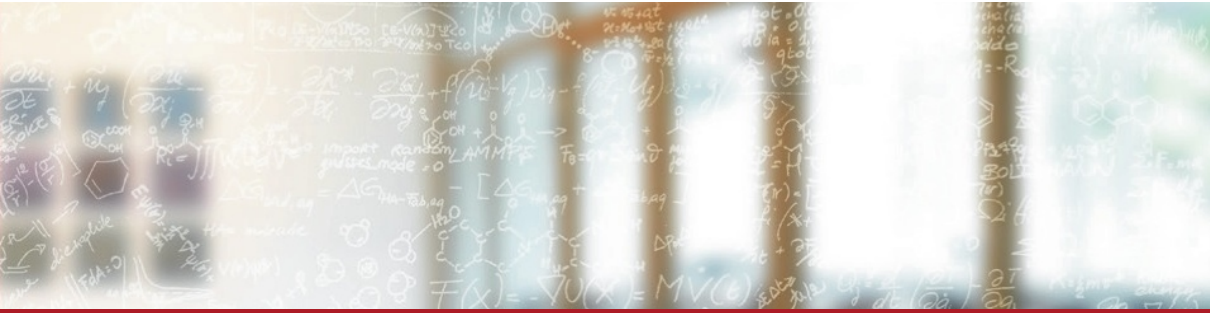
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



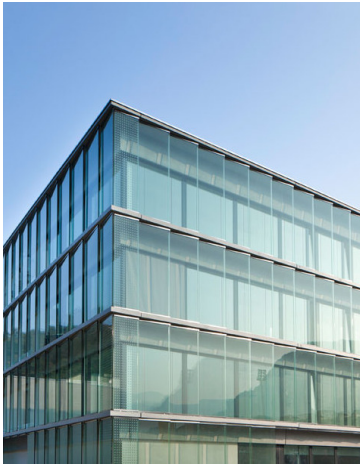
Efficient Collective Communication on Modern HPC GPU Architectures

HPC-AI Swiss Conference 2026

Andreas Jocksch (CSCS), C. Nicole Avans, Riley Shipley and Anthony Skjellum (Tennessee Technological University)

21 April, 2026

Efficient Collective Communication on Modern HPC GPU Architectures



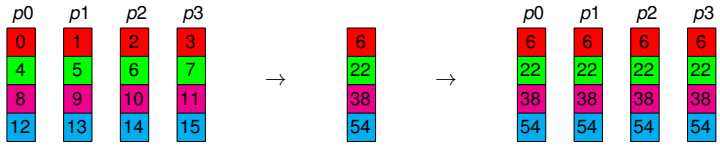
- Motivation
- Best known algorithm for allreduce
- Our solution for allreduce
- Benchmarks
- Conclusions

Motivation

Application

- Collective Communication part of many HPC applications
  - Artificial intelligence (AI) with different communication patterns
  - Parallel FFTs with alltoall communication
  - Distributed matrix multiplication
- Standard patterns
  - Alltoall
  - Allreduce
  - Reduce
  - Bcast
  - ...

Allreduce



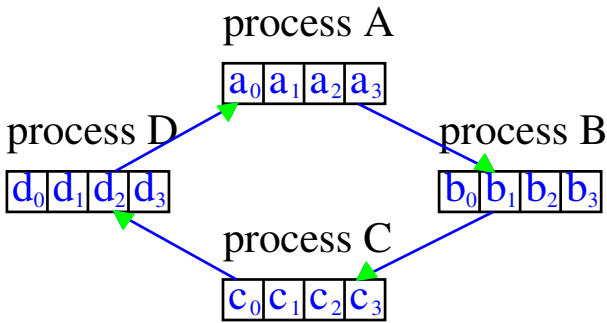
Allreduce (sum) from left (source) to right (results)

Allreduce contd.

- Frequently used
- High optimisation potential
- From its algorithms other collective operations Reduce, Bcast, ... can be derived
- MPI libraries and NCCL support Allreduce

Best known algorithm for allreduce

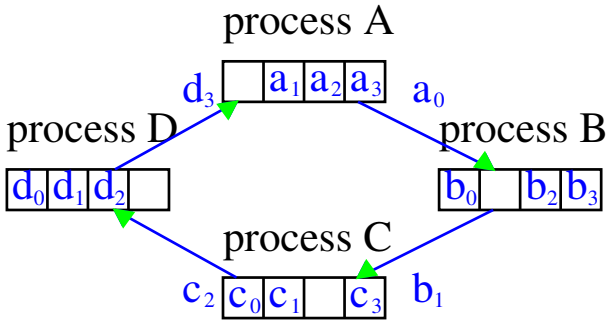
Ring algorithm



Ring algorithm for long messages

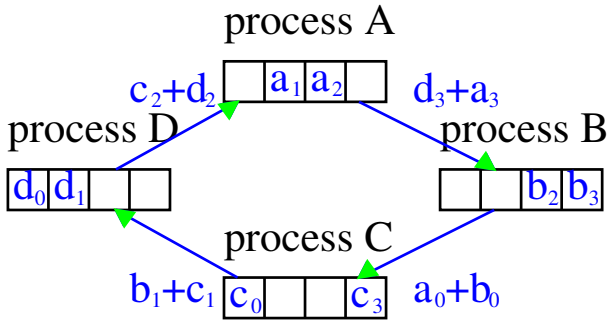
https://medium.com/data-science/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da

Ring algorithm contd.



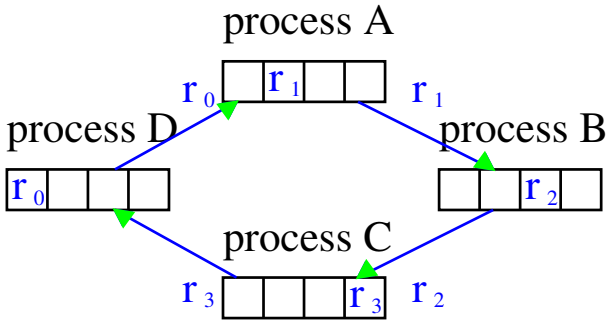
Ring algorithm for long messages

Ring algorithm contd.



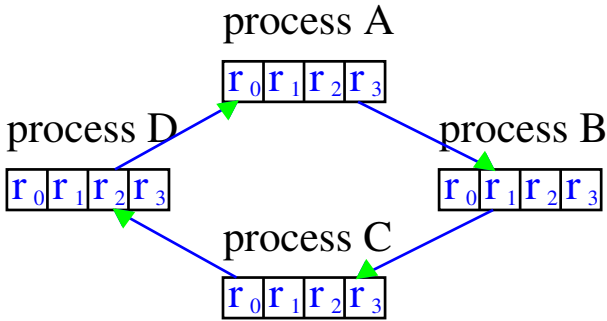
Ring algorithm for long messages

Ring algorithm contd.



Ring algorithm for long messages

Ring algorithm contd.



Ring algorithm for long messages

Shortcomings

- Only for bandwidth dominated communication ideal
- Messages might be shorter e.g. for strong scaling
- Multiple physical or logical ports per node
- Hierarchies of network and shared memory are present

Our solution for allreduce

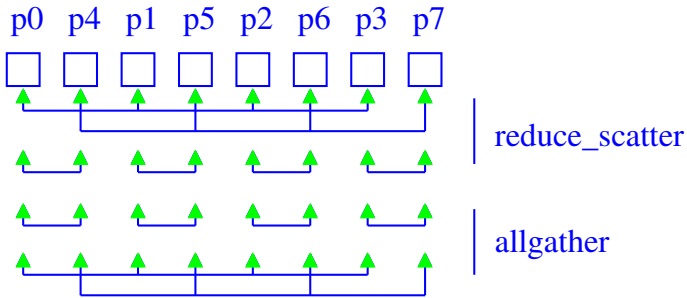
The algorithms

- $\rightarrow$ Recursive exchange or cyclic shift algorithm also for latency relevant cases optimal
- Multi-radix implementation
- High speed network between nodes assumed to be fully connected (good approximation for current HPE machines with Slingshot network), bandwidth-latency cost model
- Computer networks might support more than one port per node, message size dependent, on Slingshot network short messages many ports and long messages one port
- Latency is reduced by collecting all messages on the nodes before sending them and distributing them after receiving $\rightarrow$ reduction of number of messages
- For repeated `MPI_Allreduce` calls with same message size, persistent MPI well suited (initialisation phase for setup of the communication algorithms)
- Library build on top of MPI, (`MPI_Irecv`, `MPI_Isend`, `MPI_Waitall`)

The algorithms – summary

- Algorithms for inter-node communication and intra-node communication
- Inter-node communication algorithms implemented with respect to sockets
- Intra-node communication algorithms hierarchical
- Manual multi-radix and multi-port parametrisation
- Furthermore technical optimisations
- Compilation based

Recursive exchange



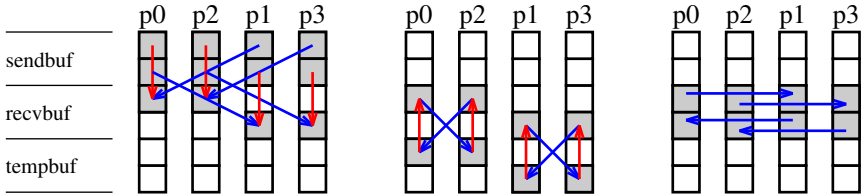
Recursive exchange radix $-4, -2, 2, 4$

Recursive exchange contd.

- Trade-off between latency and bandwidth for different message sizes
- Radices chosen empirically / experimentally

- Inter-node communication based on MPI point-to-point communication `MPI_Irecv`, `MPI_Isend`, `MPI_Waitall`

Recursive exchange contd.

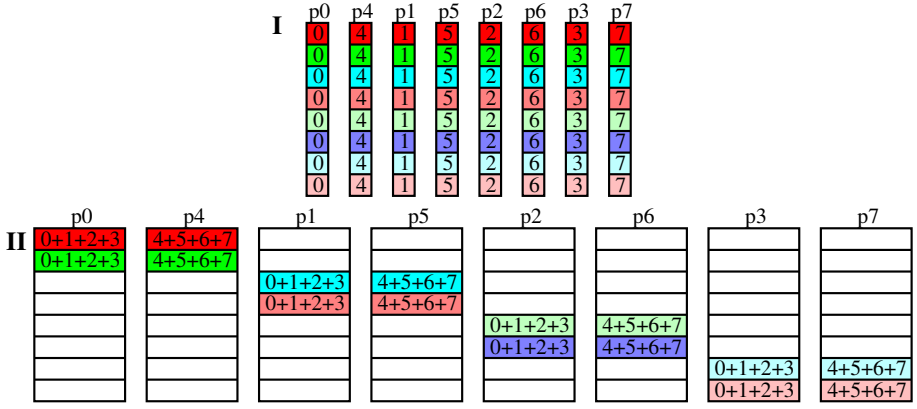


Recursive exchange, inter-node communication, radix $-2, 2, 2$

Recursive exchange contd.

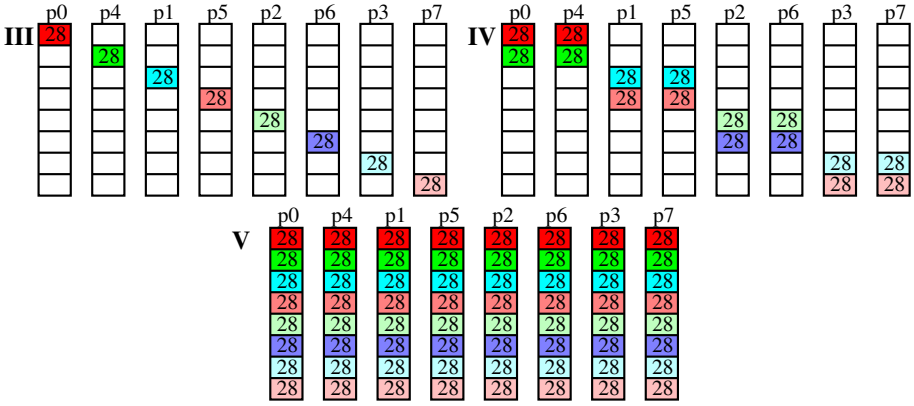
- Separate on the node communication
- No ‘latency’
- Barrier flags integrated into the algorithm

Recursive exchange contd.



Recursive exchange, intra-node communication, radix $-4, -2, 2, 4$

Recursive exchange contd.

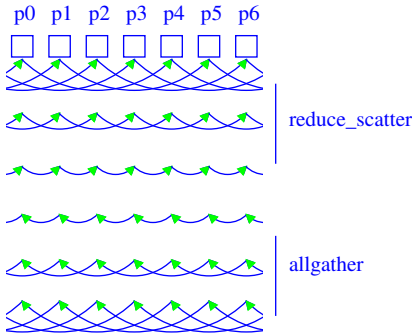


Recursive exchange, intra-node communication, radix $-4, -2, 2, 4$

Cyclic shift

- Suitable for any number of nodes independent of the radix, even large prime numbers of nodes possible
- Drawback: Might require one additional rearrangement of data within the node compared to recursive exchange

Cyclic shift contd.



Cyclic shift radix $-2, -2, -2, 2, 2, 2$

Assembler code

```
1  MEMORY_FENCE_STORE
2  SET_NODE_BARRIER
3  SMEMCPY RECVBUF[ 0 ]+ 0 SENDBUF[ 0 ]+ 0 4096
4  WAIT_NODE_BARRIER 1
5  MEMORY_FENCE_LOAD
6  SREDUCE RECVBUF[ 0 ]+ 0 SENDBUF[ 1 ]+ 0 4096
7  MEMORY_FENCE_STORE
8  SET_NODE_BARRIER
9  WAIT_NODE_BARRIER 2
10 MEMORY_FENCE_LOAD
11 SREDUCE RECVBUF[ 0 ]+ 0 RECVBUF[ 2 ]+ 0 2048
12 SMEMCPY SHMEM[ 0 ]+ 0 RECVBUF[ 0 ]+ 0 2048
13 SOCKET_BARRIER
14 IRECV SHMEM[ 0 ]+ 16384 4096 8 LOCMEM+ 0
15 ISEND SHMEM[ 0 ]+ 4096 4096 8 LOCMEM+ 1
16 WAITALL 2 LOCMEM+ 0
17 SOCKET_BSMALL
18 REDUCE SHMEM[ 0 ]+ 16384 SHMEM[ 0 ]+ 0 512
19 REDUCE SHMEM[ 0 ]+ 18432 SHMEM[ 0 ]+ 2048 512
20 SOCKET_BSMALL
21 IRECV SHMEM[ 0 ]+ 8192 2048 4 LOCMEM+ 0
22 ISEND SHMEM[ 0 ]+ 18432 2048 4 LOCMEM+ 1
23 WAITALL 2 LOCMEM+ 0
```

```
1  SOCKET_BSMALL
2  REDUCE SHMEM[ 0 ]+ 8192 SHMEM[ 0 ]+ 16384 512
3  SOCKET_BSMALL
4  IRECV SHMEM[ 0 ]+ 10240 2048 4 LOCMEM+ 0
5  ISEND SHMEM[ 0 ]+ 8192 2048 4 LOCMEM+ 1
6  WAITALL 2 LOCMEM+ 0
7  SOCKET_BSMALL
8  IRECV SHMEM[ 0 ]+ 12288 4096 8 LOCMEM+ 0
9  ISEND SHMEM[ 0 ]+ 8192 4096 8 LOCMEM+ 1
10 WAITALL 2 LOCMEM+ 0
11 SOCKET_BARRIER
12 SMEMCPY RECVBUF[ 0 ]+ 0 SHMEM[ 0 ]+ 8192 2048
13 SMEMCPY RECVBUF[ 2 ]+ 0 RECVBUF[ 0 ]+ 0 2048
14 MEMORY_FENCE_STORE
15 SET_NODE_BARRIER
16 WAIT_NODE_BARRIER 2
17 MEMORY_FENCE_LOAD
18 SMEMCPY RECVBUF[ 1 ]+ 0 RECVBUF[ 0 ]+ 0 4096
19 MEMORY_FENCE_STORE
20 SET_NODE_BARRIER
21 WAIT_NODE_BARRIER 1
22 MEMORY_FENCE_LOAD
23 RETURN
```

4 nodes, 4 tasks per node, CPU

Assembler code contd.

```
1  SET_NODE_BARRIER
2  WAIT_NODE_BARRIER 1
3  WAIT_NODE_BARRIER 2
4  WAIT_NODE_BARRIER 3
5  SMEMCPY RECVBUF[ 0 ]+ 0 SENDBUF[ 0 ]+ 0 8192
6  SREDUCE RECVBUF[ 0 ]+ 0 SENDBUF[ 1 ]+ 0 8192
7  SREDUCE RECVBUF[ 0 ]+ 0 SENDBUF[ 2 ]+ 0 8192
8  SREDUCE RECVBUF[ 0 ]+ 0 SENDBUF[ 3 ]+ 0 8192
9  SMEMCPY RECVBUF[ 1 ]+ 0 RECVBUF[ 0 ]+ 0 8192
10 SMEMCPY RECVBUF[ 2 ]+ 0 RECVBUF[ 0 ]+ 0 8192
11 SMEMCPY RECVBUF[ 3 ]+ 0 RECVBUF[ 0 ]+ 0 8192
12 SET_NODE_BARRIER
13 WAIT_NODE_BARRIER 1
14 WAIT_NODE_BARRIER 2
15 WAIT_NODE_BARRIER 3
16 RETURN
```

1 node, 4 tasks per node, 4 GPUs per node

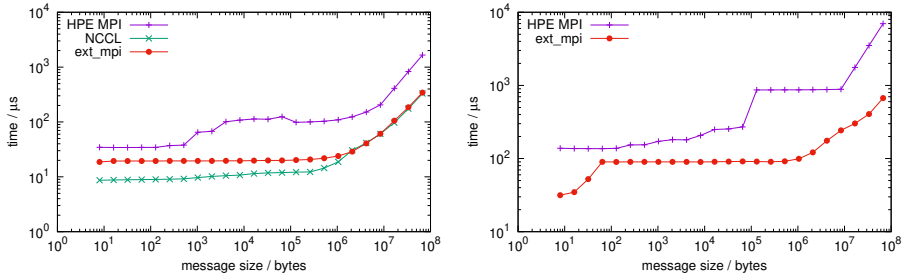
Benchmarks

Series of benchmarks

- Comparison between our `allreduce` routine from “ext_mpi” https://github.com/eth-cscs/ext_mpi_collectives and standard `MPI_Allreduce` as reference
- NCCL GPU benchmarks for further comparison
- Architecture: Grace-Hopper CPU/GPU, HPE Slingshot network

- Our library is still under development

Benchmarks

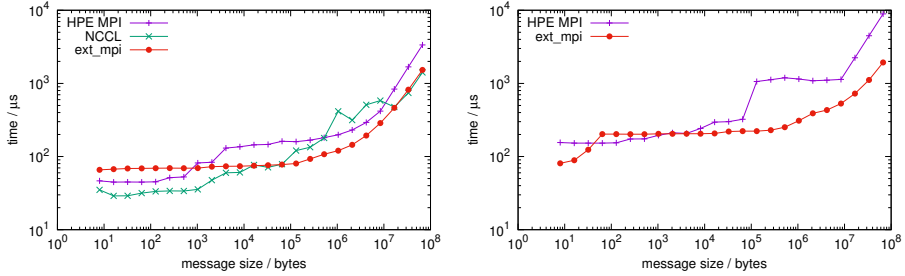


1 node 4 tasks (left) and 16 tasks (right), 4 sockets

Benchmarks contd.

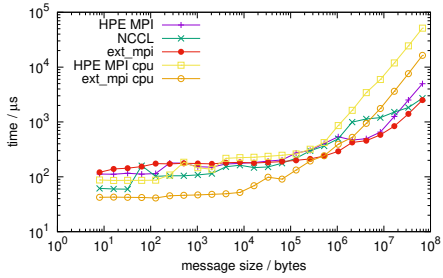
- For all message sizes our implementation is superior to HPE MPI
- For large message sizes our implementation is equally fast to NCCL
- We are slower than NCCL for small message sizes unless the ‘blocking’ switch is activated in the NCCL benchmarks which makes them equally fast to our implementation.

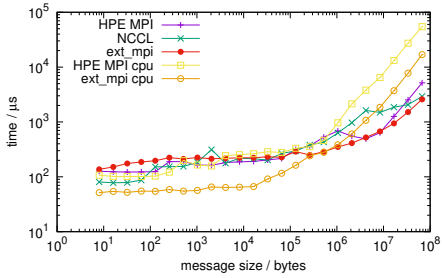
Benchmarks contd.



4 nodes 4 tasks per node (left) and 16 tasks per node (right), 4 sockets per node

Benchmarks contd.





32 nodes 4 tasks per node (left) and 64 nodes 4 tasks per node (right), 4 sockets per node

Benchmarks contd.

- For large message sizes between 10^6 and 10^8 bytes our implementation is superior to HPE MPI and NCCL
- (Our CPU implementation is faster then HPE MPI for all message sizes)
- For small message sizes
  - Further parameter search for our algorithm can help
  - The use of CUDA streams which are not utilised in our library should improve the results compared to NCCL
  - Computation of reductions could be done on CPU instead of GPU $\rightarrow$ then we are faster than the references for all message sizes

Conclusions

Literature

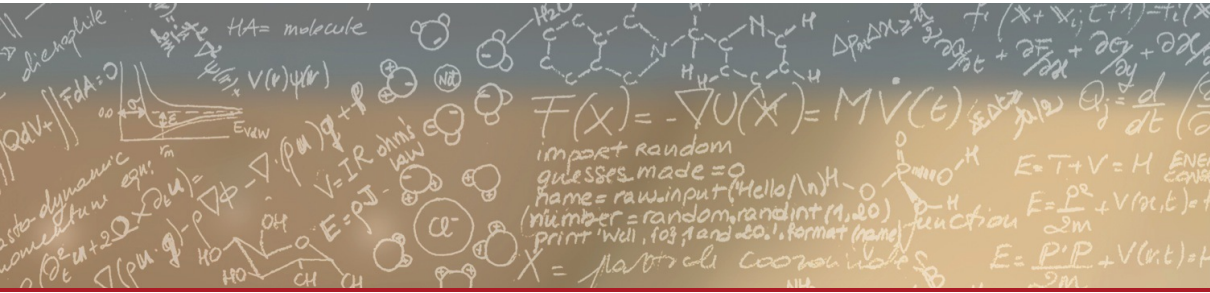
- Andreas Jocksch, C. Nicole Avans, Riley Shipley, and Anthony Skjellum. “A compilation-based approach to performant reduction and redistribution collective communication algorithms.” The International Journal of High Performance Computing Applications (2025): 40(2), pp. 219-239.
- Andreas Jocksch, Noé Ohana, Emmanuel Lanti, Eirini Koutsaniti, Vasileios Karakasis, and Laurent Villard. “An optimisation of allreduce communication in message-passing systems.” Parallel Computing 107 (2021): 102812.

Conclusions and outlook

- Allreduce collective communication for GPU equipped supercomputers has been introduced
- For large message sizes our library is equally fast to NCCL and faster than HPE MPI
- For medium message sizes our implementation is superior
- Small messages are an open topic, on the CPU our solution is also faster than the references
- Work in progress, library will be further developed

Acknowledgements

Noé Ohana, Emmanuel Lanti, Laurent Villard (Swiss Plasma Center, EPFL)



Thank you for your attention.

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