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

Brice Goglin
INRIA Runtime Team-Project

CAC Workshop – IPDPS'09 – Rome - 2009/05/25
Outline

User-space I/Os, memory pinning and Open-MX

How to optimize pinning in Open-MX
  • Pinning cache
  • Decoupling pinning from the application
  • Overlapped and on-demand pinning

Performance evaluation

Conclusion and Future Works
User-space I/Os

I/Os from/to user application (virtual) buffers

The OS may swap/migrate virtual pages
  • No guarantee that a virtual buffer remains at the same physical location

The hardware manipulates DMA addresses
  • The OS must enforce physical location of buffer during I/O
    • Pin virtual buffers down in physical memory
    • Used for disk I/Os, high-speed networks, copy offload, …
  • No buffer-cache, zero-copy, …
Open-MX stack

Applications

MX Library
MX Driver
MX Firmware
Open-MX Library
Open-MX Driver
Ethernet Driver
I/OAT Driver
Ethernet Board
I/OAT Chipset

Ethernet Packets
Memory pinning in Open-MX

Zero-copy on the send-side

- User-buffers pinned and given to the NIC

Offloading of Rx copies on I/OAT DMA engine

- Offloaded copy from received packet into user-buffer
  - User-buffer must be pinned

User-memory access outside of the process context

- Cannot page-fault another address space
  - Bottom half interrupt handler Rx copy
  - User-buffer must be pinned
MXoE Large Message Protocol

MPI_Send

MPI_Recv

matching

Rendezvous

Pin

Pull Request

Pull Replies

Pull Request

Pull Replies

Notify

Unpin

completion
## Pinning+Unpinning overhead

<table>
<thead>
<tr>
<th>Processor</th>
<th>Frequency</th>
<th>Base Overhead</th>
<th>Per-page Overhead</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Opteron 265</td>
<td>1.8 GHz</td>
<td>4.2 µs</td>
<td>720 ns</td>
<td>5.5 GB/s</td>
</tr>
<tr>
<td>AMD Opteron 8347HE</td>
<td>1.9 GHz</td>
<td>2.2 µs</td>
<td>330 ns</td>
<td>12 GB/s</td>
</tr>
<tr>
<td>Intel Xeon E5345</td>
<td>2.33 GHz</td>
<td>2.3 µs</td>
<td>250 ns</td>
<td>16 GB/s</td>
</tr>
<tr>
<td>Intel Xeon E5460</td>
<td>3.16 GHz</td>
<td>1.3 µs</td>
<td>150 ns</td>
<td>26.5 GB/s</td>
</tr>
</tbody>
</table>

**Pinning overhead goes down with new CPUs**
- But networks get faster

**5-20% impact expected on 10G Ethernet throughput**
- Depending on the processor performance
How much does pinning matter in Open-MX?

![Graph showing throughput (MiB/s) vs message size for different pinning strategies in Open-MX.]

- Open-MX - Pin once per Communication
- Open-MX - Permanent Pinning
- Open-MX + I/OAT - Pin once per Communication
- Open-MX + I/OAT - Permanent Pinning
How to optimize/hide pinning overhead?

Pinning required in Open-MX
• and impact on performance is large

Pinning cache (aka Registration Cache)
• Keep buffers pinned as long as possible
• Need to invalidate pinned buffers when the application changes the address space layout (free, munmap, …)

Pinning overlap
• Pin while the rendezvous handshake is being done
Why pinning cache isn't so easy

The cache must be updated when the application memory layout changes

Need to intercept applications memory-related calls (free, munmap, ...)
  • Only works for dynamically linked applications
  • May conflict with other middleware intercepting calls

Applications do not always reuse the same buffer
  • The cache may work and remain useless
Intercepting in the kernel (redux)

Old solutions to avoid user-level intercepting

- Patch your kernel (Quadrics)
- Hack VMA operations and hope nobody else does

Linux 2.6.27 introduces *MMU Notifiers*

- Designed to update KVM shadow guest page tables
- Similar to the above proposed interfaces
  - But this new one is supported for sure

Problem

- You still have to update the cache that the user-space library maintains
  - Kernel-User synchronization overhead
Decoupling pinning from the application

Caching buffers may be done in user-space
  • But knowledge of pinning only required in the driver
    • Where the actual I/O are implemented

Let the driver pin/unpin when needed
  • Pin when the buffer is used
  • Unpin when invalidated through *MMU Notifiers*
  • Unpin if too many pinned pages/buffers
On-demand overlapped Pinning

When a send or recv starts

- Tell the driver which buffers must be pinned
  - Means the stack is non-OS-bypass and 2-sided
- Start the communication
- The driver starts pinning when needed

The buffer is probably already pinned when needed

- Because there's a round-trip in the MXoE protocol
  - Rendezvous+PullRequest on the sender side
  - PullRequest+PullReply on the receiver side
MXoE Large Message Protocol

MPI_Recv

*matching*

MPI_Send

*completion*

*notify*

*unpin*

*pin*
Why is this different from existing overlap strategies?

MPICH-GM, OpenMPI, … overlap pinning of chunks with transmission of previous chunk

Open-MX doesn't split messages in multiple chunks
  • Single big message on the wire
    • Optimal throughput

Open-MX doesn't wait for first chunk to be pinned before sending first packets
  • Very small startup overhead

Why possible?
  • Because pinning and communication both in the driver
Improvement of ping-pong performance
Early conclusions

Overlapped pinning gives as much performance improvement as pinning cache does
  • But overlapping only reschedules the overhead
    • Result depends on the CPU availability
      • Does the above layers use the available overlap time?
      • MPI blocking calls!
  • Pinning cache reduces the overhead for real
    • But does not always work
      • Static linking, intercepting conflicts, ...
      • Applications not reusing the same buffers again and again
Overlap misses

What if we fail to pin enough pages on time?
  • A packet arrives and need a non-yet-pinned page

Queue the packet and process it later?
  • Management and locking overhead

Problems should be rare
  • Fastest machine: 78 pages pinned during round-trip (13µs)
  • Slowest machine: 19 pages pinned (18µs)
  • First packet of protocol only cares about 8 first pages
Overlap misses (2/2)

Drop the packet!
• The retransmission protocol will resend it later anyway
  • Optimistically re-requesting missing sequence numbers early
  • And multiple pull requests/replies are pipelined
  • One packet drop is hidden by the others

0.01% miss happens, but does not disturb much

Performance poor under extreme circumstances
• One CPU core overloaded by interrupt processing
  • Throughput down to 50MB/s
  • This case should not happen for MPI applications
    • Interrupts usually scattered across all cores, and coalesced a bit
# IMB (4MB) and NAS improvements

<table>
<thead>
<tr>
<th>Application</th>
<th>Pinning-cache</th>
<th>Overlapped Pinning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMB SendRecv</td>
<td>8.4 %</td>
<td>5.5 %</td>
</tr>
<tr>
<td>IMB Allgatherv</td>
<td>7.5 %</td>
<td>6.8 %</td>
</tr>
<tr>
<td>IMB Broadcast</td>
<td>4.4 %</td>
<td>2.0 %</td>
</tr>
<tr>
<td>IMB Reduce</td>
<td>7.6 %</td>
<td>0.2 %</td>
</tr>
<tr>
<td>IMB Allreduce</td>
<td>2.2 %</td>
<td>-0.6 %</td>
</tr>
<tr>
<td>IMB Reduce scatter</td>
<td>7.9 %</td>
<td>-0.8 %</td>
</tr>
<tr>
<td>IMB Exchange</td>
<td>-1.4 %</td>
<td>-2.7 %</td>
</tr>
<tr>
<td>NPB is.C.4</td>
<td>4.2 %</td>
<td>1.9 %</td>
</tr>
</tbody>
</table>
Conclusion

Pinning-cache often helps

- Reduces the actual host overhead
- *MMU Notifier*-based model, decoupled from user-space
  - No need to synchronize between kernel and user

Overlapped pinning reschedules the overhead

- Only helps if the application (or MPI implementation) does not already use the available overlap time

Use a combination of both

- Overlapped On-demand pinning
Future Works

Add pinning overlap to other subsystems
  • I/OAT copy offload clients (TCP, …)
  • Other message-passing stacks
    • Need non-OS-bypass and 2-sided

Look at application behavior before deciding to overlap or not
  • If blocking MPI call, overlap is much likely to help

Remove pinning entirely
  • Like Quadrics did thanks to a kernel patch
Thanks for your attention!
Questions?

http://open-mx.org
Brice.Goglin@inria.fr