Decoupling Memory Pinning from the Application with Overlapped on-Demand Pinning and MMU Notifiers

Brice Goglin

Decoupling Memory Pinning from the Application with Overlapped on-Demand Pinning and MMU Notifiers

Brice Goglin
INRIA Bordeaux - Sud-Ouest – France
Brice.Goglin@inria.fr

Abstract

High-performance cluster networks achieve very high throughput thanks to zero-copy techniques that require pinning of application buffers in physical memory. The Open-MX stack implements message passing over generic Ethernet hardware with similar needs.

We present the design of an innovative pinning model in Open-MX based on the decoupling of memory pinning from the application. This idea eases the implementation of a reliable pinning cache in the kernel and enables full overlap of pinning with communication. Performance evaluation shows that both these optimizations bring interesting throughput improvements.

1 Introduction

Since the emergence of clusters fifteen years ago, networking in HPC has been dominated by high-speed technologies such as INFINIBAND [8]. These technologies brought significant performance improvement thanks to dedicated software stacks and hardware features. Zero-copy is one of the most well-known features. It enables higher throughput and lower CPU overhead, but it also has severe constraints over the software. Indeed, all memory buffers that are manipulated by the network interface (NIC) have to be pinned down in physical memory during communication. This memory pinning is known to be very expensive [9]. It has been the subject of many research projects which led to the idea of caching it [15]. It reduces the observed overhead if the applications reuse the same buffers several times, but the overhead often remains high.

While high-speed networks have been dominating in high-performance clusters, there are still many installations that rely on traditional ETHERNET hardware. The MPI standard has been successfully implemented on top of these technologies but its performance is limited by the TCP/IP stack which was not designed for this context. Moreover, a convergence between high-speed networks and ETHERNET is expected in the coming years. It raises the question of which features will become legacy.

Therefore, many projects targeted the design of a high-performance MPI on top of generic ETHERNET hardware, such as GAMMA [2], EMP [13], or PM/ETHERNET-HXB [14]. OPEN-MX [4] offers such an implementation. It exposes the *Myrinet Express* [10] wire protocol and application interface by implementing the corresponding software features on top of the generic ETHERNET layer. While using standard hardware, OPEN-MX has to rely on memory pinning since it tries to mimic the zero-copy model of high-speed networks. However, its pinning requirements are simpler due to less hardware being involved. We thus envision new innovative optimizations based on the idea that pinning has to be managed by the OPEN-MX driver and does not need to be exhibited to user-space or to the hardware.

The paper is organized as follows. We first describe in Section 2 how memory pinning is involved in I/O, and high-speed networking in particular. We then detail the OPEN-MX stack and why we chose to optimize its pinning. The design of our decoupled on-demand and overlapped memory pinning is then presented in Section 3. Performance is then detailed in Section 4 while related works are discussed in Section 5.

2 Background and Motivations

In this Section, we first describe how memory pinning is involved in high-speed networks. We then introduce the OPEN-MX stack before detailing our motivation to improve its usage on memory pinning.

2.1 Memory Pinning in High-Speed Networks

Memory pinning is involved as soon as a hardware device has to manipulate the host memory. Indeed, the hardware usually reads/writes from/to host memory by DMA (*Direct Memory Access*). This mechanism involves physical memory addresses, while the applications only manipulate virtual addresses. Due to the operating system ability to swap pages out to the disk, or to migrate them in physical memory, there is usually no guarantee that a given memory buffer stays at the same physical location.

To enforce the physical location, the operating system or driver pins buffers down in physical memory so that no freeing, swapping or migration may occur. This strategy is used for many DMA-based I/Os, such as disk access, networking, or hardware copy offload with the I/OAT technology (*I/O Acceleration Technology* [7]). High-speed networking extends this model further by enabling permanent storage in the NIC of translations from virtual into physical addresses (*Memory Registration*). Once memory buffers are pinned and registered, the application may submit communication requests from user-space and let the NIC process them in the background with zero-copy.

However, these pinning and registration have an important overhead, which depends a lot on the hardware and software implementation. Registration may cost up to 100 μs on INFINIBAND [9] since the processor has to write translation to the NIC. Deregistration may also reach 200 μs on MYRINET/GM [5] because of translation synchronization between the NIC and the operating systems. Some other implementations such as MYRINET/MX let the NIC read translations from the host by DMA on demand, causing the host overhead to be much lower. In this case, memory pinning is the main overhead, usually several microseconds plus a few hundreds nanoseconds per page.

This high cost led to many research works on optimizing memory registration. The most commonly used solution is the *Pin-down Cache* [15] (or *Registration Cache*) which keeps buffers registered as long as possible so that they may be reused without being registered again. This strategy works well when the application reuses the same buffers again and again, but it raises strong technical difficulties due to the need to keep the NIC translation table synchronized with the host. Indeed, even if pages are pinned in memory, the application may free buffers or cause the operating system to duplicate a page on *Copy-on-write*. In these cases, some NIC table entries must be invalidated.

This invalidation requires the driver to be notified when such address space modifications occur. It is usually implemented in user-space by intercepting some symbols such as munmap or free. However it means that the program should be dynamically linked and should not use its own malloc implementation. It also sometimes causes conflicts with middlewares that intercepts some symbols for other reasons. Therefore, user-space notification is sometimes considered as unreliable.

Another solution consists in having the kernel notifying the driver of address space invalidation. This support has finally been introduced in LINUX kernel 2.6.27 as *MMU Notifiers* [1] to help the KVM virtualization subsystem maintaining the guest OS shadow page tables. However, in HPC, the user-space library is usually responsible for deciding which buffer has to be pinned or unpinned. So having the kernel notifying the invalidation requires a strong synchronization with user-space [16], and makes the model difficult to implement.

2.2 The Open-MX Stack

OPEN-MX [4] is an implementation of the wire protocol and user-space interface of MXOE (*Myrinet Express over Ethernet*). It works on the generic ETHERNET layer of the LINUX kernel instead of requiring MYRICOM's specific hardware. It mimics the MX features by deporting MYRICOM NIC hardware features in a driver as presented on Figure 1. Moreover, OPEN-MX is able to improve the receive performance by offloading memory copy on INTEL *I/O Acceleration Architecture* (I/OAT DMA engine).

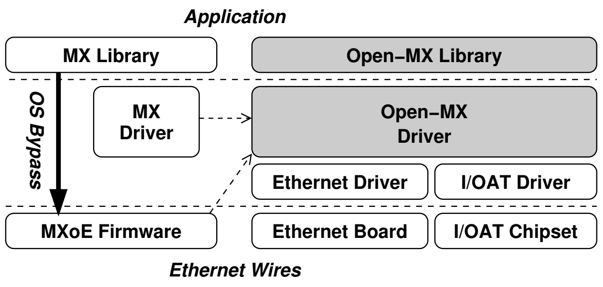

Figure 1. Design of the native MX and generic Open-MX software stacks.

The OPEN-MX stack involves memory pinning of the application pages as high-speed network do. On the send side, OPEN-MX may pass user-space pages to the ETHERNET NIC for sending. As any usual I/O, pinning is required until the send request is completed. On the receive side, the model is very different from high-speed networks. ETHERNET NICs and drivers cannot let the receiving application choose the memory location where each incoming packet should be stored. This strong limitation leads to copying each incoming OPEN-MX packet into its final application buffer. Offloading this copy on I/OAT DMA engine helps performance but it also means that application buffers are passed to a hardware device. This is one reason why OPEN-MX may have to pin user-space pages on the receive side even if no zero-copy receive is performed.

Additionally, due to the MXOE wire protocol, the OPEN-MX stack is mostly interrupt driven. Many incoming packets are not processed in the context of the target process. The target page table is thus unavailable to regular virtual copy routines. The OPEN-MX stack thus pins large receive buffers so that the receiving routine knows the physical memory location where data should be copied[1]. These buffers that may have to be pinned are called *User Regions*.

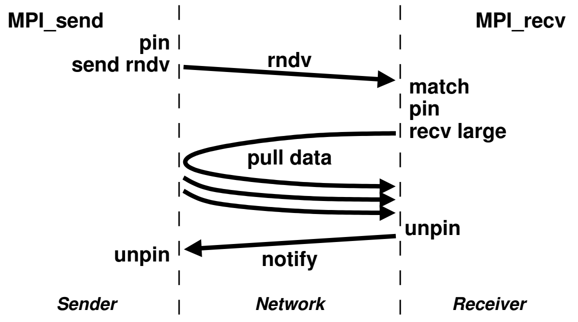

Figure 2. Timeline of a MPI large message transfer over Open-MX.

User regions are actually only involved in large message transfers after a *rendezvous*. As defined in the MXOE specification, all messages smaller than 32 kB may be sent eagerly. In this case, OPEN-MX usually relies on a statically pinned intermediate buffer [4]. For larger messages, the sender first pins its buffer and then sends a *rendezvous* message, as presented in Figure 2. When matching this *rendezvous*, the receiver pins its receive buffer and starts pulling data from the sender's buffer. Once done, a notify message is sent by the receiver so that all resources are released and the request completes. In this model, all buffer accesses are driven by incoming packets. On the sender side, when a *pull* packet is received,

[1]Once the physical address is known and guaranteed to remain the same, the kernel may remap it at a temporary virtual location and use a regular `memcpy`.

data is read from the send region and attached to *pull reply* packets that are sent to the receiver. On the receiver side, when a *pull reply* is received, the data is copied into the receive region. This model, driven by the MXoE specifications, explains why large buffers are accessed outside of the application context, and thus require memory pinning, even when I/OAT copy offload is not used.

2.3 Opportunities for Optimizing Pinning

While requiring memory pinning, the OPEN-MX stack has one important difference with high-speed networks: it does not have to register virtual to physical address translations in the NIC. This fact makes the OPEN-MX design much simpler since no synchronization is needed between the driver that pins/unpins pages and the component that uses them for sending/receiving. From this fact, we derive the following ideas. First, only the driver has to be aware of pages being pinned or not. User-space only needs to know which memory regions are being used by communication requests. Second, there is no need to pin an entire region if only part of it is going to be used in the near future.

Thanks to these ideas, we propose a innovative rework of the memory pinning model to reduce its observed overhead significantly:
$\rightarrow$ Decouple the actual memory pinning from region declaration. The driver may pin, unpin, and repin when needed, without having to ever synchronize with user-space.
$\rightarrow$ Overlap this on-demand pinning with communication since there is not need to wait for an entire region to be pinned before starting to use the first pinned pages.
$\rightarrow$ Rely on *MMU Notifiers* as a now widely supported and reliable solution for detecting memory regions that are being invalidated and thus need to be unpinned.

3 Design of a Decoupled Memory Pinning Framework

In this Section, we describe the design and the implementation of the aforementioned ideas in the OPEN-MX stack.

3.1 On-Demand Pinning and Invalidation

The idea behind *On-Demand* pinning consists in the ability to have a declared user-space region in the driver without having it pinned all the time. It enables unpinning when needed without having to undeclare it, which means there is no need to notify user-space of memory invalidation. Thanks to *MMU Notifiers*, our model has the advantage of being implemented in the kernel and thus providing a safe and reliable invalidation detection. It first means that there is no need to intercept some user-space symbols as usual registration caches do. Secondly, since the whole pinning management remains in the kernel, there is also no need to have the in-kernel invalidation detection code synchronize with the user-space code that usually manages the pinning and unpinning of regions [16].

Since a declared region may be pinned or not, the driver has to be aware of when it actually needs to pin it. Fortunately, the OPEN-MX stack is not *OS-Bypass* since it uses the kernel ETHERNET drivers directly. So, each communication request enters the kernel, makes sure the involved regions are pinned, and then accesses the ETHERNET layer. An example of execution is described in Figure 3. If the application ever gives an invalid region segment, its declaration succeeds, but pinning fails later at communication time and the corresponding request aborts with an error.

OPEN-MX uses *MMU Notifiers* by attaching a notifier to the process address space when an OPEN-MX endpoint is open. This notifier structure provides callbacks that are invoked when memory regions are invalidated within the address space. OPEN-MX may thus unpin an invalidated region when the application

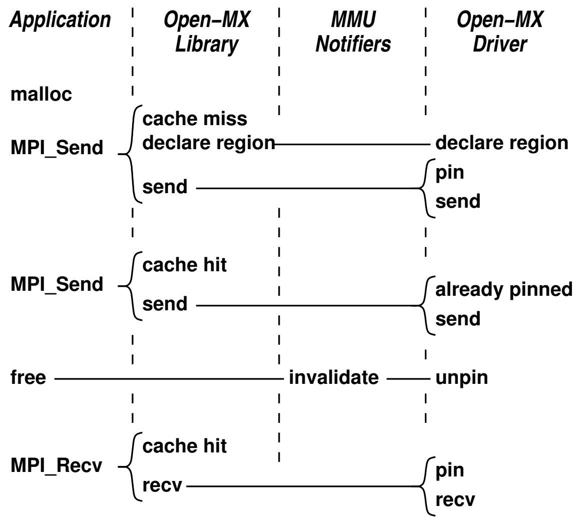

Figure 3. Timeline of a MPI execution on top of the on-demand decoupled pinning with region cache in Open-MX.

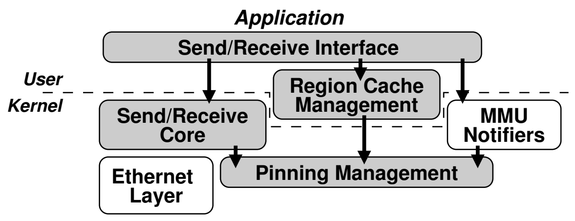

Figure 4. Software components and interactions in the region management and pinning stack.

frees it and possibly repin it later since the same buffer may be reallocated. Additionally, if there are too many pinned pages (making the system unable to perform correctly), it may also requests some unpinning. In this case, the region remains valid and will be repinned next time a communication request uses it.

3.2 Managing the Cache of Declared Regions

The set of declared regions is managed by the OPEN-MX stack above the pinning routines as described in Figure 4. It takes care of caching user regions so that there is no need to redeclare them again to the driver. However, when the number of regions becomes too high, the least recently used ones are undeclared (usual LRU algorithm).

This code has to located between the main user-space library (which manages communication requests, retransmission, ...) and the driver (which manages region pinning and the underlying ETHERNET drivers). Moving code into the kernel is generally considered a bad idea unless strictly necessary. The cache policy and region declaration code can be implemented and tuned in user-space without any overhead or code complexity. So we decided to implement the region cache management in user-space (Figure 4). There are actually other reasons supporting this choice. As explained earlier, the region management code does not need to be notified of region invalidation/unpinning. So moving it to user-space does not involve a synchronization mechanism between the driver and the user-space library. Moreover, the region cache management code takes care of translating a set of user memory segments (regions may be vectorial) into an integer descriptor that the driver may manipulate. Keeping the cache management in user-space makes each communication request system call only manipulate such an integer. The whole set of region segments is only passed to the kernel once at declaration.

3.3 Overlapped Memory Pinning

Now that the driver has the ability to pin user regions on demand, we are looking at overlapping this pinning with communication. As seen on Figure 3, if not already pinned, a region is pinned by the driver before the associated send/receive is passed to the underlying ETHERNET driver. Since the data transfer occurs in order and there is a round-trip on the network[2] before it actually begins, the OPEN-MX driver core is expected to have already pinned enough pages. Then, since memory pinning is faster than the data transfer (see Section 4.1), the following accesses to the region

[2]Usually 10-20 μs with OPEN-MX.

also occur after the target pages were pinned. The OPEN-MX communication model with overlapped pinning is presented in Figure 5. It moves the low-level driver communication request before the pinning. OPEN-MX pinned region accessors were thus modified to allow read/write in the pinned area if the whole region is not pinned yet.

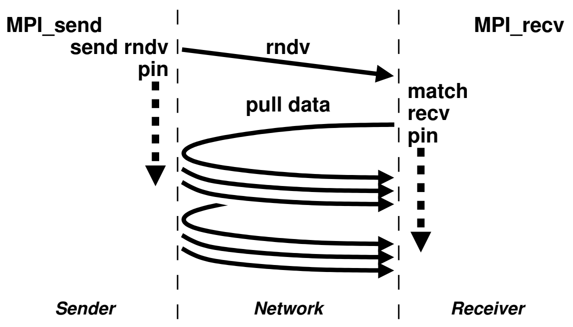

Figure 5. Timeline of a MPI large message transfer over Open-MX when memory pinning is overlapped with communication.

Even if pinning is expected to be much faster than the initiating handshake round-trip, it is still possible that the pinning processor is overloaded and cannot pin fast enough. One solution consists in delaying the packet processing until enough pages are pinned and then actually read/write into the region. But it implies a relatively high code complexity and overhead. So we have chosen another solution: drop the incoming packet and let retransmission happen later when the remote side will assume that the packet has been lost. Even if the performance will be worse, it would not have been optimal anyway since the processor was too busy to process the communication request immediately. We expect that this easy solution will provide an efficient enough workaround for this rare case.

4 Performance Evaluation

In this Section, we first look at the overhead of memory pinning to justify our work on optimizing it. We then evaluate the performance improvements brought by our optimizations.

4.1 Open-MX Pinning Overhead

Memory registration is known to be very expensive in high-speed networks due to the need to store address translations in the NIC [9]. The OPEN-MX model is simpler since only pinning is involved. Table 1 presents the corresponding overhead. It appears to be somehow related to the processor speed since both the base overhead and per-page overhead decrease with faster CPUs. Given this result, we expected that memory pinning decreases the observed communication throughput on a 10G ETHERNET network by up to 20 % on slow processors.

Processor	GHz	Base μs	ns/page	GB/s
OPTERON 265	1.8	4.2	720	5.5
OPTERON 8347	1.9	2.2	330	12
XEON E5435	2.33	2.3	250	16
XEON E5460	3.16	1.3	150	26.5

Table 1. Base and per-page overhead of the Open-MX pinning+unpinning, and the corresponding pinning throughput.

All following experiments were performed on the above XEON E5460 host with a MYRI-10G ETHERNET interface. OPEN-MX was used within OPEN MPI, on top of LINUX kernel 2.6.27 with *MMU Notifiers* enabled. From Table 1, we expect a 5-10 % performance degradation due to memory pinning on this machine.

Figure 6 presents the actual impact of memory pinning on MPI communication throughput. We observe a 5 % difference on this fast machine when switching from pinning once per communication to permanently pinning each buffer. This impact increases up to 20 % on slower machines and justifies our work on optimizing memory pinning.

4.2 Impact of Decoupled and Overlapped Memory Pinning

Figure 7 presents the impact of our decoupled pinning model on the MPI throughput. It shows that both overlapped pinning and pinning cache bring the expected 5 % performance

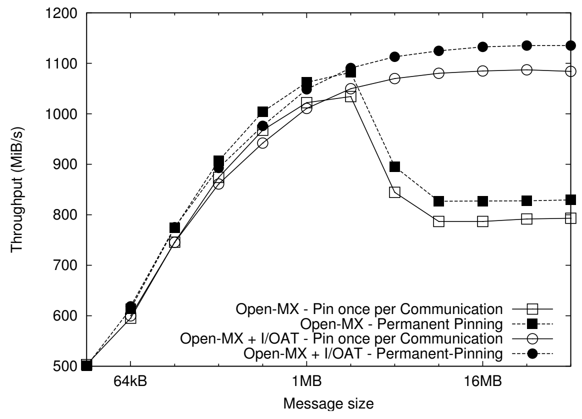

Figure 6. IMB Pingpong throughput on top of Open-MX depending on the pinning cache being enabled.

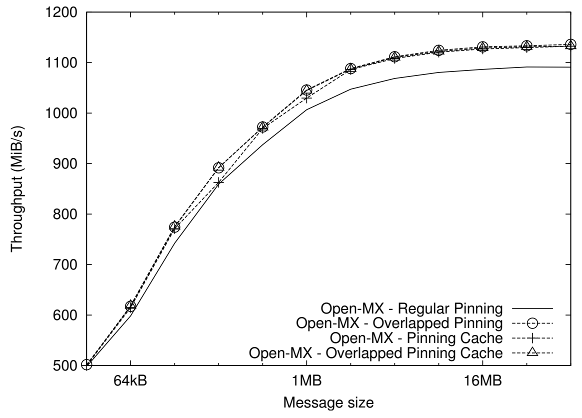

Figure 7. Impact of overlapped pinning and pinning cache on IMB Pingpong throughput on top of Open-MX.

improvement by hiding or removing the overhead of memory pinning. On slower machines, we observed a similar behavior with up to 18 % performance improvement.

It first shows that our pinning cache implementation performs as expected since the observed performance is similar to the permanent pinning experiment presented in the previous section. Second, if the application cannot benefit from the pinning cache (for instance if it does not reuse the same buffer multiple times), the same performance improvement is brought by overlapped memory pinning. Our overlap thus appears as an interesting optimization when the pinning cache cannot help.

The overhead of the overlapped pinning is very small since it only involves some additional tests on the region descriptor when an incoming packet is processed. The overhead of the pinning cache is higher since it involves looking up a region in the user-space cache and checking whether it is already pinned in the driver. But it also remains negligible against the transfer time of large messages.

4.3 Overlap Impact Discussion

Overlapping memory pinning is expected to work because pinning is much faster than the actual network data transfer (see Section 4.1). However, if the CPU is severely overloaded, it may still not pin pages fast enough. We added some counters to the OPEN-MX code to evaluate the probability of an *Overlap-Miss*. Under regular OPEN-MX load[3], less than 1 packet out of 10000 arrived before the corresponding region page was pinned. Fortunately, this packet drop does not impact performance significantly since it is resent almost immediately most of the times[4].

If a single core is overloaded by the incoming network traffic for some reason (10G traffic, many small packets, interrupts bound to a single core, ...), the strongly privileged receive processing (*Bottom Half* interrupt handler) may exhaust the core availability. In this case, a process running on the same core would have a high probability of pinning very slowly. It would cause many overlap-misses. We observed throughput degradation from 1 GB/s down to 50 MB/s. Fortunately, this case is very rare under regular MPI load. But, we are evaluating the idea of pinning a few pages synchronously anyway before sending the initiating message to reduce the chance of getting

[3]One process per core, sharing a single 10G NIC.

[4]If some packets with higher sequence numbers are received, the missing packet is requested again optimistically, instead of waiting for the retransmission timeout (1s) to expire.

some overlap-misses.

4.4 Application-level Performance

Application	Pinning-cache	Overlapping
IMB SendRecv	8.4 %	5.5 %
IMB Allgatherv	7.5 %	6.8 %
IMB Broadcast	4.4 %	2.0 %
IMB Reduce	7.6 %	0.2 %
IMB Allreduce	2.2 %	-0.6 %
IMB Reduce scatter	7.9 %	-0.8 %
IMB Exchange	-1.4 %	-2.7 %
NPB is.C.4	4.2 %	1.9 %

Table 2. Execution time improvement brought by the Open-MX pinning cache or the overlapped pinning on IMB and NPB, between 2 nodes.

Table 2 presents the impact of our pinning optimizations on the *Intel MPI Benchmarks* and *NAS Parallel Benchmarks*. As shown in the past [15], the pinning cache brings an interesting performance improvements in most cases since it benefits as soon as the same memory region is used for multiple operations. The overlapped pinning impact depends much more on the communication pattern, it goes from +6.8 % down to -2.7 %. We think that this variety of results is related to the way the internal OPEN MPI collective implementation overlaps low-level communications. Indeed, our overlapped pinning is not expected to help overlapped communications much since it does not increase the CPU availability. Moreover, our implementation still has a small overhead that may explain the negative impact in some cases.

The impact on real applications such as NPB seems to be mostly related to how many large messages are involved. The large-message intensive IS shows an interesting 1.9 % improvement thanks to pinning overlap and up to 4.2 % with pinning cache. The performance of other NAS tests does not vary much since they mostly rely on small messages while we only optimize large messages in this work.

5 Related Works and Discussion

Memory pinning has been the target of many research projects in the last decade, mostly because of the domination of high-speed networks in HPC. Many MPI implementations such as OPEN MPI [6] and MVAPICH [11] rely on a user-space memory registration cache. However, symbol interception only works for dynamically linked programs and if the usual malloc/mmap stack is used. Second, these malloc hooks are called for every deallocation, even for very small buffers that have nothing to do with communication. Kernel based implementations such as ours do not have these drawbacks since the hooks are reliable and only called when a large region is actually unmapped at the kernel level.

Several kernel based registration cache implementations have been proposed in the past. They either deviate the virtual memory area hooks in the kernel [16] or rely on a patched kernel [5]. QSNET [12] was even able to avoid the need to pin by using a heavily patched virtual memory system and an advanced memory management unit in the NIC. To the best of our knowledge, OPEN-MX is the first stack to use the new *MMU Notifiers* interface of the LINUX kernel so as to implement a pinning cache in a viable and widely supported manner. Moreover, our model is very simple and does not require any synchronization with a user-space library [16] since pinning is only managed by the driver.

MMU Notifiers have been designed for the KVM virtualization subsystem of the LINUX kernel. The KVM host uses *MMU Notifiers* to maintain a shadow copy of the guest operating system [1]. This problem is actually very similar to high-speed networks where a copy of the host page table has to be stored in the NIC. So *MMU Notifiers* were also designed with the idea of helping high-speed networking, but OPEN-MX is the only one to use them yet to the best of our knowledge.

Overlapping memory registration with communication has also been proposed in the past as a way to lower the observed registration cost. However, these implementations, for instance MPICH-GM [3] and OPEN MPI [6], were based on splitting large buffers into smaller ones and overlapping the pinning of current one with the sending of the former one. This model requires a complex management protocol. It may also reduce the overall throughput since smaller messages are sent on the wire and the first message has to wait for its own pinning to be done before being sent. OPEN-MX does not have these issues since true overlapping is managed at the driver level and the whole message is sent at once immediately.

It has to be noted that our overlapped memory pinning model does not reduce the overall communication processing time since it only reschedules the actual pinning after the low-level sending of the initiating message. Its impact is optimal if the application blocks until the communication completion since the pinning will be overlapped with the blocking. Blocking MPI operations are therefore the primary target of this optimization. Non-blocking operations will benefit less from overlapped pinning if the application overlaps communication with computation since the CPU availability will not increase. We are thus evaluating the idea of only enabling decoupled/overlapped pinning for blocking operations. However, since a non-blocking operation may complete earlier thanks to pinning being removed from the critical path, we feel that our overlapped model may still have an positive impact on real application performance if using non-blocking operations. Anyway, the pinning cache may still help both blocking and non-blocking operations if the application reuses the same buffer multiple times. If the application does not, our overlapping comes as a complement since the pinning cache cannot help.

6 Conclusion and Perspectives

Memory pinning is a key requirement of high-performance networking. It enables zero-copy communication which reduces the host load and improves throughput. However, pinning is known to be very expensive and may thus dramatically decrease the overall performance. The OPEN-MX stack depends on memory pinning similarly. And its impact on throughput varies from 5 to 20 % depending on the host and network performance.

We presented in this paper an in-depth optimization of memory pinning in the OPEN-MX stack. Thanks to OPEN-MX being non-*OS bypass*, we are able to decouple memory pinning from the user-space library and applications. This model brings the safety and reliability of a kernel-based pinning cache without requiring the complexity of notifying user-space of memory region invalidation. It also enables overlapping of pinning with communication so as to start the actual data transfer earlier.

We implemented this model using the new *MMU Notifiers* framework of the LINUX kernel which provides a reliable and widely supported way to maintain our pinning up to date. Microbenchmarks show that both our cache and overlap optimizations bring the expected throughput improvement on 10G ETHERNET networks, from 5 to 20 % depending on the host frequency. Real application experiments prove that the on-demand pinning cache improves performance as soon as the same buffer is involved in multiple operations. Overlapped memory pinning may also help applications that use blocking operations when the pinning cache is useless.

While this work may appear OPEN-MX specific, we are actually looking at applying it to other networking stacks. First, our overlapped pinning model should be applicable when it is not strictly required to pin the whole memory region at once. For instance, the MX stack or the TCP I/OAT copy offload in the LINUX kernel are good candidates for similar optimization. Second, on-demand memory pinning decoupled from the application is applicable when the kernel is aware of communication requests, which means the stack is not *OS bypass* and 2-sided operations are used. So we expect to be able to adapt this idea to other message passing stacks over generic ETHERNET hardware.

We also plan to configure our model depending on the application behavior since for instance blocking operations benefit more from overlapped pinning while overlap-aware applications may prefer a simple model with lower overhead. In the long term, the idea of removing the need to pin entirely, as implemented on QSNET, will have to be studied as well.

References

[1] A. Arcangeli. Integrating KVM with the Linux Memory Management. KVM Forum, June 2008.

[2] G. Ciaccio and G. Chiola. GAMMA and MPI/GAMMA on gigabitethernet. In *Proceedings of 7th EuroPVM-MPI conference*, Balatonfured, Hongrie, Sept. 2000.

[3] P. Geoffray. MPICH-GM and VI-GM middlewares. Myrinet Users Group Conference, May 2002.

[4] B. Goglin. Design and Implementation of Open-MX: High-Performance Message Passing over generic Ethernet hardware. In *Proceedings of the CAC 2008 Workshop*, Miami, FL, Apr. 2008.

[5] B. Goglin, L. Prylli, and O. Glück. Optimizations of Client's side communications in a Distributed File System within a Myrinet Cluster. In *Proceedings of the HSLN 2008 Workshop*, pages 726–733, Tampa, Florida, Nov. 2004.

[6] R. L. Graham, T. S. Woodall, and J. M. Squyres. Open MPI: A Flexible High Performance MPI. In *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, Poznan, Poland, Sept. 2005.

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

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

[9] F. Mietke, R. Baumgartl, R. Rex, T. Mehlan, T. Hoefler, and W. Rehm. Analysis of the Memory Registration Process in the Mellanox InfiniBand Software Stack. In *Euro-Par 2006 Parallel Processing*, pages 124–133. Springer-Verlag Berlin, 8 2006.

[10] 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`.

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

[12] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, 2002.

[13] P. Shivam, P. Wyckoff, and D. K. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *Proceeding of Supercomputing 2001*, Denver, CO, Nov. 2001.

[14] S. Sumimoto, K. Ooe, K. Kumon, T. Boku, M. Sato, and A. Ukawa. A Scalable Communication Layer for Multi-Dimensional Hyper Crossbar Network Using Multiple Gigabit Ethernet. In *Proceedings of ICS'06*, pages 107–115, Cairns, Australia, 2006.

[15] 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, Apr. 1998.

[16] P. Wyckoff and J. Wu. Memory Registration Caching Correctness. In *Proceedings of CC-Grid 2005*, Cardiff, Wales, 2005.