An efficient implementation of blocking and persistent MPI collective communication

Andreas Jocksch
andreas.jocksch@cscs.ch
CSCS, Swiss National Supercomputing Centre,
Via Trevano 131
CH-6900 Lugano, Switzerland

Jean-Guillaume Piccinali
CSCS, Swiss National Supercomputing Centre,
Via Trevano 131
CH-6900 Lugano, Switzerland

1 INTRODUCTION

Persistent collective communication [3] became a feature of the MPI standard in version 4.0 and first implementations are available in various libraries such as MPICH, OpenMPI, and MPC [1].

We improve our existing implementation of persistent collective communication of MPI allreduce, reduce_scatter_block, and allgather [4] and extend it to blocking communication. In this implementation of persistent collective communication a complex algorithmic setup is performed in the initialisation phase of the communication. If the optimal algorithm is repeatedly executed the expensive initialisation is amortised, making the approach well suited to situations where communication algorithms with exactly the same parameters are repeated regularly.

An example of a code with changing message sizes between calls of otherwise identical communication is CP2K [5]. Typically the blocking communication interface is called for these applications. Our contribution covers this case: we initialise algorithms for different message sizes at the time of the MPI communicator creation. The creation has to be called rarely only, in order to allow for compensation of the initialisation's execution time. Then for the actual communication call the required algorithm is selected from the prepared ones and adapted as necessary (with padding, for example).

A second application of our blocking communication implementation is in legacy codes that allow algorithmically the call of persistent collective communication, but with changes in the code.

2 THE ALGORITHMS

The base algorithms upon which the present work is built are described in [4]. Allreduce is implemented in three phases: reduce_scatter followed by allreduce and allgather. As an improvement from our previous work in the parameter estimation memory copies of the algorithm are considered. This has the consequence that for one MPI task per socket (node) recursive exchange has preference over cyclic shift. Furthermore the use of memory copies has been minimised. The resulting algorithms are similar to the ones of Ruefenacht [6] and Rabenseifner [7] for short and long messages, respectively. For long messages the algorithms are most similar for 2^n tasks.

EuroMPI '23, September 11–13, 2023, Bristol, UK

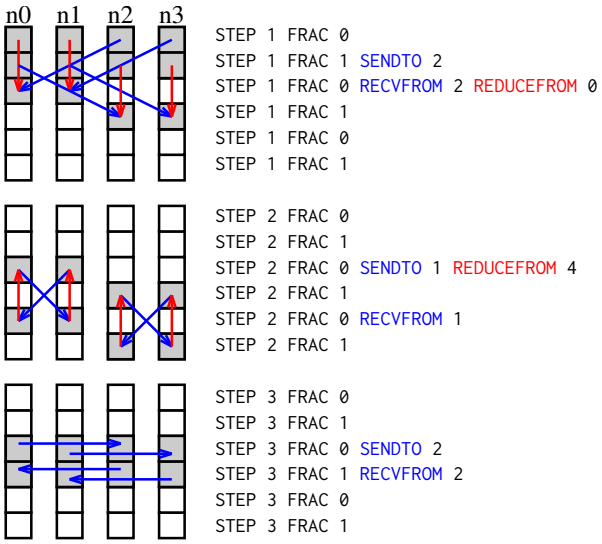


Figure 1: Algorithmic scheme allreduce 4 nodes with 1 MPI task per node, left: nodes $n0$ to $n3$, right: script for node 0

Figure 1 shows the algorithmic scheme for a four nodes and one task per node allreduce with a radix of two, a partial reduce_scatter, allreduce, and allgather. On the left of the figure is a graphical view for nodes $n0$ to $n3$. Grey cells represent data being processed, blue arrows show data transferred between nodes, and red arrows reduction operations within the nodes. There are three steps from top to bottom and data is transferred before reductions are performed. The message size of `sendbuf` and `recvbuf` is two boxes. The top two boxes of each six boxes column are the send buffer, the middle two boxes the receive buffer and the bottom two boxes a temporary buffer. On the right of the figure the scheme is shown as a script for node zero, which is very similar to the script generated internally by the library which is processed further to bytecode [4], which is executed in the actual collective call. Each line corresponds to its box in the $n0$ column (left). As already mentioned, there are three steps, indicated by the keyword `STEP`. The keyword `FRAC` describes the memory chunk which is processed on, the first two lines 0-1 are the send buffer, the second two lines 0-1 the receive buffer and the third two lines 0-1 a temporary buffer. The keywords `RECVFROM` and `SENDTO` describe a non-blocking receive and send operation, respectively, and the number following is the partner's MPI rank. For every step, sends and receives are scheduled followed by a waitall. The step is finalised with a possible reduction operation, keyword

REDUCEFROM, where the number is the absolute line number starting from zero.

During the set up of the algorithms in the initalisation phase, bytecode is generated which encodes (mostly) computation, MPI_Irecv, MPI_Isend, and MPI_Waitall. For every communicator creation the initalisation is repeatedly executed for certain message sizes. When the blocking collective communication is called, the most suitable message size from the stored algorithms is chosen and all instructions in the bytecode are adapted to this message size. Since the number of algorithms stored is limited the ideal adaptation might not be possible. Therefore, for the allreduce operation, padding might be required, which has the disadvantage of unnecessary data being processed. There is also the overhead due to the algorithm selection and the adaptation of the bytecode for the particular message size.

Allreduce, reduce_scatter_block, and allgather are provided as blocking versions. This list may be extended; however, the versions of the calls with variable vector length e.g., reduce_scatter and allgatherv cannot be accommodated easily. The reason for this is that they do not have one parameter as message length but many. We see as only one option for an implementation a padding up to equal message length which would introduce a significant overhead due to additional memory copies.

3 IMPLEMENTATION

All collective communication routines are implemented on top of MPI point-to-point communication. In order to allow for a simple plug in to applications we use the MPI profiler hook. For the parameter ranges where our implementation does not outperform the underlying MPI library, the original collective communication might be called. Our blocking communication is activated only when a particular environment variable is set, and codes which call communicator creations and destructions frequently are not penalised. An alternative solution (future work) would be to pass a specific value to the MPI_Info argument when the communicator is created, similar to what is suggested in [2].

Our implementation is publicly available at
https://github.com/eth-cscs/ext_mpi_collectives.

4 BENCHMARKS

On an HPE Cray EX system with 64-core AMD EPYC 7742 processors, using one MPI task per node, for the numbers of tasks and message sizes investigated our blocking implementation almost always outperforms HPE (Cray) MPI (Fig. 2, 3). We do not observe any significant performance penalty if the message sizes were chosen such that padding is applied, which in any case, is a few bytes only. Our persistent implementation is even faster. Table 1 shows the number of communication ports (the number of MPI_Irecv - MPI_Isend pairs acting simultaneously on the node) determined for different message sizes together with the timings. The 'num ports' indicate the number of communication ports used in the algorithm, e.g., (7 1) are seven ports in the first step and one port in the second step, both are allgather steps. Negative numbers for the ports describe reduce_scatter steps, e.g., (-7 1 7) is a reduce_scatter with seven ports an allreduce with one port and an allgather with

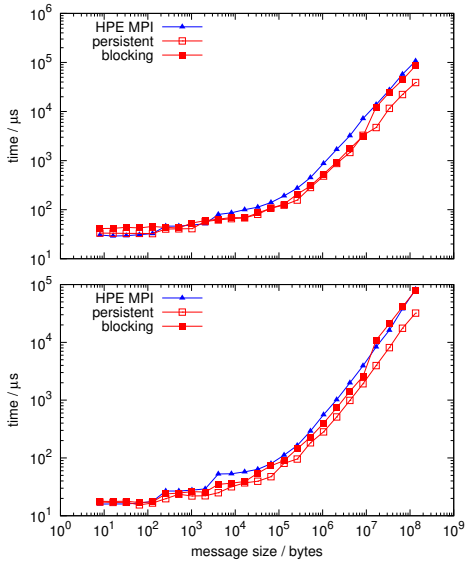


Figure 2: Allreduce on 128 nodes (top) and 16 nodes (bottom) with 1 MPI task per node, HPE Cray EX

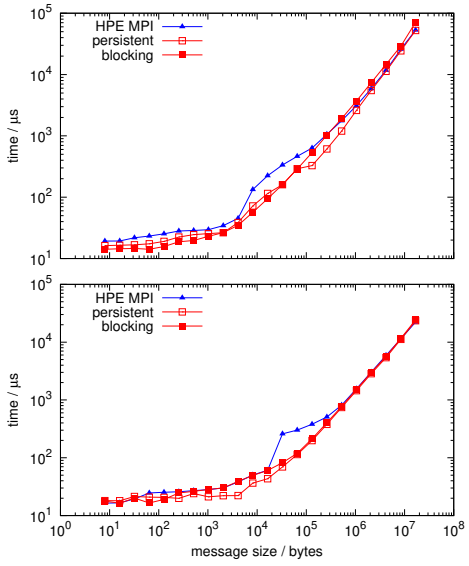


Figure 3: Reduce_scatter_block (top) and allgather (bottom) on 16 nodes with 1 MPI task per node, HPE Cray EX

seven ports. Both examples allow for the execution of the recursive exchange algorithm if applied to 16 nodes.

Figure 4 shows the performance comparison for 64 MPI tasks on a single node, on a Cray XC40 system (Aries network) with Two Intel Xeon E5-2695 v4 running at 2.10GHz (2 x 18 cores). Our persistent allreduce communication outperforms Cray MPI for message sizes larger than 10^4 bytes. For message sizes larger than 10^5 bytes also our blocking allreduce communication outperforms Cray

Table 1: Allreduce on 16 nodes with 1 task per node, HPE Cray EX

message size / bytes	num ports	time HPE MPI / μs	time / μs
8	8 1	$1.60 \cdot 10^1$	$1.73 \cdot 10^1$
.	.	.	.
.	.	.	.
.	.	.	.
8192	7 1	$5.33 \cdot 10^1$	$3.18 \cdot 10^1$
16384	-7 1 7	$5.75 \cdot 10^1$	$3.71 \cdot 10^1$
32768	-7 1 7	$6.37 \cdot 10^1$	$3.96 \cdot 10^1$
65536	-7 1 7	$7.96 \cdot 10^1$	$4.73 \cdot 10^1$
131072	-1 -7 7 1	$1.12 \cdot 10^2$	$8.10 \cdot 10^1$
262144	-3 -3 3 3	$1.64 \cdot 10^2$	$9.56 \cdot 10^1$
524288	-1 -7 7 1	$2.91 \cdot 10^2$	$1.83 \cdot 10^2$
1048576	-1 -1 -3 3 1 1	$5.60 \cdot 10^2$	$2.82 \cdot 10^2$
2097152	-1 -1 -3 3 1 1	$1.02 \cdot 10^3$	$5.12 \cdot 10^2$
4194304	-1 -1 -1 -1 1 1 1 1	$1.99 \cdot 10^3$	$9.91 \cdot 10^2$
.	.	.	.
.	.	.	.
.	.	.	.
134217728	-1 -1 -1 -1 1 1 1 1	$8.26 \cdot 10^4$	$3.21 \cdot 10^4$

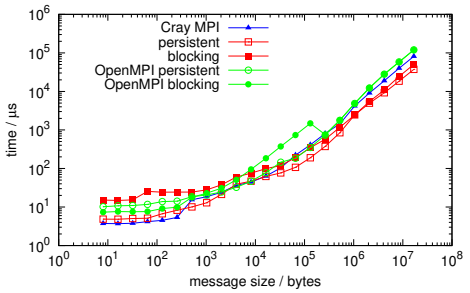


Figure 4: Allreduce on 1 node with 64 MPI tasks per node, Cray XC40

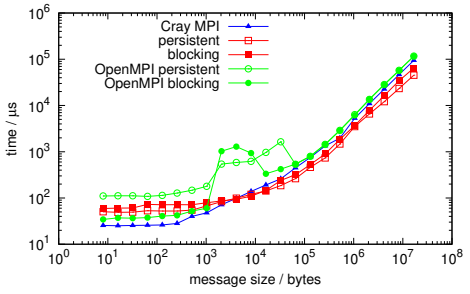


Figure 5: Allreduce on 16 nodes with 64 MPI tasks per node, Cray XC40

MPI. For 16 nodes with 64 MPI tasks per node (Fig. 5) the overhead of our approach leads to higher execution times for short messages compared to Cray MPI, while for medium size messages and long messages Cray MPI is outperformed. OpenMPI 4.1.5 is always slower than Cray MPI. However, with the current software on Slingshot (HPE Cray EX with libfabric 1.15.2.0), persistent (and blocking) allreduce on multiple tasks per node on multiple nodes does not entirely outperform the vendor's blocking implementation. Our library and the underlying HPE MPI and libfabric libraries are still under development for better performance of the multiple tasks on multiple nodes case.

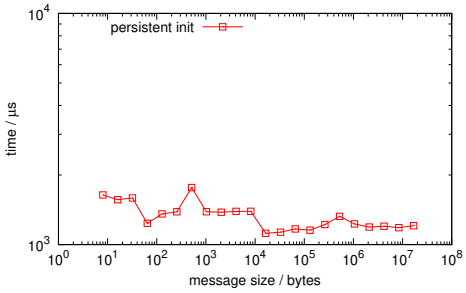


Figure 6: Allreduce initialisation on 16 nodes with 1 MPI task per node, HPE Cray EX

Figure 6 shows the cost for the allreduce initialisation on the HPE Cray EX system. For short messages it is two orders of magnitude higher than the actual execution of allreduce. For long messages initialisation and execution consume approximately the same time. If our blocking collectives implementation is activated then ten calls to the initialisation routines per collective and communicator creation are performed.

5 CONCLUSIONS

We have introduced an implementation of blocking collective communication based on algorithms for persistent collective communication. This implementation outperforms HPE/Cray MPI, when communicator creation and destruction are not called frequently. However, only a subset of blocking communication patterns can be supported in this way. Whenever possible the even faster persistent collective communication routines introduced in [4] and slightly improved in this contribution should be utilised. Non-blocking collective communication could be implemented analogously to our blocking routines, with the caveat that every open request handle would require a separate temporary memory buffer.

REFERENCES

[1] Stephane Bouhrour, Thibaut Pepin, and Julien Jaeger. 2022. Towards leveraging collective performance with the support of MPI 4.0 features in MPC. *Parallel Comput.* 109 (2022), 102860.

[2] Torsten Hoefler and Timo Schneider. 2012. Optimization principles for collective neighborhood communications. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10.

[3] Daniel J Holmes, Anthony Skjellum, Julien Jaeger, Ryan E Grant, Purushotham V Bangalore, Matthew GF Dosanjh, Amanda Bienz, and Derek Schafer. 2021. Partitioned collective communication. In *2021 Workshop on Exascale MPI (ExaMPI)*. IEEE, 9–17.

[4] Andreas Jocksch, Noé Ohana, Emmanuel Lanti, Eirini Koutsaniti, Vasileios Karakasis, and Laurent Villard. 2021. An optimisation of allreduce communication in message-passing systems. *Parallel Comput.* 107 (2021), 102812.

[5] Thomas D Kühne, Marcella Iannuzzi, Mauro Del Ben, Vladimir V Rybkin, Patrick Seewald, Frederick Stein, Teodoro Laino, Rustam Z Khaliullin, Ole Schütt, Florian Schiffmann, et al. 2020. CP2K: An electronic structure and molecular dynamics software package-Quickstep: Efficient and accurate electronic structure calculations. *The Journal of Chemical Physics* 152, 19 (2020), 194103.

[6] Martin Ruefenacht, Mark Bull, and Stephen Booth. 2017. Generalisation of recursive doubling for AllReduce: Now with simulation. *Parallel Comput.* 69 (2017), 24–44.

[7] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

An efficient implementation of blocking and persistent MPI collective communication

Andreas Jocksch[1] **and Jean-Guillaume Piccinali**[1]

[1]**CSCS, Swiss National Supercomputing Centre, Via Trevano 131**
6900 Lugano, Switzerland

Motivation

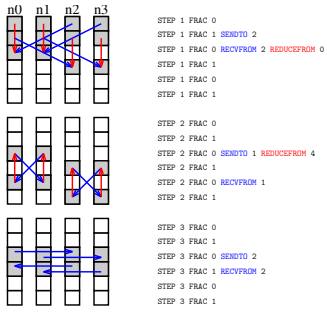
- MPI persistent collective communication [1, 2] extends the MPI interface but many codes call collective communications with varying message sizes between calls → also blocking interface required
- When communicator-create functions are rarely called they can be used for the setup of the algorithms
- We focus on the operations allreduce, reduce_scatter_block and allgather

Contributions

- Improved parameter selection for persistent collective communication compared to our previous contribution [2], memcopies are avoided
- Alternative implementation for blocking collective communication utilising communicator create functions for the setup of the algorithms

Algorithms

- Allreduce in three phases: reduce_scatter, allreduce, and allgather
- Algorithmic choice in initialisation routine (based on benchmark at installation time of the library)
- Algorithms close to Ruefenacht [3] and Rabenseifner [4] for short and long messages, respectively
- Script of the algorithm translated to bytecode




```
STEP 1 FRAC 0
STEP 1 FRAC 1 SENDTO 2
STEP 1 FRAC 0 RECVFROM 2 REDUCEFROM 0
STEP 1 FRAC 1
STEP 1 FRAC 0
STEP 1 FRAC 1

STEP 2 FRAC 0
STEP 2 FRAC 1
STEP 2 FRAC 0 SENDTO 1 REDUCEFROM 4
STEP 2 FRAC 1
STEP 2 FRAC 0 RECVFROM 1
STEP 2 FRAC 1

STEP 3 FRAC 0
STEP 3 FRAC 1
STEP 3 FRAC 0 SENDTO 2
STEP 3 FRAC 1 RECVFROM 2
STEP 3 FRAC 0
STEP 3 FRAC 1
```

Algorithmic scheme allreduce 4 nodes with 1 MPI task per node, left: nodes n0 to n3, right: script for node 0

- Blocking collectives: initialisation of algorithms for several message sizes when communicator creation
- Adaptation of the bytecode of closest message size when blocking collectives are called

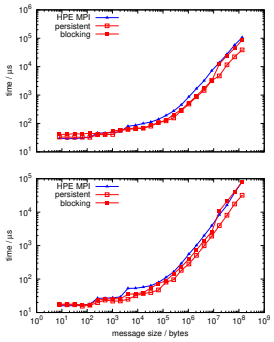
Implementation

- On top of MPI point-to-point communication
- Plug in to applications with the MPI profiler hook
- Blocking communication activated with environment variable
- Prototype library which implements part of the persistent collective communication of the MPI 4.0 standard and blocking collective communication

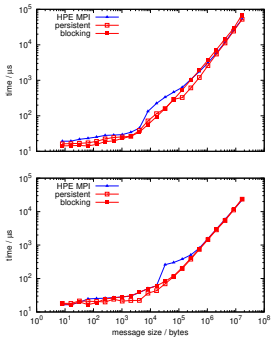
`https://github.com/eth-cscs/ext_mpi_collectives`

Benchmarks

- HPE Cray EX system with 64 core AMD EPYC 7742 processors



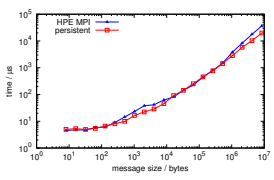
Allreduce on 128 nodes (top) and 16 nodes (bottom) with 1 MPI task per node, HPE Cray EX



Reduce_scatter_block (top) and allgather (bottom) on 16 nodes with 1 MPI task per node, HPE Cray EX

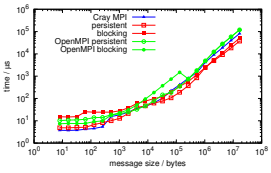
Table 1: Allreduce on 16 nodes with 1 task per node, HPE Cray EX

message size / bytes	num ports	time HPE MPI / µs	time / µs
8	8 1	$1.60 \cdot 10^{3}$	$1.73 \cdot 10^{3}$
⋮	⋮	⋮	⋮
8192	7 1	$5.33 \cdot 10^{3}$	$3.18 \cdot 10^{3}$
16384	-7 1 7	$5.75 \cdot 10^{3}$	$3.71 \cdot 10^{3}$
32768	-7 1 7	$6.37 \cdot 10^{3}$	$3.96 \cdot 10^{3}$
65536	-7 1 7	$7.96 \cdot 10^{3}$	$4.73 \cdot 10^{3}$
131072	-1 -7 7 1	$1.12 \cdot 10^{3}$	$8.10 \cdot 10^{3}$
262144	-3 -3 3 3	$1.64 \cdot 10^{3}$	$9.56 \cdot 10^{3}$
524288	-1 -7 7 1	$2.91 \cdot 10^{3}$	$1.83 \cdot 10^{3}$
1048576	-1 -1 -3 3 1 1	$5.60 \cdot 10^{3}$	$2.82 \cdot 10^{3}$
2097152	-1 -1 -3 3 1 1	$1.02 \cdot 10^{3}$	$5.12 \cdot 10^{3}$
4194304	-1 -1 -1 -1 1 1 1 1	$1.99 \cdot 10^{3}$	$9.91 \cdot 10^{3}$
⋮	⋮	⋮	⋮
134217728	-1 -1 -1 -1 1 1 1 1	$8.26 \cdot 10^{4}$	$3.21 \cdot 10^{4}$

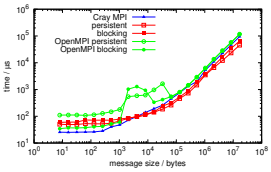


Allreduce on 1 node with 128 MPI tasks per node, HPE Cray EX

- Cray XC40 system with Two Intel Xeon E5-2695 v4 running at 2.10GHz (2 x 18 cores)

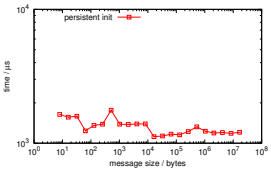


Allreduce on 1 node with 64 MPI tasks per node, Cray XC40



Allreduce on 16 nodes with 64 MPI tasks per node, Cray XC40

- Problem: Low performance of our library for multiple nodes with multiple MPI tasks per node on the HPE Cray EX system



Allreduce initialisation on 16 nodes with 1 MPI task per node, HPE Cray EX

Conclusions

- Our blocking collective routines mostly outperform the reference library HPE MPI
- Our persistent collective communication routines are faster than our blocking counterparts
- Whenever possible persistent collective communication should be used

Future work

Blocking collectives for multiple MPI tasks per node and GPU support

References

[1] Bouhrour, S., Pepin, T., Jaeger, J.: Towards leveraging collective performance with the support of MPI 4.0 features in MPC. Parallel Computing 109, 102860 (2022)

[2] Jocksch, A., Ohana, N., Lanti, E., Koutsaniti, E., Karakasis, V., Villard, L.: An optimisation of allreduce communication in message-passing systems. Parallel Computing 107, 102812 (2021)

[3] Ruefenacht, M., Bull, M., Booth, S.: Generalisation of recursive doubling for allreduce: Now with simulation. Parallel Computing 69, 24–44 (2017)

[4] Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of collective communication operations in MPICH. The International Journal of High Performance Computing Applications 19(1), 49–66 (2005)