Designs for Advanced Collectives and Performance

Dhabaleswar K. (DK) Panda
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda
InfiniBand Multicast Example

Switch decodes inbound packet header (LRH) DLID to determine target output ports.

Router decodes inbound packet header (GRH) GID multicast address to determine target out-
Hardware Multicast-aware MPI_Bcast on Broadwell + EDR

Small Messages (1,120 Cores)

Latency (us)

Message Size (Bytes)

Large Messages (1,120 Cores)

Latency (us)

Message Size (Bytes)

16 Byte Message

Latency (us)

Number of Nodes

32 KByte Message

Latency (us)

Number of Nodes

ConnectX-4 EDR (100 Gbps): 2.4 GHz Fourteen-core (Broadwell) Intel with Mellanox IB (EDR) switches
• MCAST-based designs improve latency of MPI_Bcast by up to **2X at 2,048 nodes**

• Use MV2_USE_MCAST=1 to enable MCAST-based designs
Collective Offload Support on the Adapters

• Performance of collective operations (broadcast, barrier, reduction, all-reduce, etc.) are very critical to the overall performance of MPI applications

• Currently being done with basic pt-to-pt operations (send/recv and RDMA) using host-based operations

• Mellanox ConnectX-2, ConnectX-3, ConnectX-4, and ConnectX-5 adapters support offloading some of these operations to the adapters (CORE-Direct)
  – Provides overlap of computation and collective communication
  – Reduces OS jitter (since everything is done in hardware)
One-to-many Multi-Send

- Sender creates a task-list consisting of only send and wait WQEs
  - One send WQE is created for each registered receiver and is appended to the rear of a singly linked task-list
  - A wait WQE is added to make the HCA wait for ACK packet from the receiver
Scalable Hierarchical Aggregation Protocol (SHArP)

- Management and execution of MPI operations in the network by using SHArP
  - Manipulation of data while it is being transferred in the switch network
- SHArP provides an abstraction to realize the reduction operation
  - Defines Aggregation Nodes (AN), Aggregation Tree, and Aggregation Groups
  - AN logic is implemented as an InfiniBand Target Channel Adapter (TCA) integrated into the switch ASIC*
  - Uses RC for communication between ANs and between AN and hosts in the Aggregation Tree*

Performance of Collectives with SHARP on TACC Frontera

Optimized SHARP designs in MVAPICH2-X

Up to 9X performance improvement with SHARP over MVAPICH2-X default for 1ppn MPI_Barrier, 6X for 1ppn MPI_Reduce and 5X for 1ppn MPI_Allreduce


Optimized Runtime Parameters: MV2_ENABLE_SHARP = 1
Benefits of SHARP Allreduce at Application Level

Avg DDOT Allreduce time of HPCG

<table>
<thead>
<tr>
<th>Latency (seconds)</th>
<th>(4,28)</th>
<th>(8,28)</th>
<th>(16,28)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVAPICH2</td>
<td>0.17</td>
<td>0.19</td>
<td>0.21</td>
</tr>
<tr>
<td>12%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SHARP support available since MVAPICH2 2.3a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_ENABLE_SHARP=1</td>
<td>Enables SHARP-based collectives</td>
<td>Disabled</td>
</tr>
<tr>
<td>--enable-sharp</td>
<td>Configure flag to enable SHARP</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

- Refer to Running Collectives with Hardware based SHARP support section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3-userguide.html#x1-990006.26](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3-userguide.html#x1-990006.26)
• Different cores in a NUMA platform have different communication costs.
Impact of NUMA on Inter-node Latency

- Cores in Socket 0 (closest to network card) have lowest latency
- Cores in Socket 1 (one hop from network card) have highest latency

ConnectX-4 EDR (100 Gbps): 2.4 GHz Fourteen-core (Broadwell) Intel with IB (EDR) switches
Impact of NUMA on Inter-node Bandwidth

- NUMA interactions have significant impact on bandwidth

ConnectX-4 EDR (100 Gbps): 2.4 GHz Fourteen-core (Broadwell) Intel with IB (EDR) switches
ConnectX-2-QDR (36 Gbps): 2.5 GHz Hex-core (MagnyCours) AMD with IB (QDR) switches
For MPI_Allreduce latency with 32K bytes, MVAPICH2-OPT can reduce the latency by 2.4X.


Available since MVAPICH2-X 2.3b
Optimized CMA-based Collectives for Large Messages

- Significant improvement over existing implementation for Scatter/Gather with 1MB messages (up to 4x on KNL, 2x on Broadwell, 14x on OpenPower)
- New two-level algorithms for better scalability
- Improved performance for other collectives (Bcast, Allgather, and Alltoall)


Available since MVAPICH2-X 2.3b
Enhanced MPI_Bcast with Optimized CMA-based Design

- Up to \(2x - 4x\) improvement over existing implementation for 1MB messages
- Up to \(1.5x – 2x\) faster than Intel MPI and Open MPI for 1MB messages
- Improvements obtained for large messages only
  - \(p-1\) copies with CMA, \(p\) copies with Shared memory
  - Fallback to SHMEM for small messages

Support is available in MVAPICH2-X 2.3b

---

Shared Address Space (XPMEM)-based Collectives Design

OSU_Allreduce (Broadwell 256 procs)

- MVAPICH2-2.3b
- IMPI-2017v1.132
- MVAPICH2-X-2.3rc1

OSU_Reduce (Broadwell 256 procs)

- MVAPICH2-2.3b
- IMPI-2017v1.132
- MVAPICH2-2.3rc1

- "Shared Address Space"-based true zero-copy Reduction collective designs in MVAPICH2
- Offloaded computation/communication to peers ranks in reduction collective operation
- Up to 4X improvement for 4MB Reduce and up to 1.8X improvement for 4M AllReduce


Available in MVAPICH2-X 2.3rc1
Application-Level Benefits of XPMEM-Based Collectives

- Up to **20%** benefits over IMPI for CNTK DNN training using AllReduce
- Up to **27%** benefits over IMPI and up to **15%** improvement over MVAPICH2 for MiniAMR application kernel
Optimized All-Reduce with XPMEM on OpenPOWER

- Optimized MPI All-Reduce Design in MVAPICH2
  - Up to 2X performance improvement over Spectrum MPI and 4X over OpenMPI for intra-node

Optimized Runtime Parameters: MV2_CPU_BINDING_POLICY=hybrid MV2_HYBRID_BINDING_POLICY=bunch
Network-Topology-Aware Placement of Processes

Can we design a highly scalable network topology detection service for IB?

How do we design the MPI communication library in a network-topology-aware manner to efficiently leverage the topology information generated by our service?

What are the potential benefits of using a network-topology-aware MPI library on the performance of parallel scientific applications?

Overall performance and Split up of physical communication for MILC on Ranger

Performance for varying system sizes

- Reduce network topology discovery time from $O(N^2_{hosts})$ to $O(N_{hosts})$
- 15% improvement in MILC execution time @ 2048 cores
- 15% improvement in Hypre execution time @ 1024 cores

Default for 2048 core run

Topo-Aware for 2048 core run

MPI + CUDA - Naive

• Data movement in applications with standard MPI and CUDA interfaces

At Sender:
cudaMemcpy(s_hostbuf, s_devbuf, ...);
MPI_Send(s_hostbuf, size, ...);

At Receiver:
MPI_Recv(r_hostbuf, size, ...);
cudaMemcpy(r_devbuf, r_hostbuf, ...);

High Productivity and Low Performance
MPI + CUDA - Advanced

- Pipelining at user level with non-blocking MPI and CUDA interfaces

At Sender:

```c
for (j = 0; j < pipeline_len; j++)
    cudaMemcpyAsync(s_hostbuf + j * blk, s_devbuf + j * blksz, ...);
for (j = 0; j < pipeline_len; j++) {
    while (result != cudaSuccess) {
        result = cudaStreamQuery(...);
        if (j > 0) MPI_Test(...);
    }
    MPI_Isend(s_hostbuf + j * block_sz, blksz, ...);
}
MPI_Waitall();
```

<<Similar at receiver>>

Low Productivity and High Performance
GPU-Aware (CUDA-Aware) MPI Library: MVAPICH2-GPU

- Standard MPI interfaces used for unified data movement
- Takes advantage of Unified Virtual Addressing (>= CUDA 4.0)
- Overlaps data movement from GPU with RDMA transfers

At Sender:

`MPI_Send(s_devbuf, size, …);`

At Receiver:

`MPI_Recv(r_devbuf, size, …);`

*High Performance and High Productivity*
Optimized MVAPICH2-GDR Design

**GPU-GPU Inter-node Latency**

- **MV2-(NO-GDR)**
- **MV2-GDR-2.3a**

**GPU-GPU Inter-node Bandwidth**

- **MV2-(NO-GDR)**
- **MV2-GDR-2.3a**

Intel Haswell (E5-2687W @ 3.10 GHz) node - 20 cores
NVIDIA Volta V100 GPU
Mellanox Connect-X4 EDR HCA
CUDA 9.0
Mellanox OFED 4.0 with GPU-Direct-RDMA
**MVAPICH2-GDR ROCm Support for AMD GPUs**

**Intra-Node Point-to-Point Latency**

-**Allreduce** – 64 GPUs (8 nodes, 8 GPUs Per Node)
  - MVAPICH2-GDR: 1.77 us
  - OpenMPI 4.1.0 + UCX 1.9.0: 56.54 us

-**Bcast** – 64 GPUs (8 nodes, 8 GPUs Per Node)
  - MVAPICH2-GDR: 16.54 ms
  - OpenMPI 4.1.0 + UCX 1.9.0: 56.54 ms

**Inter-Node Point-to-Point Latency**

-**Corona Cluster - ROCm-3.9.0 (mi50 AMD GPUs)**
  - MVAPICH2-GDR: 3.44 ms
  - OpenMPI 4.1.0 + UCX 1.9.0: 3310 ms

Available with MVAPICH2-GDR 2.3.5
MVAPICH2 (MPI)-driven Infrastructure for ML/DL Training

ML/DL Applications

- TensorFlow
- PyTorch
- MXNet

Horovod

MVAPICH2 or MVAPICH2-X for CPU Training
MVAPICH2-GDR for GPU Training

PyTorch

Torch.distributed
DeepSpeed

MVAPICH2 or MVAPICH2-X for CPU Training
MVAPICH2-GDR for GPU Training

More details available from: http://hidl.cse.ohio-state.edu
Optimized designs in MVAPICH2-GDR offer better/comparable performance for most cases.

MPI_Allreduce (MVAPICH2-GDR) vs. ncclAllreduce (NCCL2) on 1 DGX-2 node (16 Volta GPUs)

Platform: Nvidia DGX-2 system (16 Nvidia Volta GPUs connected with NVSwitch), CUDA 10.1

~2.5X better

~4.7X better

MVAPICH2-GDR: MPI_Allreduce at Scale (ORNL Summit)

- Optimized designs in MVAPICH2-GDR offer better performance for most cases
- MPI_Allreduce (MVAPICH2-GDR) vs. ncclAllreduce (NCCL2) up to 1,536 GPUs

Platform: Dual-socket IBM POWER9 CPU, 6 NVIDIA Volta V100 GPUs, and 2-port InfiniBand EDR Interconnect

Scalable TensorFlow using Horovod and MVAPICH2-GDR

- ResNet-50 Training using TensorFlow benchmark on 1 DGX-2 node (16 Volta GPUs)

Platform: Nvidia DGX-2 system, CUDA 10.1

Scaling Efficiency = \frac{Actual throughput}{Ideal throughput at scale} \times 100%

Distributed TensorFlow on ORNL Summit (1,536 GPUs)

- ResNet-50 Training using TensorFlow benchmark on SUMMIT -- 1536 Volta GPUs!
- 1,281,167 (1.2 mil.) images
- Time/epoch = 3 seconds
- Total Time (90 epochs) = 3 x 90 = 270 seconds = 4.5 minutes!

*We observed issues for NCCL2 beyond 384 GPUs

Platform: The Summit Supercomputer (#2 on Top500.org) – 6 NVIDIA Volta GPUs per node connected with NVLink, CUDA 10.1
Scaling PyTorch on ORNL Summit using MVAPICH2-GDR

- ResNet-50 training using PyTorch + Horovod on Summit
  - Synthetic ImageNet dataset
  - Up to 256 nodes, 1536 GPUs
- MVAPICH2-GDR can outperform NCCL2
  - Up to 30% higher throughput
- CUDA 10.1 cuDNN 7.6.5
  PyTorch v1.5.0 Horovod v0.19.1


Platform: The Summit Supercomputer (#2 on Top500.org) – 6 NVIDIA Volta GPUs per node connected with NVLink, CUDA 10.1
PyTorch at Scale: Training ResNet-50 on 256 V100 GPUs

- Training performance for 256 V100 GPUs on LLNL Lassen
  - ~10,000 Images/sec faster than NCCL training!

<table>
<thead>
<tr>
<th>Distributed Framework</th>
<th>Torch.distributed</th>
<th>Horovod</th>
<th>DeepSpeed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images/sec on 256 GPUs</td>
<td>61,794</td>
<td>72,120</td>
<td>74,063</td>
</tr>
<tr>
<td>Communication Backend</td>
<td>NCCL</td>
<td>MVAPICH2-GDR</td>
<td>NCCL</td>
</tr>
</tbody>
</table>
Dask Architecture

Dask
- Dask Bag
- Dask Array
- Dask DataFrame
- Delayed
- Future

Task Graph

Distributed
- Scheduler
- Worker
- Client

Comm Layer
- tcp.py
- ucx.py

High Performance Computing Hardware
- TCP
- UCX

Collectives
- Dask-MPI
- Dask-CUDA
- Dask-Jobqueue

Dask
- Dask Bag
- Dask Array
- Dask DataFrame
- Delayed
- Future

Network Based Computing Laboratory
Benchmark #1: Sum of cuPy Array and its Transpose (RI2)

Benchmark #1: Sum of cuPy Array and its Transpose (TACC Frontera GPU Subsystem)

1.71x better on average

A. Shafi, J. Hashmi, H. Subramoni, and D. K. Panda, Efficient MPI-based Communication for GPU-Accelerated Dask Applications,
https://arxiv.org/abs/2101.08878

MPI4Dask 0.1 release
(http://hibd.cse.ohio-state.edu)
Benchmark #2: cuDF Merge (TACC Frontera GPU Subsystem)

2.91x better on average

2.90x better on average


MPI4Dask 0.1 release
(http://hibd.cse.ohio-state.edu)
Accelerating cuML with MVAPICH2-GDR

- Utilize MVAPICH2-GDR (with mpi4py) as communication backend during the training phase (the fit() function) for Multi-node Multi-GPU (MNMG) setting over cluster of GPUs

- Communication primitives:
  - Allreduce
  - Reduce
  - Broadcast

- Exploit optimized collectives