Interconnect Your Future

Paving the Road to Exascale

OpenSHMEM Workshop 2017
Exponential Data Growth – The Need for Intelligent and Faster Interconnect

CPU-Centric (Onload)

Data-Centric (Offload)

Must Wait for the Data Creates Performance Bottlenecks

Process Data as it Moves!

Faster Data Speeds and In-Network Computing Enable Higher Performance and Scale
Data Centric Architecture to Overcome Latency Bottlenecks

**CPU-Centric (Onload)**

**Data-Centric (Offload)**

HPC / Machine Learning Communications Latencies of 30-40us

In-Network Computing

HPC / Machine Learning Communications Latencies of 3-4us

Intelligent Interconnect Paves the Road to Exascale Performance
In-Network Computing to Enable Data-Centric Data Center

In-Network Computing Key for Highest Return on Investment
Highest-Performance 100/200Gb/s Interconnect Solutions

**Adapters**

- ConnectX-6: 200Gb/s Adapter, 0.6us latency
- 200 million messages per second
- (10 / 25 / 40 / 50 / 56 / 100 / 200Gb/s)

**Switch**

- Quantum: 40 HDR (200Gb/s) InfiniBand Ports
- 80 HDR100 InfiniBand Ports
- Throughput of 16Tb/s, <90ns Latency

**Switch**

- Spectrum: 32 100GbE Ports, 64 25/50GbE Ports
- (10 / 25 / 40 / 50 / 100GbE)
- Throughput of 3.2Tb/s

**Interconnect**

- Transceivers
  - Active Optical and Copper Cables
  - (10 / 25 / 40 / 56 / 100 / 200Gb/s)

- VCSELs, Silicon Photonics and Copper

**Software**

- HPC-X™
  - MPI, SHMEM/PGAS, UPC
  - For Commercial and Open Source Applications
  - Leverages Hardware Accelerations
Quantum 200G HDR InfiniBand Smart Switch

40 Ports of 200G HDR InfiniBand
80 Ports of 100G HDR100 InfiniBand
Switch System 800 Ports 200G, 1600 Ports 100G

16Tb/s Switch Capacity
Extremely Low Latency of 90ns
15.6 Billion Messages per Second

In-Network Computing (SHARP Technology)
Flexible Topologies (Fat-Tree, Torus, Dragonfly, etc.)
Advanced Adaptive Routing
ConnectX-6 200G HDR InfiniBand and Ethernet Smart Adapter

100/200Gb/s Throughput
0.6usec End-to-End Latency
175/200M Messages per Second

PCle Gen3 and Gen4
Integrated PCle Switch and Multi-Host Technology
Advanced Adaptive Routing

In-Network Computing (Collectives, Tag Matching)
In-Network Memory
Storage (NVMe), Security and Network Offloads
Mellanox’s OpenSHMEM Support
Mellanox’s OpenSHMEM Support: Status Update

- Major focus: Utilize Mellanox hardware capabilities to accelerate the OSHMEM implementation
  - Offload Network management
  - RDMA
  - Hardware atomic support
  - Dynamically Connected Transport
  - SHARP
  - DC/RC adaptive routing support
  - On Demand Paging
  - SHIELD

- V1.3 OpenSHMEM SPEC fully support
- SHMEM big job start improvements with PMIx
- OpenSHMEM collectives
  - SHARP supported (HCOLL)
  - OpenSHMEM shared memory collectives
- UCX support in spml UCX
- Data path performance improvements with direct verbs in UCX
  - Includes atomic support
- Added support for transparent huge page allocation
- Added support for shmem pointer
- Active participants in the OpenSHMEM community
  - Working groups of greatest interest
    - Collectives
    - Threading support
SHIELD
SHIELD – Self Healing Interconnect Technology

5000X Faster Network Recovery
Enable Unbreakable Data Centers
The SHIELD Self Healing Interconnect Technology

- Software-based solutions for network failures create long delays: 5-30 seconds for 1K to 10K node clusters
- During software-based recovery time, data can be lost, applications can fail
- Adaptive Routing creates further issues (failing links may act as “black holes”)
- Mellanox SHIELD technology is an innovative hardware-based solution
- SHIELD technology enables the generation of Self-Healing Interconnect
- The ability to overcome network failures by the network intelligent devices
- Accelerates network recovery time by 5000X
- Highly scalable and available for EDR and HDR solutions and beyond

Self-Healing Network Enables Unbreakable Data Centers
On Demand Paging and OSHMEM Startup Optimization
The SHMEM startup problem

- **Goal**: Reduce SHMEM startup time
- Since RDMA is used, the global address space has to be registered
  - memheap + global data
- Heap registration is the dominant factor of shmem_init() time
  - Pinning and clearing the pages
  - Writing translations to the HCA
- Can take few seconds or more for large heaps
- Need to reduce memory registration time
Memory Pinning: Additional Costs

- 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
On-Demand Paging (ODP)

- 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
The ODP promise:
IO virtual address mapping == Process virtual address mapping
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 (!)

```c
mr = ibv_reg_mr(pd, NULL, -1, IBV_ACCESS_LOCAL_WRITE | IBV_ACCESS_ON_DEMAND);
```
ODP: Memory Prefetching

- Pre-faults (populates) ODP MRs

- 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);
```
Register the heap with ODP instead of pinning
- Completes much faster than traditional registration
- `shmembinit()` time is reduced significantly

Optimization: background ODP-prefetch
- High rate of ODP faults may affect runtime performance
- Driver maps pages in the background while application continues to run
- A page would mostly likely be mapped before accessed by the application
- Small amount of page faults in the common case

Application Flow
- Application allocates memory with `shmalloc()`
- SHMEM asks UCX to prefetch the allocated memory
  - Calling `ucp_mem_order(UCP_MADV_WILLNEED)`
- UCX issues `ibv_exp_prefetch_mr()` verb
- Driver schedules background work to:
  - `get_user_pages()` on the allocated memory
  - Issue UMR to update HW translation tables
### Performance results

- **Measuring shmem_init() time for an applications which uses 180G per node**
  - Skipping few warmup iterations

- **Setup details:**
  - 64 nodes, 32 processes per node
  - Intel(R) Xeon(R) CPU E5-2697A v4 @ 2.60GHz
  - DDR4 2400 MHz
  - ConnectX-4 EDR, Fat tree

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Default startup time (seconds)</th>
<th>ODP-optimized startup time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.308</td>
<td>2.209</td>
</tr>
<tr>
<td>2</td>
<td>5.287</td>
<td>2.205</td>
</tr>
<tr>
<td>3</td>
<td>5.244</td>
<td>2.228</td>
</tr>
<tr>
<td>4</td>
<td>5.235</td>
<td>2.359</td>
</tr>
<tr>
<td>5</td>
<td>5.507</td>
<td>2.287</td>
</tr>
<tr>
<td>6</td>
<td>5.321</td>
<td>2.086</td>
</tr>
<tr>
<td>7</td>
<td>5.282</td>
<td>2.164</td>
</tr>
<tr>
<td>8</td>
<td>5.263</td>
<td>2.266</td>
</tr>
<tr>
<td>9</td>
<td>5.273</td>
<td>2.283</td>
</tr>
<tr>
<td>10</td>
<td>5.222</td>
<td>2.242</td>
</tr>
</tbody>
</table>
Alltoall – Small data Aggregation
Algorithm Overview

- Newly developed algorithm – aggregation based (Patent pending)
- Use hardware scatter/gather to reduce memory copies
All-to-All latencies – 8 bytes

- 30 nodes, 20 processes per node

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open MPI - Pairwise</td>
<td>202.35</td>
</tr>
<tr>
<td>Open MPI - Bruck</td>
<td>175.11</td>
</tr>
<tr>
<td>Open MPI – Bruck + Shared memory</td>
<td>105.44</td>
</tr>
<tr>
<td>IntelMPI + Shared Memory</td>
<td>117.39</td>
</tr>
<tr>
<td>New Algorithm</td>
<td>95.98</td>
</tr>
</tbody>
</table>
Broadcast Collective Operation Optimization
Key Features

- Hardware multicast: capability to replicate packets at the switch – Order (1) cost
- However “unreliable”, needs reliability protocol.
- Attempted before: always duplicating data through reliable channel – either before [1], or after [2] the multicast.
- Target: No duplication, O(1) good case, O(logF) for F failures.


64 Processes, Varying Data Size

![Graph showing time per broadcast (usec) vs message size (bytes) for different MPI implementations. The graph indicates that vBcast, HPCX-OMPI, IntelMPI, and MVAPICH2 have varying performance depending on the message size.](image)
32KB Varying Number of Processes

![Graph showing time per broadcast (usec) against number of processes for different MPI implementations: vBcast, HPCX-OMPI, IntelMPI, and MVAPICH2.](image)
1MB Varying Number of Processes

![Graph showing time per broadcast (usec) varying with the number of processes for different MPI implementations: vBcast, HPCX-OMPI, IntelMPI, and MVAPICH2.](image)
Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)
Compute in the Interior of the network
Feature Description

- Reliable Scalable General Purpose Primitive, Applicable to Multiple Use-cases
  - In-network Tree based aggregation mechanism
  - Large number of groups
  - Many simultaneous outstanding operations in flight

Accelerating HPC applications

- Scalable High Performance Collective Offload
  - Barrier, Reduce, All-Reduce
  - Sum, Min, Max, Min-loc, max-loc, OR, XOR, AND
  - Integer and Floating-Point
  - Repeatable results

- Significantly reduce MPI collective runtime
- Increase CPU availability and efficiency
- Enable communication and computation overlap
SHARP Allreduce Performance Advantages

SHARP enables 75% Reduction in Latency
Providing Scalable Flat Latency
Thank You