Modern HPC clusters have nodes that incorporate large amounts of shared memory cores and powerful network adapters. The efficiency of hybrid programming models are critical for scientific applications to reach desired levels of scalability on those machines.

In this work, we combine API extensions in distributed programming models that enable fine-grained control of communication in multi-threaded codes, and an implementation of those APIs that utilizes a low-latency network library which is designed to drive the network to its full potential using multiple CPUs. The resulting hybrid microbenchmarks achieved up to 64 times higher communication throughput when compared to the traditional implementation, for small to medium message sizes and atomic operations.

OpenUCX provides workers objects that act as network injection points and communication managers. Workers’ task matching spaces are independent of each other, as well as their Send/Recv/RDMA progression and completion environments. Being able to select the optimized backends for each type of interconnect, the workers can effectively utilize hardware resources and thus have performance comparable to individual MPI ranks, even if they are created inside a single process.

Instead of letting all threads share a single worker like in the mainstream implementations, each thread will now use a private worker. To evaluate this new approach, we have created several multi-thread microbenchmarks that perform different communication operations, and recorded the average latencies. Up to 64x speedup is achieved for 24 threads.

Our work will be integrated into implementations of distributed programming models in the following ways:

- OpenSHMEM Contexts: Containers that define independent ordering and completion environments that can be used to manage the communication operations performed by separate threads within a multi-threaded PE.
- MPI Endpoints: Ongoing MPI proposal that adds the ability to create sub-ranks within ranks, allowing threads to be addressed directly.

Related Works

- Intel Omni-Path multi-EP [2][3]: Starting from Intel MPI release 2019, MPI_THREAD_SPLIT is added to allow threads to use the multiple network injection points in the Omni-Path architecture. This approach is very restrictive in which sets of threads can communicate with each other.
- Single MPI process offloading by Karthikeyan Vaidyanathan, Dhiraj Kalamkar, et al [5]. This work reduced MPI library call overhead by offloading all MPI calls to a single process, so the threads can go back to computation immediately. It does not eliminate library internal overhead.

Future Work

- Translate this performance gain to MPI/OpenSHMEM implementations. Mutexes that were used to protect the internal data structures need to be removed, and the levels of indirections can be reduced.
- Improve the GPU support of distributed programming models by exploring the GPU allocated memory support in OpenUCX. Coordinate the communication between threads, CUDA/ROCm kernels and remote nodes more efficiently.

References

1. OpenMP Tasking and MPI in a Lattice QCD Benchmark, Larry Meadows and Ken-ichi Ishikawa, IWOMP 2017
2. Multiple Endpoints for Improved MPI Performance on a Lattice QCD Code, Larry Meadows, Kenichi Ishikawa, Taisuke Boku, Masashi Horikoshi, HPC ASIA 18
6. OpenUCX. https://www.openucx.org/