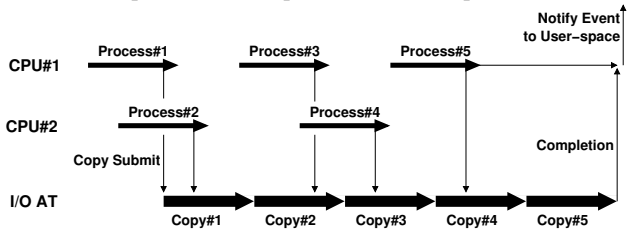


Figure 8: Timeline of the receipt of a 5-packets large message on 2 processors with I/O AT offload of asynchronous copies. The last packet callback waits for the completion of all asynchronous copies before notifying user-space of the receipt completion. All other packet callbacks release the CPU right after processing the packet header (*Process#n*) and submitting the asynchronous copy (*Copy#n*).

The I/O AT DMA engine will therefore be used to receive OPEN-MX large messages as follows: during the processing of any large message packet, the usual `memcpy` is replaced with the submission of the corresponding I/O AT asynchronous copy. If the current packet is the last one, the OPEN-MX callback waits for all previous copies to be completed for this large message. Then, it reports a completion event to user-space as usual. If the current packet is not the last one, then there is nothing to do after submitting asynchronous

copies, the CPU is released immediately. Waiting for all packet copies to be completed is actually quite inexpensive because the I/O AT hardware processes requests in order and reports their completions directly in the host memory. Therefore OPEN-MX simply needs to invoke the I/O AT routine that returns the identifier of the last processed copy request, and compare it with the identifier of the last submitted request.

As shown in Figure 8, this model enables the full overlap of the copies of all but the last packet payload for large messages. Processing packets with I/O AT enabled is actually slightly more expensive due to the need to submit asynchronous copy requests. However, the CPU is released much faster, and the I/O AT hardware performs memory copies faster as explained later in Section 5.3. The overall receiving cost is thus dramatically reduced.

One drawback of the model is the need to free packet descriptors so as to avoid memory starvation in the case of very large messages. However, they cannot be freed before their copy requests are completed The OPEN-MX driver thus periodically has to check the I/O AT progress and cleanup the already processed packet descriptors. Because the MX large message model involves the periodic requesting of new packets to the remote side[1], the cleanup routine is invoked when a new request is sent. This routine is also invoked when the retransmission timeout expires in case of packet loss.

4.4. Performance Tuning

OPEN-MX works on any generic Ethernet hardware and may thus accommodate to usual tuning of NICs for IP workloads. However, because its message passing requirements are often very different from those of general-purpose networking (small message latency is often much more important than equity between flows), some OPEN-MX-specific tuning may improve the performance significantly.

The interrupt-driven model of OPEN-MX receiver side is problematic for the latency. The native MX implementation may switch between interrupts and polling at runtime depending on the application behavior. However, OPEN-MX has to rely on a interrupt-driven model to receive events because incoming packets are only reported to OPEN-MX through a callback from the underlying Ethernet drivers. The faster the interrupt arrives, the lower the latency. Although NAPI [44] may perform some polling that could facilitate reducing the latency, the LINUX networking stack does not let protocols such as OPEN-MX control this polling. Therefore, OPEN-MX may not accommodate the delivery of its incoming packets to its protocol and performance requirements. Moreover, delaying interrupts (*Interrupt Coalescing*) may be enabled to shorten the host overhead [45]. Depending on the application's communication pattern, it may thus be interesting to reduce the NIC interrupt coalescing delay so as to reduce the small message latency, or to increase it, so as to improve the large message throughput.

Generic Ethernet hardware may also vary significantly in their data transfer strategies. For instance, some NICs cannot transfer a packet from/to sparse host memory buffers (*Scatter/Gather*). In this case, OPEN-MX may switch to a mode where all outgoing packet buffers are directly made contiguous. Even if a memory copy is required, it may be optimized by the OPEN-MX driver based on the knowledge of how it was stored in the sender application (objective **O4**).

Then, the OPEN-MX wire protocol may be modified to improve performance. Indeed, the MXoE wire protocol was designed for the MX firmware of Myricom NICs (low latency, high throughput, message passing capabilities, small per-packet host overhead). Its characteristics are very different from those of the generic and potentially-slower Ethernet hardware that OPEN-MX supports. However, the MX wire compatibility is only useful when actually mixing the native MX stack and the OPEN-MX stack on the same fabric.

[1] Four pipelined blocks of eight packets are outstanding for each large message under normal circumstances, see Section 3.1.

Apart from specific systems such as BLUEGENE/P (see Figure 1), OPEN-MX is usually only used to communicate with itself, in which case wire-compatibility is not strictly required. The OPEN-MX wire protocol may thus be changed to better fit the characteristics of Ethernet fabrics.

First, because the per-packet overhead is higher on Ethernet hardware than in the native MX stack, OPEN-MX may use the whole available MTU (*Maximal Transfer Unit*, usually 1500 or 9000 bytes) instead of only power-of-2 packet sizes up-to 4 kB as MX does. Then, because the latency is usually much higher than with MX (from 5 to 100 μs instead of 2 μs), filling the wire between two hosts requires many more packets. As explained in Section 3.1, large messages are transferred as *blocks* of 8 packets. This block size may be increased to 32 in OPEN-MX to better match the network latency.

5. Performance Evaluation

In this section, we introduce the performance of OPEN-MX and compare it with the native MX and TCP implementations. We look at some microbenchmarks and present the improvements offered by several optimization techniques before describing application performance in detail.

5.1. Experimental Platform

The experimental platform is composed of two hosts interconnected without any switches through a commodity 1 gigabit/s (1G) NIC[2] and a Myricom MYRI-10G dual-protocol NIC. The MYRI-10G NIC may either run the native MX stack or a plain 10 gigabit/s (10G) Ethernet stack. A simple firmware/driver reload thus lets us compare OPEN-MX on top of this 10G Ethernet mode and the native MX using the same hardware. Unless specified, jumbo-frames are used (MTU 9000). Each host is composed of 2 quad-core 2.33 GHz INTEL XEON E5345 *Clovertown* processors as depicted in Figure 6.

Both hosts run a 2.6.30 LINUX kernel and OPEN-MX 1.2.1. *Intel MPI Benchmarks* 3.2 (IMB [46]) is used for benchmarking MPI point-to-point and collective operations, and *NAS Parallel Benchmarks* 3.3 (NPB [47]) is used for application-level performance. These MPI applications are built and launched on top of OPEN MPI 1.4.1 using either its *BTL* TCP or *MTL* MX components[3]. Indeed, because OPEN-MX is binary compatible with MX, OPEN MPI may transparently run on top of MX or OPEN-MX.

To achieve reproducible results, OPEN MPI assigned each process to a single core. Thus the system scheduler does not ever try to migrate processes and cause cacheline bounces if a system daemon wakes up, for instance. Additionally, the following tuning options help concentrate on the actual performance and bottlenecks in the network stack in evaluating the microbenchmarks. This configuration will be referred to as *Tuned-mode* in the following sections:

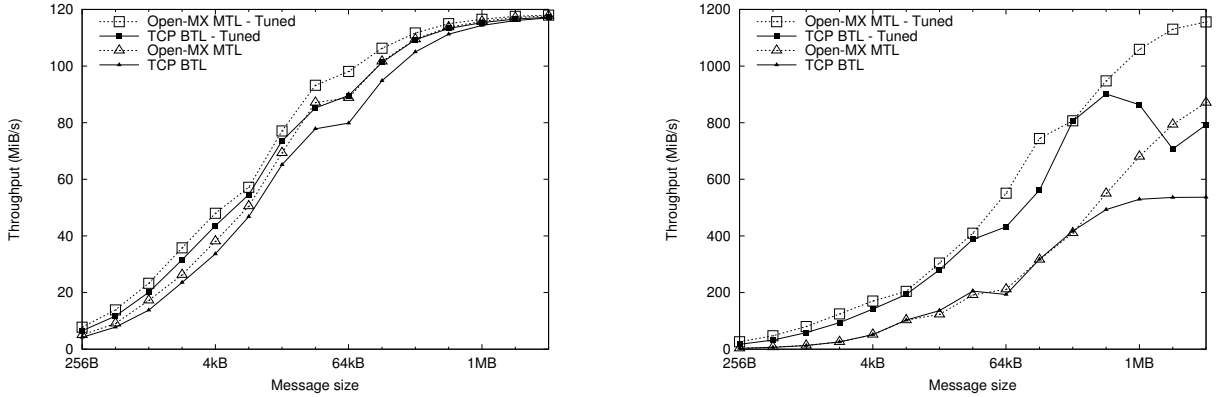
- Interrupts are bound to a single core to avoid the cache-line bounces that usually arise if the operating system lets the interrupt handler runs on any random core.
- Interrupt coalescing is disabled to avoid random latencies due to the NIC deferring the delivery of interrupts (see Section 4.4).
- Power-saving is disabled in the processor so that incoming packets do not wait for any core to wake up before their interrupts are processed.

[2] BROADCOM NETXTREME II BCM5708.

[3] OPEN MPI also offers a *BTL* MX component but it is slower because it does not benefit from MX and OPEN-MX matching capabilities.

5.2. Comparison with TCP

Figures 9(a) and 9(b) present the performance of MPI point-to-point operations measured with the IMB Pingpong tests with 1G and 10G fabrics. In both cases, we compare the performance of Open-MX and TCP with and without the aforementioned tuning of the network stack.



(a) between 1G NICs.

(b) between 10G NICs.

Figure 9: IMB Pingpong throughput.

On the 1G fabric, both Open-MX and TCP achieve the full wire capacity for large messages (megabytes), but Open-MX achieves up to a 10% better throughput when the size is on the order of tens or hundreds of kilobytes. Additionally, the Open-MX latency is 44.6 μs, and TCP's is 50.5 μs. These latencies are mostly related to the high interrupt coalescing delay in the NICs, but the difference is due to Open-MX bypassing the TCP/IP stack to reduce the critical path. Tuning the stack does not improve the throughput significantly because the 1G wire is supposedly the major bottleneck. Both Open-MX and TCP improve by up to 10% thanks to this tuning. More important, because the interrupt coalescing is disabled, it decreases the latency by about 20 μs, down to 24.9 μs for Open-MX and 30.8 μs for TCP.

When using 10G NICs, Open-MX exhibits about 80% higher throughput for large messages. The latency is again 5 μs lower for Open-MX, down to 70 μs. Tuning the network stack improves performance significantly, up to doubling the throughput and reducing the latency to 7.3 μs for Open-MX and 13.2 μs for TCP. Indeed, when a powerful 10G NIC is involved, the host bottlenecks are revealed and careful optimization is required. Fortunately, Open-MX enables the full usage of the 10G wire for large messages.

These results show that Open-MX bypassing TCP/IP and implementing a dedicated stack is an efficient solution because it reduces the host overhead (and thus latency) and because I/O AT copy offload produces better throughput for large messages.

Table 1 summarizes the improvement is offered by Open-MX over TCP on collective operations with IMB. Such communication patterns saturate the 1G fabric so easily that host bottlenecks are negligible. Thus this comparison is mostly relevant for 10G networks. The lower latency of Open-MX improves operations with small messages (or no data at all in *Barrier*) by almost a factor of 2, on average. It also improves most large message operations significantly, by about 50%. It is not clear why *Broadcast* is not improved, although its communication pattern looks similar to *Scatter* for instance.

Open-MX also offers interesting improvements for medium messages (from kilobytes to tens of kilobytes) but the difference is smaller. Indeed these messages do not benefit for any specific advantage of the Open-MX model. The lower latency is mostly beneficial to small messages, and I/O AT copy offload only

Operation	16 bytes	4 kbytes	64 kbytes	4 megabytes
SendRecv	+107%	+4.66%	+8.88%	+32.7%
Exchange	+143%	+8.38%	+24.2%	+29.0%
Allreduce	+87.9%	+2.51%	+11.6%	+4.77%
Reduce	+64.7%	+23.1%	+26.5%	-23.6%
Allgather	+79.2%	-7.1%	+21.5%	+56.0%
Gather	+72.5%	+52.6%	+4.57%	+76.4%
Scatter	+86.3%	+51.6%	+18.4%	+80.9%
Alltoall	+47.0%	+15.2%	+47.2%	+54.7%
Bcast	+44.6%	+20.1%	-26.9%	-39.6%
Barrier	+73.7%			

Table 1: Speedup of IMB collective operations using OPEN-MX instead of TCP. 16 processes were used on 2 nodes with a 10G interface in I/O AT and tuned mode.

helps for large messages.

5.3. Impact of I/O AT Copy Offload

We now look at the actual impact of I/O AT copy offload on the overall OPEN-MX performance. Figure 10 presents a comparison of native MX and OPEN-MX Pingpong with 10G NICs. MX achieves up to 1140 MiB/s for large messages, and OPEN-MX without I/O AT copy offload saturates near 800 MiB/s[4]. For the purpose of performance prediction, if we disable OPEN-MX copies in the receive callback of the OPEN-MX driver, 10G line rate appears to be achievable. This motivated our work on offloading these memory copies on I/O AT hardware so as to achieve line rate performance.

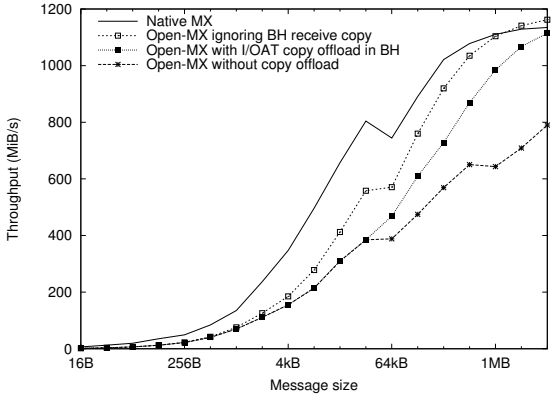


Figure 10: Comparison between MX, OPEN-MX without copy offload support, with the receiver side copy ignored in the bottom half interrupt handler (BH), and with I/O AT copy offload.

I/O AT hardware copies data much faster (2.5 GB/s) than a standard `memcpy` on our machine (1.5 GB/s). However, it costs 350 ns to submit each physically-contiguous memory chunk copy to the I/O AT hardware. In the end, I/O AT is only interesting when the incoming packets are larger than 1 kB. Moreover, each single message is completed once the payload of all its packets has been copied. Given the I/O AT management

[4]The actual data rate of 10 Gbit/s Ethernet is 9953 Mbit/s = 1244 MB/s = 1186 MiB/s.

and completion polling overhead, copy offload is only advantageous when messages contain at least tens of kilobytes. Thanks to such careful tuning, enabling I/O AT copy offload improves OPEN-MX performance dramatically for large messages (larger than 32 kB), now achieving 10G line rate. However, copy offload cannot help for messages smaller than 32 kB because its management overhead is no longer negligible, and also because it is harder to apply to the MX medium message model, as explained in Section 4.2.

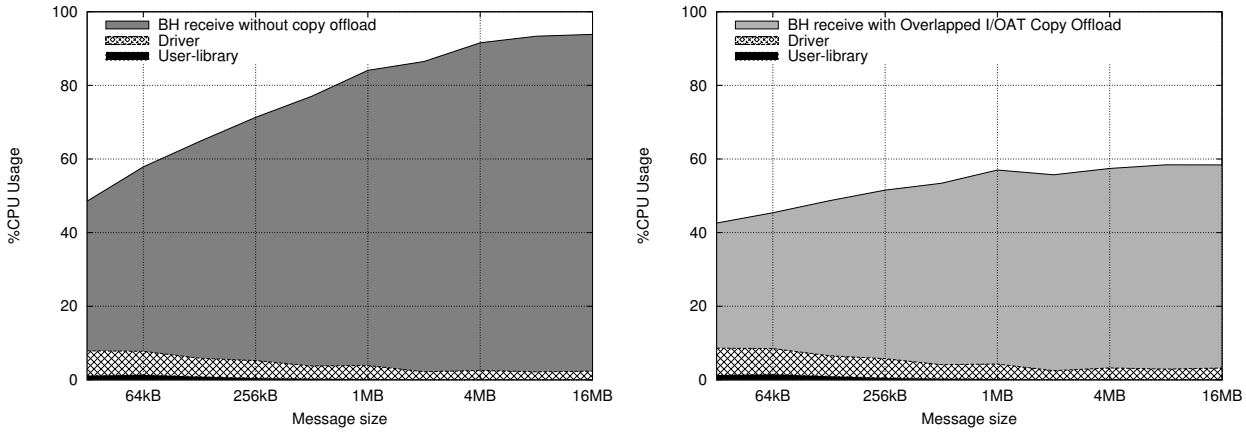


Figure 11: CPU usage of the OPEN-MX library, driver command processing, and bottom half receive processing while receiving a stream of synchronous large messages with and without overlapped asynchronous copy offload.

Figure 11 presents the CPU usage of the OPEN-MX stack receiving a unidirectional stream of large messages. It shows that the standard `memcpy`-based large receive implementation saturates one of the 2.33 GHz XEON cores up-to 95 %. The user-space CPU load is mostly negligible because it only consists of posting requests to the driver and then waiting for a completion event. The driver time is higher because it involves memory pinning during a system call prior to the data transfer. Both the user and driver times do not depend on I/O AT being enabled but the corresponding CPU usage percentages are higher with I/O AT enabled because the overall communication is faster. The actual small CPU availability (in white) comes from the rendezvous handshake before the large data transfer occurs since the processors are idle during the round-trip on a network.

When our I/O AT overlapped copy model is enabled, the overall CPU usage drops from 50 to 42 % for 32 kB messages, and from 95 % to 60 % for multi-megabyte messages. On average, the processor is now available for overlap during 48% of the communication time instead of 28%. Therefore, even if the overall communication time is reduced by about a third, the absolute time for overlap is still increased by about 10% on average. Thus, in addition to significantly improving the OPEN-MX throughput for large messages up to the 10G Ethernet line rate, the I/O AT hardware also dramatically reduces the host load, eliminating the CPU as the bottleneck.

Table 2 compares the improvement produced by I/O AT for TCP and OPEN-MX as a function of the fabric MTU. As explained earlier, I/O AT is only an advantage when large Ethernet packets are used, which makes it almost useless when MTU is 1500. Moreover, I/O AT hardly improves the TCP throughput: it actually only improves CPU availability [42]. The reason why TCP does not benefit from I/O AT as much as OPEN-MX may be that the TCP receive stack cannot overlap the copy submission and processing early unless very large receive requests are posted, which is not the case in the MPI implementation.

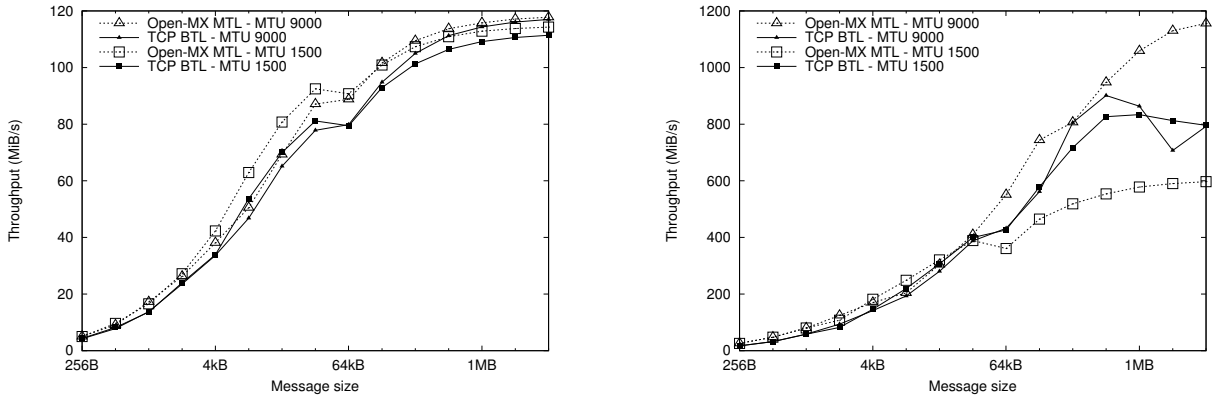
An early performance evaluation on latest INTEL platforms reveals that I/O AT offers a less significant performance improvement. The major reason is that the memory copy throughput improved dramatically

MTU	Stack	4 MB Pingpong Speedup	4 MB Alltoall16 Speedup
9000	OPEN-MX	+24.3%	+10.6%
	TCP	-1.8%	-12.9%
	OPEN-MX tuned	+32.2%	+57.2%
	TCP tuned	0%	0%
1500	OPEN-MX	+10.4%	+7.6%
	TCP	+1.3%	0%
	OPEN-MX tuned	+0.7%	+5.6%
	TCP tuned	0%	0%

Table 2: Influence of I/O AT copy-offload on the throughput of 4 MB Pingpong and 4 MB Alltoall operation between 16 hosts.

in latest INTEL processors (Nehalem and Westmere) and the QPI interconnect is much more scalable than the old centralized front-side bus. In the meantime, the I/O AT hardware did not evolve significantly (its raw throughput increased by only about 20%) and the QPI topology places it further away from memory banks than the processor (NUMA architecture, *Non Uniform Memory Access*). Fortunately, we believe that the DMA engine technology will be integrated into next generation processors. It should make its copy throughput comparable to processor `memcpy`, and thus I/O AT copy offload should offer attractive improvements again.

5.4. Impact of MTU



(a) on 1G fabric in non-tuned mode.

(b) on 10G fabric in I/O AT tuned mode.

Figure 12: IMB Pingpong throughput depending on the MTU.

We now take a deeper look at the impact of packet size, through the MTU, on the performance of OPEN-MX and TCP. Figure 12(a) shows the throughput of a IMB Pingpong on our 1G network in the non-tuned case. Although OPEN-MX is always a bit faster than TCP, larger packets only bring a 3 % improvement for large messages. This limited improvement is related to the fact than 1G networks do not cause high packet rate and thus do not overload the host with packet processing when small packets are used. The smaller MTU is even faster for medium messages. This unexpected result is likely a result of from NIC vendors optimizing the hardware for the most common use, which today is still MTU 1500.

Figure 12(b) compares the throughput on our 10G network in the tuned case. TCP does not show a large performance difference because the NIC is able to segment and reassemble small packets in hardware (TSO), causing the host to manipulate large packets independently of the actual physical MTU underneath. OPEN-MX does not benefit from such hardware features and is thus improved significantly by a large MTU, by a factor of 2. Fortunately, OPEN-MX compensates for these missing hardware features with a stack that was designed for message passing and that uses I/O AT copy offload, achieving up to 50 % better throughput than TCP for large messages.

5.5. MX Wire-Compatibility

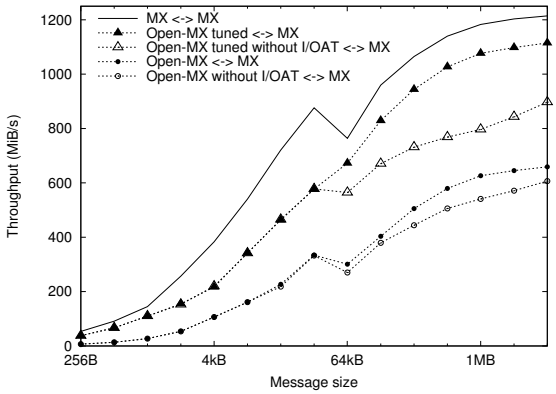
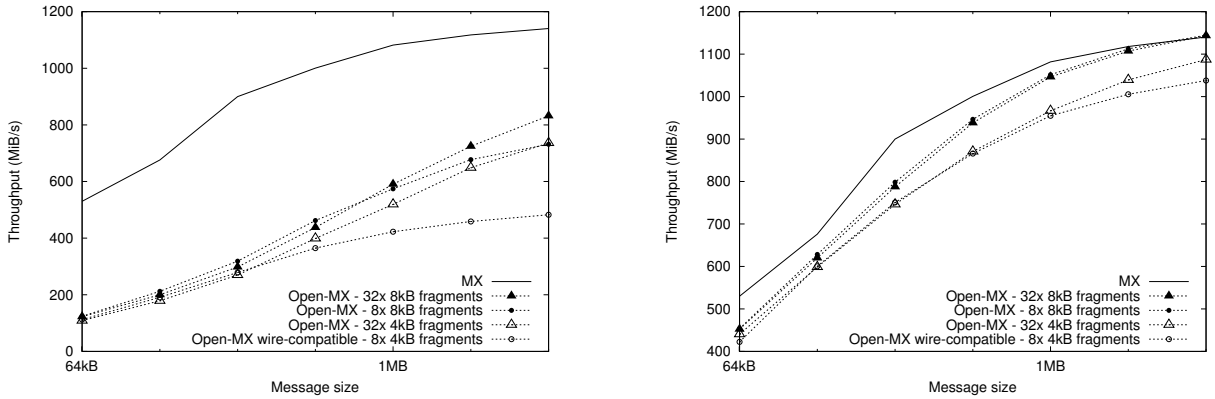


Figure 13: Performance of the OPEN-MX Wire-Compatibility with MXoE. A native MXoE host talks to a host running the wire-compatible OPEN-MX stack on a MYRI-10G NIC in native Ethernet mode.

Previous experiments have shown that a large MTU and offloading memory copies on I/O AT hardware are two important features that help OPEN-MX achieve interesting performance. Because OPEN-MX is wire-compatible with the native MX stack, we now look at the impact of OPEN-MX on MX performance. One of our host now runs the native MX stack on its MYRI-10G NIC, and the other still runs the Ethernet stack with OPEN-MX. As expected, the latency is the average of the latencies of MX and OPEN-MX, 4.6 μs in tuned mode, 38 μs otherwise.

Figure 13 presents the throughput of IMB Pingpong. It shows that OPEN-MX can approach the performance of the native MX, down-to at least 65% of its throughput, and even 90% for large messages. However, as expected, it also shows that tuning the network stack and offloading memory copies on I/O AT hardware is critical to obtaining the highest performance.

One major constraint with the MX wire-compatibility is that it enforces the use of a wire protocol that was designed for dedicated hardware, and OPEN-MX usually runs on slower hardware. We now look at the impact of breaking the wire-compatibility when it is not strictly required, as described in Section 3.2. Figures 14(a) and 14(b) present the performance improvements enabled by using larger packets and high numbers of packets in each block of large messages. As explained in Section 5.4, the use of larger packets improve performance significantly in all cases because it reduces the number of packets to be processed by the OPEN-MX stack. Increasing the block size helps dramatically in the non-tuned mode. Indeed, the latency is high (about 70 μs) and thus requires a lot of in-flight packets to fill the wire between the hosts. The tuned case is not as significantly improved because the latency is much slower and thus does not require a large block size. However, both optimizations offer some interesting improvement and justify the idea of breaking the MXoE wire compatibility when it is not useful.



(a) non-tuned I/O AT mode.

(b) tuned I/O AT mode.

Figure 14: Impact of larger packets and deeper pipeline on the OPEN-MX throughput.

Benchmark	TCP	OPEN-MX	MX
bt.C.16	272.37	269.07	266.25
cg.C.16	90.79	89.75	84.76
ep.C.16	30.40	30.66	30.28
ft.B.16[5]	21.59	20.48	19.16
is.C.16	5.84	*5.62*	4.57
lu.C.16	205.44	202.39	199.33
mg.C.16	44.10	43.59	43.44
sp.C.16	556.53	547.13	541.71

Table 3: NAS Parallel Benchmarks execution time, in seconds. Class C was used with 16 processes on 2 nodes.

5.6. Applications

Table 3 presents the absolute execution time of the *NAS Parallel Benchmarks* over the native MX and TCP stacks, as well as OPEN-MX. MX demonstrates 6.2% of speedup over TCP on average. Given that most of these applications are not communication intensive [47], this improvement is attractive. As expected, the communication intensive IS improved the most (+21.7%) while EP is almost unchanged because it uses almost no communication.

OPEN-MX improves the execution time of all NAS benchmarks by 1.8% on average over TCP. It is only a third of the speedup that MX brings over TCP, but OPEN-MX does not require dedicated message passing capabilities in the NIC to do so. IS actually required the careful binding of processes and interrupts because this application involves many large messages and collective operations. They cause contention in the memory bus and caches that may increase the execution time by up to a factor of 4 under some unclear circumstances. Indeed, having the interrupt handlers process incoming packets on all cores and also pass them to target applications running on all cores may induce many cache-line bounces between cores. A more suitable way to address this issue is to add *Multiqueue* support to the stack so that packets are processed on the core that runs their target application, as explained in [48].

[5]FT class C could not complete because it uses more memory than available on our test machines.

6. Conclusion

The spreading of clusters in high-performance computing in the last decade has led to the development of specialized high-speed networks such as InfiniBand or Myrinet. However Ethernet is still the most popular technology and many research projects try to offer high-performance message passing on top of it. Open-MX is an open-source networking stack that applies advanced software innovations such as I/O AT copy offload to circumvent the limitations of the Ethernet model. It also benefits from its interface compatibility with the popular MX stack and thus works transparently with many existing MPI implementations. Open-MX is available for download at `http://open-mx.org`.

In this article, we presented an in-depth study of the implementation and performance of Open-MX. Its overlapped copy offload model enables high throughput and reduces CPU utilization, while its careful tuning of the interrupt coalescing significantly decreases the latency. Open-MX surpasses the performance of TCP-based MPI implementations, especially on 10G networks where it reaches a latency of 7 μs for small messages and the 10G line rate for large messages. The overall execution time of the NAS Parallel Benchmarks is reduced by 2 % on average.

Although it is slower than the specialized MX stack on the same hardware, Open-MX shows that applying high-performance networking ideas to commodity Ethernet hardware can improve performance. It also demonstrates that it is not necessary to rewrite a custom application interface dedicated to Ethernet because the MX interface was successfully mapped on top of Open-MX. On the other hand, the wire protocol specification of MX had to be modified to better match the capabilities of the fabric, especially its higher latency and larger MTU.

Moreover, our work also raises the question of which dedicated features should be added to Ethernet NIC to facilitate message passing. For instance, our results show that TCP implementations suffer less from smaller MTUs because they benefit from simple hardware features such as segmentation offload (TSO) which reduces the host overhead. Open-MX cannot benefit from such features. The possible convergence between specialized HPC networks and commodity Ethernet will certainly have to address this question.

References

[1] Top500 Supercomputing Sites, `http://top500.org`.

[2] Message Passing Interface Forum, MPI: A message-passing interface standard, Tech. Rep. UT-CS-94-230 (1994).

[3] Infiniband architecture specifications, InfiniBand Trade Association, `http://www.infinibandta.org` (2001).

[4] Myricom Inc., `http://www.myri.com`.

[5] FCoE (Fibre Channel over Ethernet), `http://www.fcoe.com`.

[6] Coraid: The Linux Storage People, `http://www.coraid.com`.

[7] G. Ciaccio, G. Chiola, GAMMA and MPI/GAMMA on gigabitethernet, in: Proceedings of 7th EuroPVM-MPI conference, Balatonfured, Hongrie, 2000, pp. 129–136.

[8] S. Sumimoto, K. Kumon, PM/Ehernet-kRMA: A High Performance Remote Memory Access Facility Using Multiple Gigabit Ethernet Cards, in: 3rd International Symposium on Cluster Computing and the Grid (CCGrid2003), IEEE, 2003, pp. 326–334.

[9] J. Chen, W. Watson III, R. Edwards, W. Mao, Message Passing for Linux Clusters with Gigabit Ethernet Mesh Connections, in: IPDPS'05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 9, Denver, CO, 2005.

[10] S. Sumimoto, K. Ooe, K. Kumon, T. Boku, M. Sato, A. Ukawa, A Scalable Communication Layer for Multi-Dimensional Hyper Crossbar Network Using Multiple Gigabit Ethernet, in: ICS'06: Proceedings of the 20th International Conference on Supercomputing, Cairns, Australia, 2006, pp. 107–115.

[11] S. Karlsson, S. Passas, G. Kotsis2, A. Bilas, MultiEdge: An Edge-based Communication Subsystem for Scalable Commodity Servers, in: Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS'07), Long Beach, CA, 2007, p. 28.

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

[13] 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, 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, 2004, pp. 97–104.

[14] D. Buntinas, G. Mercier, W. Gropp, Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem, Parallel Computing, Selected Papers from EuroPVM/MPI 2006 33 (9) (2007) 634–644.

[15] The Parallel Virtual File System, version 2, `http://www.pvfs.org`.

[16] 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, IEEE Computer Society Press, Miami, FL, 2008.

[17] B. Goglin, Improving Message Passing over Ethernet with I/OAT Copy Offload in Open-MX, in: Proceedings of the IEEE International Conference on Cluster Computing, IEEE Computer Society Press, Tsukuba, Japan, 2008, pp. 223–231.

[18] T. E. Anderson, D. E. Culler, D. A. Patterson, A Case for NOW (Networks of Workstations, IEEE Micro 15 (1) (1995) 54–64.

[19] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, C. V. Packer, BEOWULF: A parallel workstation for scientific computation, in: Proceedings of the 24th International Conference on Parallel Processing, Oconomowoc, WI, 1995, pp. I:11–14.

[20] R. P. Martin, A. M. Vahdat, D. E. Culler, T. E. Anderson, Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture, in: Proceedings of the 24th Annual International Symposium on Computer Architecture, Denver, CO, 1997, pp. 85–97.

[21] D. D. Clark, V. Jacobson, J. Romkey, H. Salwen, An Analysis of TCP Processing Overhead, IEEE Communications Magazine 27 (1989) 23–29.

[22] A. Barak, I. Gilderman, I. Metrik, Performance of the Communication Layers of TCP/IP with the Myrinet Gigabit LAN, Computer Communication 22 (11).

[23] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser, Active Messages: a Mechanism for Integrated Communication and Computation, in: Proceedings of the 19th Int'l Symp. on Computer Architecture, Gold Coast, Australia, 1992.

[24] S. Pakin, V. Karamcheti, A. A. Chien, Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors, IEEE Concurrency 5 (1997) 60–73.

[25] Z. Yi, P. P. Waskiewicz, Enabling Linux Network Support of Hardware Multiqueue Devices, in: Proceedings of the Linux Symposium (OLS2007), Ottawa, Canada, 2007, pp. 305–310.

[26] L. Grossman, Large Receive Offload Implementation in Neterion 10GbE Ethernet Driver, in: Proceedings of the Linux Symposium (OLS2005), Ottawa, Canada, 2005, pp. 195–200.

[27] Linux Foundation, Net:TOE, `http://www.linuxfoundation.org/en/Net:TOE` (2006).

[28] Myricom Myri-10G, `http://www.myri.com/Myri-10G/`.

[29] Mellanox ConnectX - 4th Generation Server & Storage Adapter Architecture, `http://mellanox.com/products/connectx_architecture.php`.

[30] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, P. Grun, Remote Direct Memory Access over the Converged Enhanced Ethernet Fabric: Evaluating the Options, in: Proceedings of the 17th Annual Symposium on High-Performance Interconnects (HotI'09), New York, NJ, 2009, pp. 123–130.

[31] P. Shivam, P. Wyckoff, D. K. Panda, EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing, in: Proceeding of Supercomputing ACM/IEEE 2001 Conference, Denver, CO, 2001, p. 57.

[32] ParTec Cluster Competence Center, `http://www.par-tec.com`.

[33] M. J. Rashti, 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, 2007, p. 234.

[34] D. Dalessandro, A. Devulapalli, P. Wyckoff, Design and Implementation of the iWarp Protocol in Software, in: Proceedings of PDCS'05, Phoenix, AZ, 2005, pp. 471–476.

[35] The IBM Blue Gene team, Overview of the IBM Blue Gene/P project, IBM Journal of Research and Development 52 (1/2), `http://www.research.ibm.com/journal/rd/521/team.html`.

[36] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, R. Thakur, Toward Efficient Support for Multithreaded MPI Communication, in: Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag, 2008, pp. 120–129.

[37] B. Goglin, High Throughput Intra-Node MPI Communication with Open-MX, in: Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2009), IEEE Computer Society Press, Weimar, Germany, 2009, pp. 173–180.

[38] H. K. J. Chu, Zero-Copy TCP in Solaris, in: Proceedings of the USENIX Annual Technical Conference, San Diego, CA, 1996, pp. 253–264.

[39] S. Passas, K. Magoutis, A. Bilas, Towards 100 Gbit/s Ethernet: Multicore-based Parallel Communication Protocol Design, in: Proceedings of the 23rd international conference on Supercomputing (ICS'09), ACM/SIGARCH, Yorktown Heights, NY, 2009, pp. 214–224.

[40] G. Regnier, S. Makineni, I. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, A. Foong, TCP Onloading for Data Center Servers, Computer 37 (11) (2004) 48–58.

[41] A. Grover, C. Leech, Accelerating Network Receive Processing (Intel I/O Acceleration Technology), in: Proceedings of the Linux Symposium (OLS2005), Ottawa, Canada, 2005, pp. 281–288.

[42] K. Vaidyanathan, D. K. Panda, Benefits of I/O Acceleration Technology (I/OAT) in Clusters, in: Proceedings of the International Symposium on Performance Analysis of Systems and Software, San Jose, CA, 2007, pp. 220–229.

[43] K. Vaidyanathan, W. Huang, L. Chai, 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, Long Beach, CA, 2007, p. 234.

[44] J. H. Salim, R. Olsson, A. Kuznetsov, Beyond softnet, in: Proceedings of the 5th annual Linux Showcase & Conference, Oakland, CA, 2001.

[45] K. Salah, To Coalesce or Not To Coalesce, International Journal of Electronics and Communications 61 (2007) 215–225.

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

[47] 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, S. K. Weeratunga, The NAS Parallel Benchmarks, The International Journal of Supercomputer Applications 5 (3) (1991) 63–73.

[48] B. Goglin, NIC-assisted Cache-Efficient Receive Stack for Message Passing over Ethernet, in: Proceedings of the 15th International Euro-Par Conference, Lecture Notes in Computer Science, Vol. 5704 of Lecture Notes in Computer Science, Springer, Delft, The Netherlands, 2009, pp. 1065–1077.