Handling Datatypes in MPI-3 One Sided

Robert Gerstenberger (Student)∗
Chemnitz University of Technology
gerro@hrz.tu-chemnitz.de

Torsten Hoefler (Advisor)
ETH Zurich
Dept. of Computer Science
Universitätstr. 6
8092 Zurich, Switzerland
htor@inf.ethz.ch

ABSTRACT
With the rise of modern interconnects offering direct remote memory access (RDMA), one sided programming becomes increasingly popular. The MPI-3.0 one sided chapter specifies a programming interface that leverages these features directly. fOMPI (fast one-sided MPI, [3]) was introduced as a reference implementation to provide highest performance and minimal overheads.

Many applications have to communicate non-consecutive data. In message passing each partner specifies their own MPI datatype (a memory layout), while for one sided communication one process defines both types (for the origin and the target process), which requires different kinds of optimizations for truly one sided implementations. In this work three different strategies are proposed to handle the combination of those MPI datatypes and implement them in fOMPI. Those strategies were evaluated with DDTBench, a set of data access pattern micro-applications, which were extended to handle one sided communication. Some cases improved up to a magnitude.

Categories and Subject Descriptors
C.5.1 [Computer Systems Organization]: Computer System Implementation—Super computers

General Terms
Performance

1. MOTIVATION
A lot of work was done on improving the performance of MPI datatypes for message passing, where a single datatype is used to pack/unpack data into/from a contiguous intermediate form. Most optimizations concentrate on either improving the copy algorithms ([8], [1]) or the datatype representation ([11], [6]).

∗Robert is in his last year of his first higher computer science degree, which is called "Diplom" in German.

For one sided communication only one process, called origin, specifies the MPI datatypes for the local and the remote memory (on the target). Optimizations in this area have received little attention. Träff et al. proposed for shared memory machines two approaches: (a) packing/unpacking via an intermediate buffer for globally shared memory or (b) a two sided algorithm for process local memory [4]. Both approaches aren’t feasible for truly one sided implementations with distributed memory and communication operations limited to put, get and a few atomic operations.

Our key contributions are:

- Description of three strategies with different communication and memory overhead to handle datatypes and an implementation in fOMPI
- Performance evaluation using data access pattern micro-applications with improvements up to a magnitude for some cases

2. DATATYPE HANDLING STRATEGIES
fOMPI was written to enable fast communication of contiguous datatypes with low overheads. Since it is non-trivial to decide whether a datatype is contiguous with only metadata (type size, extent), fOMPI builds a datatype representation that is accessible from outside MPI. The MPI-Types library [7] is used to flatten the datatype to a list of {size, offset}-tuples. If both datatypes are contiguous, i.e. the list lengths are one, the data is communicated directly. fOMPI implements a fast path for MPI_INT, MPI_CHAR, MPI_DOUBLE, MPI_UINT64_T.

If at least one of the types is non-contiguous, a so-called maximal block list is created, which consists of {size, origin offset, target offset}-tuples. In each step the next block size is the minimum of the remaining sizes of the current blocks of both (origin, target) lists. If a block isn’t fully processed, the remaining size is decreased and the offset increased accordingly. Otherwise the next block of the list is used. Since in each step at least one of the blocks on either list is eliminated, the length of the new list is lesser than the sum of both list lengths.

Afterwards fOMPI applies, using the new list, one of the three following strategies, as proposed by Schneider et al. [10], to communicate the data. The strategy is selected at compile time. fOMPI caches the flattened datatypes.
2.1 Maximal Block Strategy
This strategy puts each of the maximal blocks separately and was the original bufferless/zero copy approach, which was presented in the technical paper. Probably it provides the biggest overlap, since the least time is spent in the communication call. It allows on the fly creation of the maximal block list. This strategy is always used for intra-node communication to avoid data serialization.

2.2 Fixed Buffer Size Strategy
This strategy uses multiple fixed size buffers. The algorithm first sorts the maximal block list and afterwards iterates over every block. If a block is consecutive on the target with the next, both are copied together into a temporary buffer. The temporary buffer is communicated before an overflow occurs or if the current block is non-consecutive with the buffer content. A block is directly communicated, in case it is non-consecutive with both neighbors or too big to fit in the buffer. Multiple buffers are used to overlap communication with copy operations.

The size and number of buffers are compile-time parameter. While this strategy reduces the number of communication operations, it may reduce overlap due the additional copy and sorting overhead. Otherwise overlap may increase, since DMAPP outsources the handling of larger transfers to a network controller.

2.3 Target Block Strategy
This strategy is a special case of the fixed buffer size strategy, where the overall buffer size is at least big enough to hold the complete MPI datatype. Since there is no upper bound for the memory a datatype covers, it has no practical use, but allows measurements with the lower bound of communication operations. For the evaluation foMPI was compiled with a buffer size big enough to hold any of the applications datatypes.

3. EVALUATION
DDTBench [9] is a collection of micro-applications, which extract data access patterns from various scientific applications. For this poster DDTBench was extended for one sided communication to evaluate the datatype handling of foMPI.

![Figure 1: Measurement Loop for DDTBench](image)

4. DISCUSSIONS AND CONCLUSIONS
While foMPI currently only uses DMAPP contiguous communication operations, DMAPP also offers non-contiguous operations (strided put/get, scatter/gather), which may allow further improvements for vector and indexed types. However this would call for a complete rewrite of foMPIs datatype engine. Another interesting optimization would be the integration of input-output datatypes, as proposed by Mir et al. [5], which allow to directly copy/communicate between two non-contiguous datatypes without serialization to an intermediate buffer.

5. REFERENCES