VSS Challenge Problem: Verifying the Correctness of AllReduce Algorithms in the MPICH Implementation of MPI

Paul D. Hovland
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, IL, USA
hovland@anl.gov

We describe a challenge problem for verification based on the MPICH implementation of MPI. The MPICH implementation includes several algorithms for allreduce, all of which should be functionally equivalent to reduce followed by broadcast. We created standalone versions of three algorithms and verified two of them using CIVL.

1 Introduction

The message passing interface (MPI) standard includes several forms of collective communication, including broadcast, allgather, reduce, and allreduce [2]. Implementations of MPI such as MPICH and OpenMPI often include multiple algorithms for each collective communication operation, with the particular algorithm selected at runtime based on hardware characteristics such as network topology and latencies and application characteristics such as process count and array lengths [3]. All of the algorithms for a given operation should be functionally equivalent; the only differences should be in performance and, perhaps, restrictions on the domain of applicability (e.g., only power-of-two process counts or only for builtin (not user-defined) datatypes). Consequently, the correctness of the implementations of the various algorithms can be confirmed by verifying their equivalence to one another and to the behavior specified in the MPI standard.

As a concrete example, we consider the implementation of `MPI_Allreduce` in the MPICH library. This function performs a global reduction operation such as product or sum and distributes the result to all processes. It can be viewed as equivalent to `MPI_Reduce` followed by `MPI_Broadcast`. MPICH version 4.3.0 implements 7 different algorithms for (blocking) allreduce (not including special cases for intercommunicator reductions and shared memory parallelism): recursive doubling, recursive exchange, recursive multiplying, reducescatter-allgather, k-way reducescatter-allgather, ring exchange, and tree exchange. In addition, MPICH supports calling one of the nonblocking algorithms and waiting on the result. Some of these algorithms are described in [3].

The MPICH implementations of allreduce algorithms typically rely on internal communication routines such as `MPIC_Send` and `MPIR_Reduce_local` as well as MPICH-specific data structures. To facilitate verification of individual algorithms, we created a header file that replaces MPICH's `mpiimpl.h` and redefines MPICH subroutines and macros using standard MPI functions. This enables one to, for example, compile the file `allreduce_intra_recursive_doubling.c` with no modifictions and without any additional dependencies and use `MPIR_Allreduce_intra_recursive_doubling` as a replacement for `MPI_Allreduce`. A limitation of this approach is that we assume that the reduction operator is a builtin reduction operator (maximum, minimum, sum, product, etc.) and the datatype is a builtin datatype

Submitted to:
VSS 2025

(integer, float, double, etc.). It may be possible to remove the first restriction by making fewer assumptions about commutativity but the latter assumption will be difficult to overcome, in part because one would need to expose or replicate much of the complex mechanisms inside MPICH to deal with derived datatypes.

Currently, our challenge problem consists of the following files:

`allreduce_intra_recursive_doubling.c`: verbatim copy of the MPICH v4.3.0 file, also available at `https://github.com/pmodels/mpich/blob/main/src/mpi/coll/allreduce/allreduce_intra_recursive_doubling.c`

`allreduce_intra_recursive_multiplying.c`: slightly modified version of the file from MPICH v.4.3.0 (modifications include replacing direct access to fields within MPICH internal data structures with accessor macros and allocating memory for nonblocking communication requests)

`allreduce_intra_reduce_scatter_allgather.c`: slightly modified version of the file from MPICH v.4.3.0 (replace direct access to communicator internals and alloc/free temporary data structures)

`mpiimpl.h`: wrapper functions and macros to replace MPICH-specific functions and data structures

`mpir_threadcomm.h`: empty file to satisfy one of the include dependencies

`rd_allreduce_driver.c`: A driver that calls `MPIR_Allreduce_intra_recursive_doubling` and `MPI_Allreduce` and tests that the results are identical

`rm_allreduce_driver.c`: A driver that calls `MPIR_Allreduce_intra_recursive_multiplying` and `MPI_Allreduce` and tests that the results are identical

`rsag_allreduce_driver.c`: A driver that calls `MPIR_Allreduce_intra_reduce_scatter_allgather` and `MPI_Allreduce` and tests that the results are identical

We have verified the correctness of the MPICH implementation of recursive doubling by using CIVL [1] to prove the equivalence of MPIR_Allreduce_intra_recursive_doubling and `MPI_Allreduce` for arbitrary inputs (up to a maximum array length of 10 for `MPI_SUM` and `MPI_PROD` and 10 for) and arbitrary process counts (up to 16. In doing so, we leveraged the fact that CIVL has a builtin model of the semantics of `MPI_Allreduce`. We needed to provide CIVL with definitions of `MPI_Type_size` and `MPI_Reduce_local` since those functions are not included in the builtin model of MPI semantics. In addition, we needed to provide a definition of the `MPIR_Localcopy` that did not use `MPI_Sendrecv` (alternatively, we could have removed the `const` modifier from the send buffer parameter). Finally, we needed to modify the memory allocation for a temporary buffer to use the `datatype` parameter to cast the pointer to a pointer of appropriate type (CIVL requires that all memory allocations be cast to a non-void type). We were also able to verify the MPICH implementation of reduce_scatter-allgather following similar modifications. We do not expect to be able to use CIVL to verify the correctness of the MPICH implementation of recursive multiplying, as the latter relies on nonblocking sends and receives and CIVL cannot currently model these MPI operations.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under the RAPIDS Institute within the SciDAC program, under contract DE-AC02-06CH11357. We thank Hui Zhou, Ken Raffenetti, and Mike Wilkins for their insights into the MPICH implementation of allreduce.

References

[1] Stephen F. Siegel, Manchun Zheng, Ziqing Luo, Timothy K. Zirkel, Andre V. Marianiello, John G. Edenhofner, Matthew B. Dwyer & Michael S. Rogers (2015): *CIVL: the concurrency intermediate verification language*. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, Association for Computing Machinery, New York, NY, USA, doi:10.1145/2807591.2807635.

[2] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker & Jack Dongarra (1998): *MPI-The Complete Reference, Volume 1: The MPI Core*, 2nd. (revised) edition. MIT Press, Cambridge, MA, USA.

[3] Rajeev Thakur, Rolf Rabenseifner & William Gropp (2005): *Optimization of collective communication operations in MPICH*. *The International Journal of High Performance Computing Applications* 19(1), pp. 49–66, doi:10.1177/1094342005051521.