Analysis of the Memory Registration Process in the Mellanox InfiniBand Software Stack

Frank Mietke, Robert Rex, Robert Baumgartl,
Torsten Mehlan, Torsten Hoefler and Wolfgang Rehm

Department of Computer Science
Chemnitz University of Technology, Germany
{firstname.surname}@informatik.tu-chemnitz.de

Abstract. To leverage high speed interconnects like InfiniBand it is important to minimize the communication overhead. The most interfering overhead is the registration of communication memory.
In this paper, we present our analysis of the memory registration process inside the Mellanox InfiniBand driver and possible ways out of this bottleneck. We evaluate and characterize the most time consuming parts in the execution path of the memory registration function using the Read Time Stamp Counter (RDTSC) instruction. We present measurements on AMD Opteron and Intel Xeon systems with different types of Host Channel Adapters for PCI-X and PCI-Express. Finally, we conclude with first results using Linux hugepage support to shorten the time of registering a memory region.

1 Introduction

High speed interconnects like InfiniBand [4] or Myrinet [11] use DMA engines in conjunction with user level communication protocols to achieve high bandwidth, low latency and a low CPU utilization. That is the user level application (Consumer in InfiniBand Architecture) just creates a communication request including the relevant information like starting address and length of the communication buffer. This communication request is then transmitted to the network adapter (Host Channel Adapter in InfiniBand) through a simple user level API function call. For a normal send operation the HCA takes the request to create the appropriate packet structure and programs the DMA engine to get the user data. After this the packet is immediately transferred to the other communication partner. This process is depicted in figure 1.

The DMA engine responsible for transferring the data from main memory to the network adapter handles only physical addresses. Thus the virtual addresses of the communication buffer have to be translated into a physical one. Furthermore it is important to ensure that every page of the communication buffer is pinned to prevent swapping. This process of pinning and address translation is called memory registration. Every communication operation of InfiniBand needs

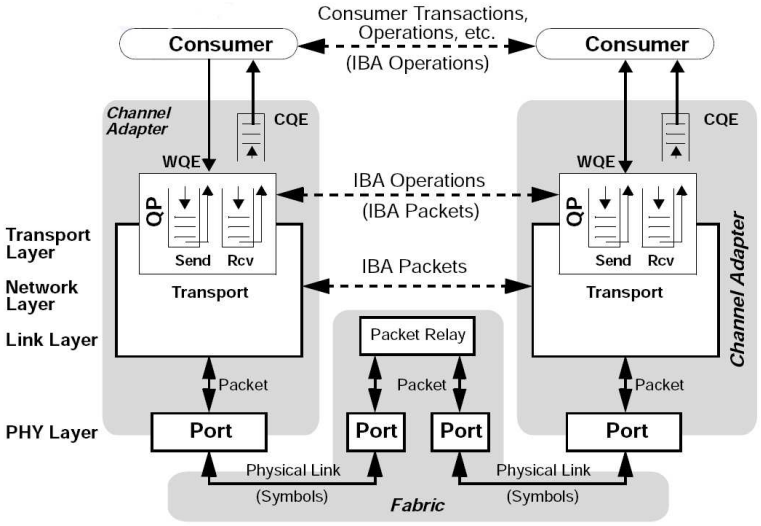

Fig. 1. InfiniBand Architecture Communication Stack ([4])

registered memory except the inline send operation where the data is directly transferred to the network adapter inside the communication request.

To avoid the expensive registration costs several approaches were investigated and integrated. We present them in the next section. In section 3 we give a detailed description of the memory registration function of the Mellanox InfiniBand software stack and how we measured the single components inside this function. This section includes also the appropriate measurements and presents the results of our first approach to shorten the registration time. We summarize and conclude in section 4.

2 Related Work

Due to the costs of memory registration several approaches try to reduce the impact of this operation on middleware or application level. These approaches can roughly be categorized in two classes static and dynamic. Static means that every memory area is registered in advance or it is hidden in a memory allocation call. Dynamic means that a memory area is registered on the fly in the communication path. Typically – to complicate there work further – the amount of pinned memory pages and the number of registered memory regions may be limited.

Avoiding the registration operation by using memory copies in conjunction with pre-registered memory regions belongs to the static class. This is typically used

if only small messages are sent or received to improve the latency behaviour. Application examples are several InfiniBand MPI implementations like MVAPICH [8], MVAPICH2 [7] and MPICH2-CH3-IB [3].
Registration of the whole physical memory or parts thereof in advance is another approach in the static class. A call to malloc allocates then already registered memory for the application. DSM systems like [6] use this approach.

Tezuka proposed the Pin-Down caching [14] where a lazy deregistration mechanism is applied. That is memory regions are registered once and then hold in a cache. To improve the search speed a hash table is used in MVAPICH [8] and MPICH2-CH3-IB [3]. To find memory areas with different starting page addresses that reside inside of another is not possible in a hash table. To remedy this problem tree structures are used in VIA-RPI [9] for LAM/MPI and Open-MPI [16] instead of hash tables. All these approaches belong to the dynamic class and are typically used to transfer large messages.

Other dynamic approaches are Fast Memory Registration and Deregistration (FMRD) [17] as well as Optimistic Group Registration (OGR) [18]. Both are proposed for an InfiniBand PVFS implementation to improve the speed of Pin-Down caching and noncontiguous memory registration.

Further proposals to improve the handling with the registration operation were made in [13],[15],[19] and [2].
But all the above mentioned approaches merely tried to mitigate the registration costs in an application specific manner and expect an efficient implementation of the registration operation. To the best of our knowledge there has been no detailed analysis which went underneath the registration call.

3 Memory Registration Analysis

It has been observed by several researchers that registering memory for communication is very time-consuming. Table 1 compares the best case (no registration at all) and worst case (every buffer must be registered) scenario running the SendRecv test of the Intel MPI Benchmark [5] suite between two Opteron test-systems each hosting a PCI-express InfiniBand HCA. This comparison clearly shows how big the influence on communication performance is. A detailed analysis regarding the impact on applications is done in [10].
The main goal of the work described here is to obtain a precise understanding of the execution timing of all InfiniBand driver functions contributing to memory registration. We aimed at identifiying potential performance bottlenecks and entry points for optimization.

Msg size	Bandwidth when Registration necessary	FOI if No Registration
32kB	270MB/s	3.22
64kB	457MB/s	2.55
128kB	701MB/s	2.00
256kB	892MB/s	1.74
512kB	1058MB/s	1.56
1024kB	1217MB/s	1.39
2048kB	1295MB/s	1.33
4096kB	1332MB/s	1.31

Table 1. Factor of improvement (FOI) when there are no registration costs

3.1 Profiling the Driver

Prior to data transfer, the following main functions are performed sequentially by the driver:

- pin the requested quantity of memory pages for subsequent DMA transfers by the IB controller using `mlock()`,
- translate the virtual addresses of the pinned pages into physical addresses,
- transmit the obtained physical addresses to the IB controller.

It is irrelevant, whether a send or a receive operation follows that preparation. The sequence constitutes the *memory registration*.

The first experiment focused on profiling this sequence. We instrumented the relevant driver functions (mainly `VAPI_register_mr()`) of the Mellanox InfiniBand driver API with `rdtsc` machine instructions and code to write the obtained time stamps into the kernel log. This writing needed approximately 2000-5000 clock cycles which is two to three orders of magnitude smaller than the functions profiled. Therefore we could safely neglect that measurement error.

Two different situations concerning `mlock()` can be distinguished:

a) All or most of the pages to be pinned are present in main memory.
b) The pages are not present in memory. `mlock()` generates page faults and its performance degrades.

Situation b) typically occurs when: Either the buffer is allocated and registered for the very first time or the pages have been swapped out due to tight memory. That is, the former case usually occurs when receiver memory is registered, or sender memory is pre-registered during the init stage. The latter case should be avoided at all costs, e. g, by fitting a maximum of physical main memory into the machine. We conducted our experiments for both situations.

The experimental setup consisted of an AMD Opteron 244 Dual Processor Machine clocked at 1.8 GHz and equipped with 2 GBytes of RAM. We used the

PCI-Express InfiniBand Adapter MT25208 InfiniHost III Ex with 256MB RAM and the MemFree version respectively. All experiments were conducted with the Mellanox InfiniBand Gold Edition Package (IBGD), versions 1.7.0 and 1.8.0. The operating system was a standard Linux kernel, version 2.6.10UP and SuSe 2.6.11.4-20a-smp (MemFree HCA). One observation that can be made between the UP and SMP kernel is a slightly bigger overhead due to the spinlock insertion in SMP kernels. All figures are in doubly logarithmic representation.

Figure 2 and 3 depicts the execution timings for the individual steps and the overall memory registration for different communication buffer sizes when the pages of the buffer to be registered are present in memory.

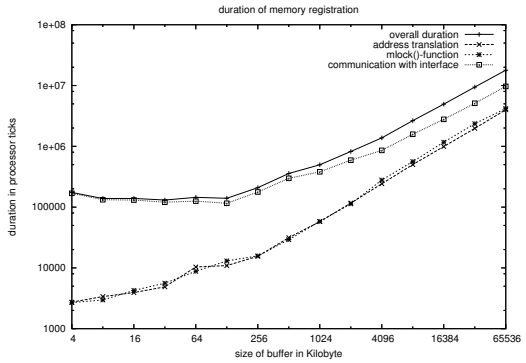

Fig. 2. Memory Registration Performance, Pages Present

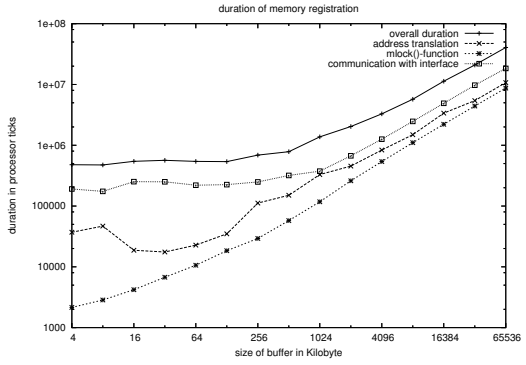

Fig. 3. Memory Registration Performance, Pages Present (MemFree HCA)

The most time-consuming factor is the address transfer to the IB controller. Pinning and address translation contribute to overall timing only marginally with a slightly larger influence for large buffers. Unfortunately, the communication with the IB controller does not exhibit much optimization potential, because it is bound by the controller's reaction time.

Figure 4 and 5 depicts the same execution timings with the pages of the buffer not present in memory.

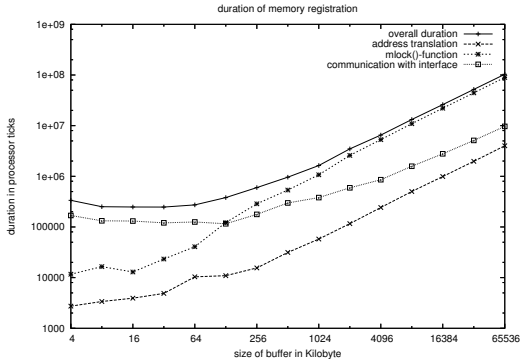

Fig. 4. Memory Registration Performance, Pages not Present

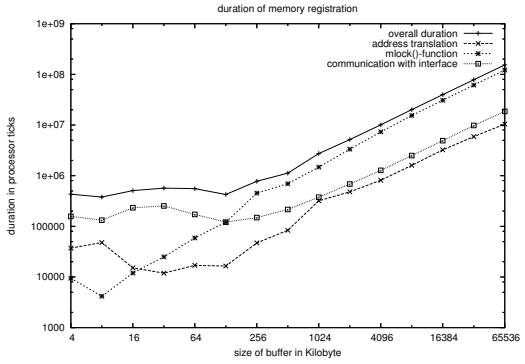

Fig. 5. Memory Registration Performance, Pages not Present (MemFree HCA)

Several observations can be made here: For buffers larger than 256 kBytes registration time is almost completely dominated by pinning whereas for small buffer sizes the communication with the adapter is the most influential factor. This is not surprising, because the number of occuring page faults increases with

buffer size. Virtual-to-Physical address translation is almost not influencing the registration timing. Registration of small buffers has almost a constant timing overhead regardless of the exact buffer size.

We repeated both experiments on an Intel Xeon SMP system hosting 2 CPUs at 2.4 GHz and 2 GBytes of memory and a PCI-X InfiniBand HCA. The software used was the same as on the Opteron System with the 256MB HCA mentioned above. We used a slightly modified methodology due to some driver peculiarities and obtained very similar timing proportions as one can see in figure 6 and 7.

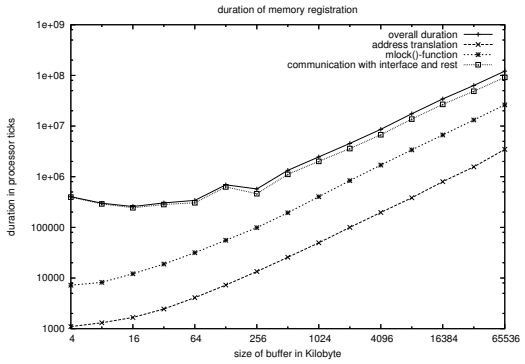

Fig. 6. Memory Registration Performance, Pages Present (PCI-X HCA)

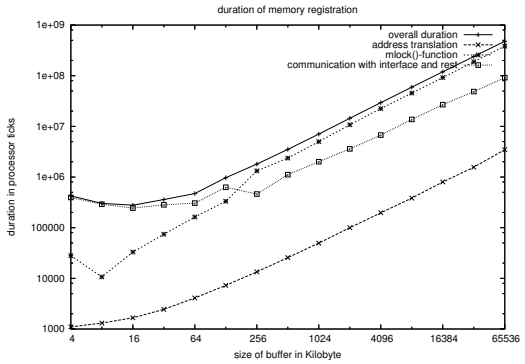

Fig. 7. Memory Registration Performance, Pages not Present (PCI-X HCA)

But the quantitative values are worse compared to AMD Opterons due to the different locations of the memory controller and PCI-X vs. PCI-express (Opteron) HCAs. More details can be found in [12].

3.2 Performance of `mlock()`

Because `mlock()` performance seemed relevant if pages are not present, we next concentrated on profiling it. The table lists the obtained timings for pinning a single page of 4kBytes on the AMD Opteron and the Intel Xeon processor when the page is not present in memory. The shown profile is a typical case when registering receiver memory or preregister sender memory. All times are in processor cycles.

Function	*AMD Opteron*	*Intel Xeon*
Search for Free Page Frame and Update Page Table	3500	9000
Zero out Page Frame	1000	2000
Pin the Page	1800	5400

Table 2. Timing Profile for mlock()

Even if you normalize the numbers of the timings the Xeon system needs almost twice the time to execute `mlock()` due to its memory subsystem. As you can see the pinning itself is now only a fraction of the costs of the `mlock()` call. We tried in some experiments to remove the zeroing step but failed with libraries which presume zeroed pages.

3.3 Using Large Pages

Most modern processors like Intel Xeon or AMD Opteron support different page sizes. The most obvious improvement of registration time could be the usage of larger pages. The current 2.6 Linux kernels [1] provide the hugetlbfs to use these different page sizes simultaneously. Apparent advantages using larger page sizes for registering memory are:

- `mlock()` has to pin less pages
- there are less address translations
- and thus less translations has to be transferred to the HCA
- the Mellanox driver can already use large page sizes

To use the hugetlbfs it is necessary to utilize `mmap()` or shared memory system calls. In table 3 the registration times are shown using 4kB and 2MB page sizes. The timings in the 4kB column correspond to the values of figure 4. To be comparable the timings in the 2MB column include `mmap()` and the register call. By using hugetlbfs one attains improvements of 15% up to 25%.

Buffer Size in kB	*Registration Time 4kB (ms)*	*Registration Time 2MB (ms)*
2048	1.8	1.5
4096	3.7	2.9
8192	7.4	5.7
16384	14.7	11.3
32768	28.8	22.5
65536	57.9	45.0

Table 3. Comparison of registration time for 4kB and 2MB page sizes

4 Summary and Conclusions

With this paper we have given a quantitative analysis of the execution timing of the memory registration inside the Mellanox InfiniBand driver. We showed that in the case where the pages are not present the `mlock()` call is the dominant factor. Otherwise the communication with the adapter to communicate the address translations is the dominant part. Furthermore, we showed that the AMD Opteron has a much better timing behaviour of the `mlock()` call than the Intel Xeon.
Finally we presented our first results using larger page sizes and showed that improvements of 15% up to 25% are attainable using the mmap approach.

To improve the behaviour of `mlock()` when pages are not present, a seperate kernel thread could fill the pages with zeros when the kernel has time. This could drastically reduce the amount of work which `mlock()` does. To avoid the address translation and thus the communication with the HCA, one would have to change the behaviour of the HCA that it can handle virtual addresses and has access to the kernel page tables. Finally to better utilize hugetlbfs we have to provide a malloc/free library that supports multiple page sizes simultaneously. Therefore also the communication protocol to convey the address translation in the InfiniBand driver has to be changed. Then the applications can transparently make use of this kernel feature in a memory footprint efficient manner.
All these propositions will be investigated in the future.

References

[1] L. K. Archives. Website. `http://www.kernel.org`.
[2] C. Bell and D. Bonachea. A New DMA Registration Strategy for Pinning-Based High Performance Networks. In *In Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS 03)*, April 2003.
[3] R. Grabner, F. Mietke, and W. Rehm. Implementing an MPICH-2 Channel Device over VAPI on InfiniBand. In *Proceedings of the 18th Int'l Parallel and Distributed Processing Symposium, IPDPS*, 2004.

[4] InfiniBand Trade Association. *InfiniBand Architecture Specification 1.2*, 2004.

[5] Intel GmbH, Hermlheimer Str. 8a, D-50321 Brhl, Germany. *Intel MPI Benchmarks – Users Guide and Methodology Description*.

[6] L. Liss, Y. Birk, and A. Schuster. In-Kernel Integration of Operating System and Infiniband Functions for High Performance Computing Clusters: A DSM Example. *IEEE Transactions on Parallel and Distributed Systems*, 16(9), September 2005.

[7] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In *In Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS 04)*, April 2004.

[8] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *In the Proceedings of 17th Annual ACM International Conference on Supercomputing*, June 2003.

[9] T. Mehlan, W. Rehm, R. Engler, and T. Wenzel. Providing a High-Performance VIA-Module for LAM/MPI. In *In Proceedings of IEEE International Conference on Parallel Computing in Electrical Engineering (PARELEC'04)*, September 2004.

[10] F. Mietke, R. Rex, T. Mehlan, T. Hoefler, and W. Rehm. Reducing the Impact of Memory Registration in InfiniBand. In *Proceedings of the 1. Workshop Kommunikation in Clusterrechnern und Clusterverbundsystemen (KiCC)*, 2005.

[11] Myrinet. Myrinet Inc. `http://www.myri.com`.

[12] R. Rex. Analysis and Evaluation of Memory Locking Operations for High-Speed Network Interconnects. Student Project, Chemnitz University of Technology, October 2005.

[13] S. Sur, U. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda. High Performance RDMA Based All-to-all Broadcast for InfiniBand Clusters. In *In Proceedings of International Conference on High Performance Computing (HiPC 2005)*, December 2005.

[14] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *In Proceedings of 12th Int. Parallel Processing Symposium*, March 1998.

[15] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D. K. Panda. Host-Assisted Zero-Copy Remote Memory Access Communication on InfiniBand. In *In Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS 04)*, April 2004.

[16] O. M. Website. A High Performance Message Passing Library. `http://www.open-mpi.org`.

[17] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *In Proceedings of International Conference on Parallel Processing (ICPP 03)*, October 2003.

[18] J. Wu, P. Wyckoff, and D. K. Panda. Supporting Efficient Noncontiguous Access in PVFS over InfiniBand. In *In Proceedings of IEEE International Conference on Cluster Computing (Cluster'2003)*, December 2003.

[19] J. Wu, P. Wyckoff, D. K. Panda, and R. Ross. Unifier: Unifying Cache Management and Communication Buffer Management for PVFS over InfiniBand. In *In Proceedings of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 04)*, April 2004.