Efficient Implementation of MPI-3 RMA over OpenFabrics Interfaces

Hajime Fujita, Chongxiao Cao, Sayantan Sur, Charles Archer, Erik Paulson, Maria Garzaran
Intel Corporation
Contact Address: hajime.fujita@intel.com

Extended Abstract

MPI-3 has a set of features to support one-sided communication, also known as Remote Memory Access (RMA) [2]. Unlike the traditional two-sided message passing operations, such as send/receive, RMA does not require the destination to be involved in the communication. In this talk, we describe the design and the implementation of MPI-3 RMA in MPICH-OFI*, which is the MPICH-based open source implementation of the MPI standard that uses the OpenFabrics Interfaces* (OFI*) to communicate with the underlying network fabric.

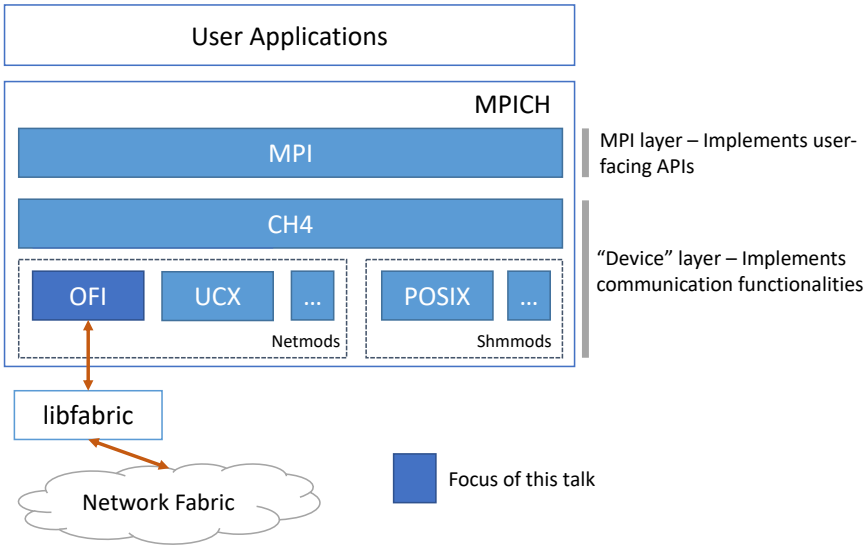

Figure 1: Internal Structure of MPICH, Highlighting the MPICH-OFI Configuration with OFI.

Figure 1 shows the internal structure of MPICH. MPI function calls from applications are first handled by the MPI later, where simple error checks and handle translations are performed. Then, the control flow goes to CH4 [4], which implements the communication layer in MPICH and that has been designed from the ground up to leverage modern computer and network architectures. It contains multiple different network modules ("netmods") and shared memory modules ("shmmods") to support many different communication network fabrics and memory architectures. Compared to its predecessor CH3, CH4 has the following novel features: 1) significant lower overhead due to a dramatic reduction in instruction count through function inlining, avoidance of indirect jumps such as function pointers, and elimination of branches, and 2) new design that allows netmods/shmmods to directly handle most of the MPI functions, to exploit advanced features of the underlying network fabrics whenever available.

OpenFabrics Interfaces (OFI) [5] provide a common interface for modern high-speed interconnect networks. It supports various communication functions useful for implementing MPI, including two-sided MPI (tag-matching send/receives) and one-sided MPI (RMA put/get/atomics). OFI supports multiple "providers" where each provider corresponds to a different underlying fabric. CH4 has the OFI netmod to implement MPI on top of libfabric, the implementation of OFI.

In this talk, we focus on an MPICH configuration with CH4 and the OFI netmod underneath. We will refer to this configuration as MPICH-OFI.

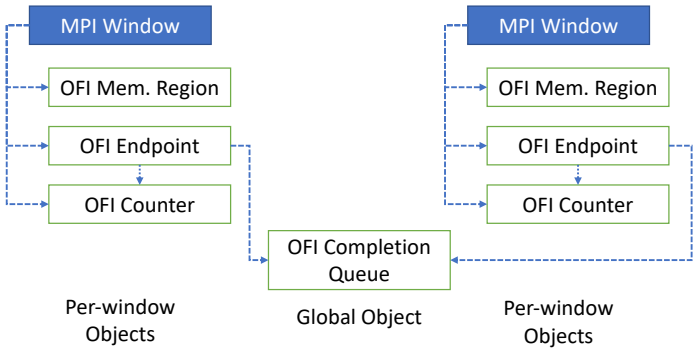

Figure 2: OFI Objects to Implement RMA in the OFI Netmod.

The overall design goal of MPICH-OFI is to map MPI functions onto OFI functions as closely as possible. RMA path also follow the same principle. Next, we describe the details about the design and implementation of the RMA path of the OFI netmod.

Figure 2 shows the OFI objects implementing the main RMA features. Each MPI window has three OFI objects, memory region, endpoint, and counter. A memory region represents an address range that is exposed to remote processes. An endpoint is an abstraction of a hardware command queue, where communication operations including RDMA read/write are posted. A counter tracks number of completed operations, and is bound to an endpoint so that once a posted operation completes, OFI increments it by one. An endpoint is also bound to a completion queue to generate completion events.

If the OFI provider supports scalable mode, MPICH-OFI uses FI_MR_SCALABLE mode for the memory regions. With this mode, MPICH-OFI can compute a remote memory key locally, avoiding the query to the remote process and the need to remember it for future use. With this mode, since MPICH-OFI does not need to store remote keys to communicate with its peers, it is possible to eliminate O(p) data structures associated with each window.

All RMA operations in OFI are non-blocking, meaning that completion will occur and be notified later. For RMA, MPICH-OFI uses two forms of completion, 1) bulk completion through a counter and 2) individual completion event through a completion queue (CQ). The first form is used for RMA operations that don't return request objects (e.g. MPI_PUT). As each endpoint is bound to a counter, OFI will increment the counter once an issued operation completes (both locally and remotely). When a synchronization function like MPI_WIN_FENCE is called, MPICH-OFI waits for the completion counter to reach the number of operations issued. MPICH-OFI is uses per-window counters, to allow synchronization on a window to proceed independently on the synchronization on a different window. The second form of completion is used for RMA operations that return request objects (e.g. MPI_RPUT). OFI is configured to allow individual operations to raise a completion event if needed. When issuing these operations, MPICH-OFI requests that a completion event is generated for that particular operation. Thus, when the operation completes, a completion event entry is generated in the CQ and MPICH-OFI can determine which operation is the one that has completed.

Derived datatypes. OFI functions for RMA support scatter/gather I/O through iovec structures. MPICH uses these to support derived datatypes. For non-contiguous data-types, such as strided vector types, MPICH generates an iovec to represent the data layout and pass it to OFI functions.

Atomics. MPI-3 has four atomic operations (MPI_ACCUMULATE, MPI_GET_ACCUMULATE, MPI_FETCH_AND_OP, and MPI_COMPARE_AND_SWAP). OFI also has atomic operations (fi_atomic, fi_fetch_atomic, and fi_compare_atomic.) These OFI atomic operations take as parameters a datatype (e.g. FI_INT32) and the operation to perform (e.g. FI_SUM). MPICH translates the MPI datatype/operation to the corresponding OFI datatype/operation. Different OFI providers may have different set of datatype/operation support, because sometimes certain combinations of datatype/operation may be difficult to implement efficiently over the underlying fabric. To deal with this difference, OFI provides a set of query functions to inquire about the atomic support. If the desired atomic operation is not supported by OFI or provider, MPICH falls back to a generic software-based active message path, similar to the one in MPICH with CH3 [6].

References

1. MPI: A Message Passing Interface Standard: http://www.mpi-forum.org/docs/docs.html
2. Torsten Hoefler, James Dinan, Rajeev Thakur, Brian Barrett, Pavan Balaji, William Gropp, and Keith Underwood. 2015. Remote Memory Access Programming in MPI-3. ACM Trans. Parallel Comput. 2, 2, Article 9 (June 2015), 26 pages. DOI=http://dx.doi.org/10.1145/2780584
3. MPICH | High-Performance Portable MPI: https://www.mpich.org
4. Ken Raffenetti et al. Why is MPI so slow?: analyzing the fundamental limits in implementing MPI-3.1. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17).
5. Libfabric: https://ofiwg.github.io/libfabric/
6. X. Zhao, P. Balaji and W. Gropp, "Scalability Challenges in Current MPI One-Sided Implementations," *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, Fuzhou, 2016, pp. 38-47.

Optimization notice

Acknowledgements MPICH-OFI is based on MPICH, an open source project, and thus has been supported by contributions from the community. We especially thank Pavan Balaji, Wesley Bland, Marc Gamell, Dmitry Gladkov, Yanfei Guo, Nusrat Islam, Sergey Oblomov, Ken Raffenetti, Alexander Sannikov, and Min Si, for their contributions to the RMA implementation.