OpenSHMEM on ConnectX-6 and Mellanox SHARP

Manjunath Gorentla Venkata
On behalf of OpenSHMEM members @ Mellanox

OpenSHMEM BoF @ SC19, November 19th, 2019
Higher Data Speeds
Faster Data Processing
Better Data Security

Adapters
Switches
Cables & Transceivers

SmartNIC
System on a Chip
InfiniBand Technology and Solutions

- 200G HDR InfiniBand end-to-end, extremely low latency, high message rate, RDMA and GPUDirect
- Advanced end-to-end adaptive routing, congestion control and quality of service
- In-Network Computing acceleration engines (Mellanox SHARP, MPI offloads)
- Self Healing Network with SHIELD for highest network resiliency
- High speed gateways to Ethernet, and long reach up to 40Km
- Standard, backward and forward compatibility
Accelerating OpenSHMEM with Mellanox Hardware and Software

**Hardware**

- High throughput and message rate
  - Dual ports of 200Gb/s VPI Adapter (HDR)
  - Message rate: 200 million messages per sec
  - Latency: 0.6usec
- In-Network Computing
  - SHARP v2 (Switch Hierarchical Aggregation Protocol) support
  - Enables offloading high bandwidth collective / reduce operations
- Atomic Operations enhancements
  - XOR, AND, OR, ADD, MIN, MAX
  - PCIe atomics
  - MEMIC memory for atomics and counters

**Software**

- OSHMEM: OpenSHMEM implementation in Open MPI
  - Supports OpenSHMEM 1.4 and some OpenSHMEM 1.5 features
  - Leverages UCX and HCOLL
  - Supports Intel, Arm and POWER architectures
  - Supports ConnectX-4, ConnectX-5, and ConnectX-6
  - The production quality implementation used by Mellanox, Arm, HPE, Cray and other vendors

- Mellanox vendor tests
  - Opensource and available on the community on GitHub
  - Used for verifying the OpenSHMEM implementations

- Nightly functionality and performance testing

**Specification community engagement**

- Committee and working group meetings
- Engage with Users
OSHMEM Software Stack

- OPAL
- ORTE
- OMPI
- Open MPI
- OSHMEM
  - Atomics
  - Collectives
  - Heap
  - SPML

MCA - Modular Component Architecture

OPAL and ORTE Components

- Basic
- UCX
- MXM
- MP1
- BUDDY
- PMALLOC
- IKRIT

Other Components

- uGNI
- Vader
- SM
- Ethernet
- HCOLL
- UCX
- SM
- Ethernet
HCOLL: A high-performing Collectives Library

Fig. 2: HCOLL components to realize hierarchical collective implementation and customizing the implementation to various communication mechanisms in the system. To understand the design, consider the implementation of shmem reductions using HCOLL. The shmem reductions are implemented by combining reduction, allreduce and a broadcast primitive. The primitives are not necessarily OpenSHMEM compliant, however, the composition of these primitives is OpenSHMEM compliant.

The various components in the HCOLL are shown in Figure 2. It includes components Messaging Layer (ML), Subgroup, and Basic Collective (BCOL). The ML layer is responsible for orchestrating the collective primitives. The subgrouping component is responsible for grouping the processes PE in the job into multiple subgroups based on the communication mechanism shared among them. The BCOL components provide the basic collective primitives, which are then combined to provide the implementation for OpenSHMEM collective operations.

Though the HCOLL library accelerates all collective operations typically used in the parallel programming models and defined by the MPI and OpenSHMEM specification, we focus here on the shmem barrier, shmem float sum to all, and shmem broadcast32 operations. For these operations, a variety of collective algorithms are implemented including hierarchical and non-hierarchical n-ary, recursive doubling, and recursive k-ing algorithms. The collective algorithm and number of hierarchies could be selected either at compile time or runtime based on the system architecture or communication characteristics. For this paper, we use hierarchical recursive doubling and configure with two hierarchies (intra-node and inter-node). In this configuration, we designate a PE on each node as a leader process, which is responsible for communication between the nodes.

C. OpenSHMEM Collectives using SHARP and HCOLL: The OpenSHMEM collective operations in OSHMEM can use either no-acceleration, software accelerated, or hardware- and software-accelerated (In-network Computing) paths.

Non-root OpenSHMEM PE
Intra-node path (lower latency)
Inter-node path (higher latency)
Root OpenSHMEM PE

Fig. 3: No acceleration approach: High-latency paths taken for shmem broadcast32 depends on the topology and algorithm chosen.

Non-leader OpenSHMEM PE
Intra-node path (lower latency)
Inter-node path (higher latency)
Root OpenSHMEM PE

Fig. 4: Software acceleration approach: The number of high-latency paths taken for shmem broadcast32 is proportional to \( \log(\text{number of nodes}) \).

Switch
Non-root OpenSHMEM PE
Intra-node path (lower latency)
Inter-node path (higher latency)
Root OpenSHMEM PE

Fig. 5: In-network Computing: The number of high-latency paths taken for shmem broadcast32 is proportional to the number of layers of switches in the path of the operation.
Experiment Testbed

- **Software**
  - OSHMEM – Master
  - UCX - Master
  - HCOLL from HPC-X v2.4

- **Benchmarks**
  - OSU OpenSHMEM benchmarks

- **Hardware**
  - Dual Socket Intel® Xeon® 16-core CPUs E5-2697A V4 @ 2.60 GHz
  - Mellanox ConnectX-6 HDR100 100Gb/s InfiniBand adapters
  - Mellanox Switch-IB 2 SB7800 36-Port 100Gb/s EDR InfiniBand switches
  - Memory: 256GB DDR4 2400MHz RDIMMs per node
OpenSHMEM: Put and Get Latency and Overlap

- Put and Get latency below 1.5 usecs when SHMEM_THREAD_MULTIPLE enabled
- Overlap is close to 100%
- Wait time is close to 0%
OpenSHMEM on ConnectX-6: Bandwidth

OpenSHMEM Put Mem Bandwidth (MB/s)
AMD Rome / PCIe 4 and ConnectX-6

Bandwidth (MB/s)

Message Size (Bytes)
OpenSHMEM on ConnectX-6: Bi-directional Message Rate
Experimental Setup

- The Summit system at ORNL
  - 4608 compute nodes
  - Each Node
    - Two IBM POWER9 processors and Six NVIDIA Volta GPUs
    - Mellanox’s ConnectX-5 EDR 100 Gb/s HCAs
    - Switch-IB 2 InfiniBand switches, which supports SHARP

- Software
  - Open MPI 4.0.0
  - HCOLL
  - SHARP software

- Benchmarks
  - OSU Microbenchmarks: Invoke collective operation in a tight loop
  - 2D-Heat application kernel
Barrier using the In-Network Computing Approach has better Performance and Scaling Characteristics

- Barrier implemented using the In-Network Computing approach is 710% when compared with a no acceleration approach
  - Acceleration is a result of using software and hardware mechanisms

- The software-accelerated approach based barrier is 200% faster when compared with a no acceleration approach
  - Acceleration is a result of hierarchical approach to implementing collective operations

- The performance changes only 25% when increasing the problem size from 40 to 5120 PEs
  - With no-acceleration it increases by 425%
Reductions using the In-Network Computing Approach has better Performance and Scaling Characteristics

![Graph showing latency for different message sizes and PEs](image)

![Graph showing completion time for 2D-Heat application kernel](image)
In-Network Computing Accelerates 2D Heat Kernel

Kernel models 2D heat conduction using Grass-Siedel method

- Barriers are used for synchronization and reductions are used for exchanging intermediate results
- The workload is dominated by shmem_put

For 5120 PE problem configuration, the In-Network computing approach implementation improves performance by 13%
Thank You