ABSTRACT

- IO and memory pinning
- The price of pinning
- The price of memory management
- Getting it all (1/2) – On Demand Paging
- Getting it all (2/2) – The address space key
- APIs
- Statistics
- Development
- Evaluation
- What’s next?
IO AND PINNING

- PCI devices are granted access to buffers
  - NIC Tx/Rx buffers
  - Guest physical pages when passing through a PCI device to a VM
  - Underlying pages comprising an RDMA memory region

- Underlying pages must be available for DMA until IO completion
  - Until an Rx buffer is used on a NIC
  - The VM lifetime in PCI pass-through
  - Memory region lifetime in RDMA

Large, long-lived mappings
THE PRICE OF PINNING

- No canonical memory optimizations

<table>
<thead>
<tr>
<th>Demand paging</th>
<th>Over commitment</th>
<th>Page migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delayed allocation</td>
<td>Swapping</td>
<td>NUMA migration</td>
</tr>
<tr>
<td>Mmap-ed files</td>
<td>Deduplication</td>
<td>Compaction</td>
</tr>
<tr>
<td>Calloc with zero page</td>
<td>Copy on write</td>
<td>Transparent huge pages</td>
</tr>
</tbody>
</table>

- Complicates administration and deployment
  - Unprivileged applications must be granted explicit rights to lock memory
  - Worst-case pinning in the absence of a good alternative to estimate pinning requirements
  - IO buffers limited to size of physical memory
THE PRICE OF MEMORY MANAGEMENT

- Application must be aware which memory is registered and which isn’t

- Application-specific memory pools put aside decades of memory management development
  - OS
  - C runtime
  - Libraries

- Memory registration is a major inhibitor to RDMA adoption
  - Require complex, expert programming
  - A non-starter for many new-comers
MITIGATING COSTS: IO BOUNCE BUFFERS

- **Fixed pool of buffers**
  - Data is copied in/out of these buffers for IO

- **Efficient for small messages**

- **Drawbacks**
  - Significant costs for large messages
  - Hard to estimate pinned buffer size
    - Large variance between common-case and worst-case
    - Dynamic resizing is costly and difficult
MITIGATING COSTS: DYNAMIC PINNING

- **Pin-down cache libraries**
  - Pin/unpin buffers on the fly
  - Amortize high pinning costs by caching registrations

- **Drawbacks**
  - Complex logic
  - Hard to generalize due to software layering
  - Hard to optimize
  - Hard to track and maintain consistency
    - Need to hook every allocation / free function

---

Diagram:

- Application
- MPI
- Communications library
  - malloc
  - calloc
  - mmap
  - munmap
  - sbrk
  - fork
  - ...  
- OS
- Pin-down cache
- Buffer allocation
- Buffer registration
HCA translation tables may contain non-present pages
- Initially, a new MR is created with non-present pages
- Virtual memory mappings don’t necessarily exist

MR pages are *never* pinned by the OS
- Paged in when HCA needs them
- Paged out when reclaimed by the OS

Eliminates the price of pinning
- Unlimited MR sizes
  - No need for special privileges
- Physical memory optimized to hold current working set
  - For both CPU and IO access
- Application pages may be migrated at will
ODP promise:
IO virtual address mapping == Process virtual address mapping
ODP IMPLEMENTATION

- **ib_core**
  - Key->MR tree
  - MR interval tree

- **mlx5_core/** mlx5_ib
  - Key->MR tree

- **HCA**
  - QP page fault event

- **Kernel**
  - mmu_notifier page invalidation

- **MR**
  - umem
  - Page/DMA list
  - mlx5_ib_mr
  - HW translation tables

- **OpenFabrics Alliance Workshop 2017**
GETTING IT ALL (2/2)
Address Space Key

- Register the whole process address space with a single key
  - MR covers existing and future memory mappings

- **MR covers unmapped address ranges**
  - Permissions checked at access (page fault) time
    - VMA permissions
    - MR access rights
    - RDMA access rights revoked upon invalidation or permission changes

- **Granular remote permissions via Memory Windows**
  - User-space equivalent for Fast Registration Work Requests…

- **Eliminates the price of memory management**
  - All data transfer done based on the address space key
  - No need to register and track any other MRs
enum odp_transport_cap_bits {
    ODP_SUPPORT_SEND  = 1 << 0,
    ODP_SUPPORT_RECV  = 1 << 1,
    ODP_SUPPORT_WRITE = 1 << 2,
    ODP_SUPPORT_READ  = 1 << 3,
    ODP_SUPPORT_ATOMIC= 1 << 4,
};

enum odp_general_caps {
    ODP_SUPPORT  = 1 << 0,
};

struct ibv_odp_caps {
    uint32_t comp_mask;
    uint32_t general_caps;
    struct {
        uint32_t rc_odp_caps;
        uint32_t uc_odp_caps;
        uint32_t ud_odp_caps;
    } per_transport_caps;
};

int ibv_query_odp_caps(struct ibv_context *context, struct ibv_odp_caps *caps, size_t caps_size);
### ODP MEMORY REGIONS

- **Registering the whole address space**
  - `ibv_reg_mr(pd, NULL, (u64)-1, flags)`

```c
enum ibv_access_flags {
    IBV_ACCESS_LOCAL_WRITE = 1,
    IBV_ACCESS_REMOTE_WRITE = (1<<1),
    IBV_ACCESS_REMOTE_READ = (1<<2),
    IBV_ACCESS_REMOTE_ATOMIC = (1<<3),
    IBV_ACCESS_MW_BIND = (1<<4),
    IBV_ACCESS_ZERO_BASED = (1<<5),
    IBV_ACCESS_ON_DEMAND = (1<<6)
};

struct ibv_mr *ibv_reg_mr(struct ibv_pd *pd, void *addr,
                           size_t length, int access);
```
```c
int main()
{
    struct ibv_odp_caps caps;
    ibv_mr *mr;
    struct ibv_sge sge;
    struct ibv_send_wr wr;
    ...
    if (ibv_query_odp_caps(ctx, &caps, sizeof(caps)) ||
        !(caps.rc_odp_caps & ODP_SUPPORT_SEND))
        return -1;
    mr = ibv_reg_mr(ctx->pd, NULL, -1, IBV_ACCESS_LOCAL_WRITE | IBV_ACCESS_ON_DEMAND);
    ...
    p = mmap(NULL, 10 * MB, PROT_READ | PROT_WRITE, MAP_SHARED, 0, 0);
    ...
    sge.addr = p;
    sge.lkey = mr->lkey;
    ibv_post_send(ctx->qp, &wr, &bad_wr);
    ...
    return 0;
}
```
MEMORY PREFETCHING

- **Best effort hint**
  - Not necessarily all pages are pre-fetched
  - No guarantees that pages remain resident
  - Asynchronous
    - Can be invoked opportunistically in parallel to IO

- **Use cases**
  - Avoid multiple page faults by small transactions
  - Pre-fault a large region about to be accessed by IO

- **EFAULT returned when**
  - Range exceeds the MR
  - Requested range not mapped to address space

```c
struct ibv_prefetch_attr {
    uint32_t comp_mask;
    int flags; /* IBV_ACCESS_LOCAL_WRITE */
    void *addr;
    size_t length;
};

int ibv_prefetch_mr(struct ibv_mr *mr,
                    struct ibv_prefetch_attr *attr,
                    size_t attr_size);
```
STATISTICS

- **Core statistics**
  - Maintained by the IB core layer
  - Tracked on a per device basis
  - Reported by sysfs

- **Use cases**
  - Page fault pattern
    - Warm-up
    - Steady state
  - Paging efficiency
  - Detect thrashing
  - Measure pre-fetch impact

```
/sys/class/InfiniBand_verbs/uverbs<dev-idx>/
  invalidations_faults_contentions
  num_invalidation_pages
  num_invalidations
  num_page_fault_pages
  num_page_faults
  num_prefetches_handled
```

<table>
<thead>
<tr>
<th>Counter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>invalidations_faults_contentions</code></td>
<td>Number of times that page fault events were dropped or prefetch operations were restarted due to OS page invalidations</td>
</tr>
<tr>
<td><code>num_invalidation_pages</code></td>
<td>Total number of pages invalidated during all invalidation events</td>
</tr>
<tr>
<td><code>num_invalidations</code></td>
<td>Number of invalidation events</td>
</tr>
<tr>
<td><code>num_page_fault_pages</code></td>
<td>Total number of pages faulted in by page fault events</td>
</tr>
<tr>
<td><code>num_page_faults</code></td>
<td>Number of page fault events</td>
</tr>
<tr>
<td><code>num_prefetches_handled</code></td>
<td>Number of prefetch Verb calls that completed successfully</td>
</tr>
</tbody>
</table>
STATISTICS (CONTINUED)

- **Driver debug statistics**
  - Maintained by the mlx5 driver
  - Tracked on a per device basis
  - Reported by debugfs

- **Use cases**
  - Track accesses to non-mapped memory
  - ODP MR usage

```
/sys/kernel/debug/mlx5/<pci-dev-id>/odp_stats/
  num_failed_resolutions
  num_mrs_not_found
  num_odp_mr_pages
  num_odp_mrs
```

<table>
<thead>
<tr>
<th>Counter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_failed_resolutions</td>
<td>Number of failed page faults that could not be resolved due to non-existing mappings in the OS</td>
</tr>
<tr>
<td>num_mrs_not_found</td>
<td>Number of faults that specified a non-existing ODP MR</td>
</tr>
<tr>
<td>num_odp_mr_pages</td>
<td>Total size in pages of current ODP MRs</td>
</tr>
<tr>
<td>num_odp_mrs</td>
<td>Number of current ODP MRs</td>
</tr>
</tbody>
</table>
## DEVELOPMENT

<table>
<thead>
<tr>
<th>Feature</th>
<th>Upstream</th>
<th>Mellanox OFED</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC Send-Receive, RDMA UD Send</td>
<td>3.19</td>
<td>2.3</td>
</tr>
<tr>
<td>Statistics</td>
<td>TBD</td>
<td>2.3</td>
</tr>
<tr>
<td>Pre-fetch</td>
<td>TBD</td>
<td>2.3</td>
</tr>
<tr>
<td>RC Atomics</td>
<td>4.11</td>
<td>3.4</td>
</tr>
<tr>
<td>Global MR</td>
<td>4.11</td>
<td>3.4</td>
</tr>
<tr>
<td>Memory Windows</td>
<td>TBD</td>
<td>3.4</td>
</tr>
<tr>
<td>DC Send, RDMA, Atomics</td>
<td>TBD</td>
<td>3.4</td>
</tr>
<tr>
<td>SRQ, DC Receive</td>
<td>TBD</td>
<td>Planned for 4.1</td>
</tr>
<tr>
<td>Huge-pages</td>
<td>TBD</td>
<td>Planned for 4.1</td>
</tr>
</tbody>
</table>
“Designing MPI library with on-demand paging (ODP) of InfiniBand: challenges and benefits”
• Mingzhe Li, Khaled Hamidouche, Xiaoyi Lu, Hari Subramoni, Jie Zhang, Dhabaleswar K. Panda; in proc. of SC’16
• Reported x11 reduction in memory footprint while matching pinned-buffer MPI performance

“Page Fault Support for Network Controllers”
• Ilya Lesokhin, Hagai Eran, Shachar Raindel, Guy Shapiro, Sagi Grimberg, Liran Liss, Muli Ben-Yehuda, Nadav Amit, Dan Tsafrir; to appear in ASPLOS’17
• We evaluate ODP contribution for HPC, Storage, and user-level TCP
Added OpenMPI ODP support
- Implemented in MXM library
- Removed 10K LOC of pin-down cache (!)

Benchmarks
- IMB application suite
- B_eff

Same performance as a state-of-the-art pinned 0-copy implementation

Ref: “Page Fault Support for Network Controllers”, ASPLOS’17
- **Standard iSER initiator**
  - Stock Linux kernel

- **Modified iSER target**
  - Based on open-source tgt project
  - Minor code modifications to register staging buffer as ODP
    - 10’s LOCs

- **fio benchmark**
  - Random 64KB / 512KB reads
Avoid worst-case pinning!

Ref: “Page Fault Support for Network Controllers”, ASPLOS’17
 Added ODP support for lwIP TCP stack using Raw Ethernet QPs
  - Mimics VMs with PCI pass-through

 Evaluated memcached server performance
  - Measured memaslap Get() hits-per-second

<table>
<thead>
<tr>
<th>memcached instances</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPF</td>
<td>186</td>
<td>311</td>
<td>407</td>
<td>484</td>
</tr>
<tr>
<td>pinning</td>
<td>185</td>
<td>310</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Ref: “Page Fault Support for Network Controllers”, ASPLOS’17
WHAT'S NEXT?

- **MPI support for ODP**
  - Already integrated into OSU MVAPICH
  - UCX integration in Q2'17

- **Accelerate RDMA adoption through ODP**
  - Efficient RDMA support for managed runtimes, e.g., Java, Python, Go
  - CEPH, Hadoop, Spark, etc.

- **Enable GPU direct with shared CPU-GPU address spaces**
  - RDMA into pages that migrate between CPU and GPU

- **RDMA and PMEM**
  - No need to pre-register huge PMEM file mappings

- **RDMA and coherent accelerators**
  - Accelerators will share a process address space
  - Allow RDMA into and out of accelerator buffers
THANK YOU

Liran Liss

Mellanox Technologies