Towards an optimal allreduce communication in message-passing systems

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Andreas Jocksch[1], Noé Ohana[2], Emmanuel Lanti[2], Vasileios Karakasis[1] and Laurent Villard[2],

[1]CSCS, Swiss National Supercomputing Centre, Via Trevano 131, 6900 Lugano, Switzerland
[2]Ecole Polytechnique Fédérale de Lausanne (EPFL), Swiss Plasmas Center, 1015 Lausanne, Switzerland

Motivation

- Goal: Fast allreduce operation as MPI persistent collective communication
- Collective communication provides complex communication patterns which can be highly optimised
- Allreduce
  - Frequently used on HPC systems [7]
  - Applications in data science but also in other fields
- Related work [3, 6, 5]

Contributions

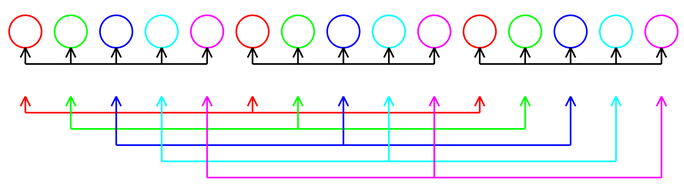
- Alternative implementation and further formalisation of known algorithms
- Parameter choice based on benchmark at installation time of the library and computation in initialisation routine (persistent like collective communication)
- Good speedups

Algorithms

- Proposed in a theoretical framework [1]
- Different algorithms for different message sizes
  - Long messages
    - Algorithm II in Ref. [1]
    - Established implementation: reduce_scatter_block followed by allgather, binary blocks (Rabenseifner)
    - Proposed approach: reduce_scatter followed by allgatherv
    - Established implementation: reduce_scatter for 2^n (n =number of processing elements) with special treatment of additional processing elements (1 processing element implementation dependent e.g. 1 MPI process or 1 node)
    - Proposed approach: reduce_scatter with cyclic shift, computational complexity as for allgatherv [2] plus computation
    - Basic algorithms allgatherv and reduce_scatter without performance penalty compared to their equal message size counterparts
  - Short messages
    - Algorithm I in Ref. [1]
    - Established implementation: Recursive doubling / halving with special treatment of non 2^n processing elements
    - Proposed approach: cyclic shift with prefix operation
- Factors of cyclic shift flexible between steps for all algorithms
- All algorithms exploit memory shared between MPI processes on the node, no oversubscription of cores by MPI processes and only a part of the MPI processes per node are sending and receiving

Cyclic shift with prefix operation

- Decomposition of the number of nodes into prime factors [6]



Independent cyclic shifts with prime factors 5 and 3; circles indicate nodes, black connections indicate the first communication step with groups of 5 nodes and coloured connections indicate the second communication step with groups of 3 nodes; reduction of data between the MPI processes within the nodes before and distribution of data to the MPI processes of the node after this operation is not displayed

- For every prime (also called group [5]) an independent allreduce is possible; example: 39 tasks, radix 2 → algorithm applied to 1×39 or 3×13, 6 steps for both cases, but 8 or 7 times the basic message size → without factorisation latency optimal but not optimal in general
- Also since our number of MPI processes per node might be relatively large, combining small primes is efficient

	$n0$	$n1$	$n2$	$n3$	$n4$	$n5$	$n6$	$n7$	$n8$	$n9$	nA
X	0	1	2	3	4	5	6	7	8	9	A
X	1	2	3	4	5	6	7	8	9	A	0
X	2	3	4	5	6	7	8	9	A	0	1
X	3	4	5	6	7	8	9	A	0	1	2
	4	5	6	7	8	9	A	0	1	2	3
	5	6	7	8	9	A	0	1	2	3	4
	6	7	8	9	A	0	1	2	3	4	5
X	7	8	9	A	0	1	2	3	4	5	6
	8	9	A	0	1	2	3	4	5	6	7
	9	A	0	1	2	3	4	5	6	7	8
X	A	0	1	2	3	4	5	6	7	8	9

$n0$	$n1$	...	nA
$\sum_{i=0}^{0} i$	$\sum_{i=1}^{1} i$	...	$\sum_{i=A}^{A} i$
$\sum_{i=0}^{1} i$	$\sum_{i=1}^{2} i$	...	$\sum_{i=A}^{A} i + \sum_{i=0}^{0} i$
$\sum_{i=0}^{2} i$	$\sum_{i=1}^{3} i$	...	$\sum_{i=A}^{A} i + \sum_{i=0}^{1} i$
$\sum_{i=0}^{3} i$	$\sum_{i=1}^{4} i$	...	$\sum_{i=A}^{A} i + \sum_{i=0}^{2} i$
–	–	...	–
–	–	...	–
$\sum_{i=0}^{7} i$	$\sum_{i=1}^{8} i$	...	$\sum_{i=A}^{A} i + \sum_{i=0}^{6} i$
–	–	...	–
–	–	...	–
$\sum_{i=0}^{A} i$	$\sum_{i=1}^{A} i + \sum_{i=0}^{0} i$	...	$\sum_{i=A}^{A} i + \sum_{i=0}^{9} i$

Cyclic shift algorithm adapted from `allgather`, nodes $n0$-nA with messages 0-A (hexadecimal notation), radix 2, horizontal lines _ indicate the end of every step, X are the lines required for `allreduce` (top), for sum reduction inclusive scans from top to bottom which are actually stored and communicated (bottom)

- Allows efficient execution even for large prime number of nodes

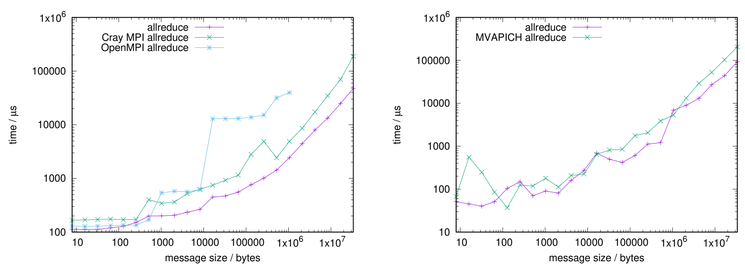
Implementation and tuning

- Our collective communication is build on top of MPI point-to-point communication [4]
- Bytecode generated in initialisation phase for repeated execution
- The number of send and receive MPI processes per node is message size dependent (implemented for long messages only)
  - Many senders and receivers per node for relatively short messages, a few for long ones on our networks
  - Benchmark at installation time of the library
- Prototype library similar to use as persistent collective communication of the MPI standard
- Blocking implementation with current limitations with respect to datatypes and distribution of communicators to the nodes

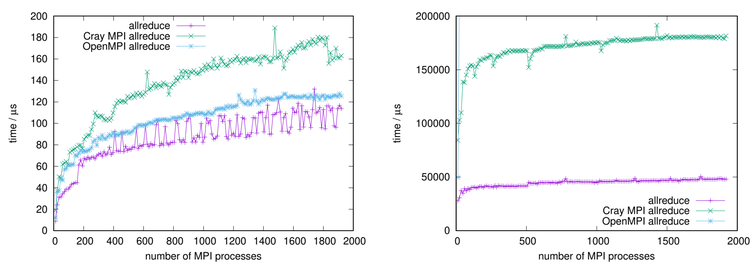
`https://github.com/eth-cscs/ext_mpi_collectives`

Benchmarks

- Cray XC 40 KNL 64-core Intel(R) Xeon Phi(TM) CPU 7230 processors at 1.30GHz, 5 virtual 12 core CPUs per KNL
- Infiniband cluster using 17 nodes with two 12 core Haswell(TM) E5-2650v.3 2.66GHz CPUs



Allreduce on 160 nodes with 9600 MPI processes Cray (left) and on 17 nodes with 408 MPI processes Infiniband (right)



Allreduce with 8 bytes (left) and 33554432 bytes (right), Cray

- Our routine mostly outperforms the references

Future work

- Medium size messages, algorithm III [1]; note that an implementation of an algorithm for medium size messages without cyclic shift by other authors exist [5]
- Integration in our plasma physics application ORB5

Acknowledgement

The authors wish to thank Meteoswiss for the utilisation of their Piz Kesch system.

References

[1] Bruck, J., Ho, C.T.: Efficient global combine operations in multi-port message-passing systems. Parallel Processing Letters 3(04), 335–346 (1993)

[2] Bruck, J., Ho, C.T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Transactions on parallel and distributed systems 8(11), 1143–1156 (1997)

[3] End, V., Simmendinger, C., Yahyapour, R., Alrutz, T.: Butterfly-like algorithms for GASPI split-phase allreduce. International Journal on Advances in Systems and Measurements 9 (2016)

[4] Jocksch, A., Ohana, N., Lanti, E., Karakasis, V., Villard, L.: Optimised allgatherv, reduce_scatter and allreduce communication in message-passing systems (2020)

[5] Kolmakov, D., Zhang, X.: A generalization of the allreduce operation (2020)

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

[7] 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)