

The Allgather-merge Collective Operation: Algorithms, Implementation, Performance

Leonhard Ender Jesper Larsson Träff

EuroMPI 2026

October 8, 2026

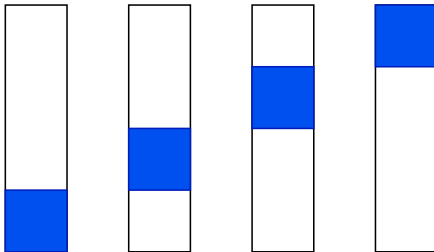
Allgather-merge

- ▶ p processes each receive a sorted array as input
- ▶ The total size across all processes is m
- ▶ Regular case: Every process starts with m/p data
- ▶ In the end, each process should have the result of merging all inputs (sorted array of size m)

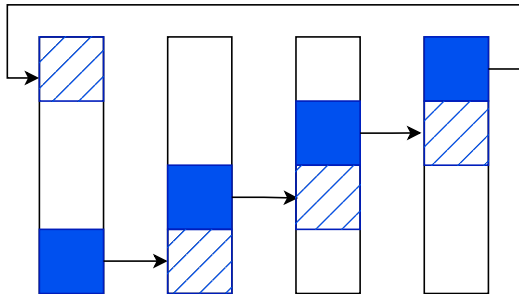
Baseline Solution

- ▶ Two separate steps:
 1. Regular Allgather (MPI)
 2. Merge p arrays
- ▶ Communication cost: $\alpha \lceil \log_2 p \rceil + \beta \frac{p-1}{p} m$
- ▶ Merge volume: $O(m \log_2 p)$
- ▶ Huge inefficiency: Every process does the merging on its own, doing redundant work

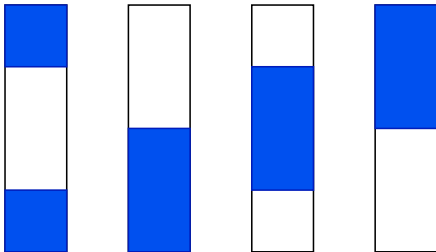
Bruck Allgather



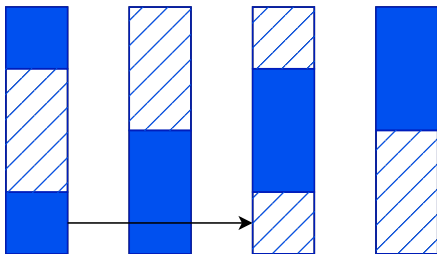
Bruck Allgather



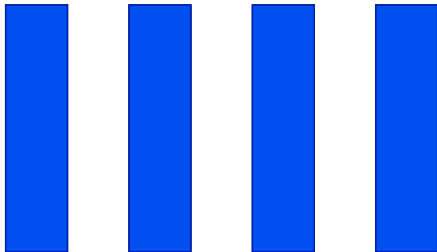
Bruck Allgather



Bruck Allgather



Bruck Allgather



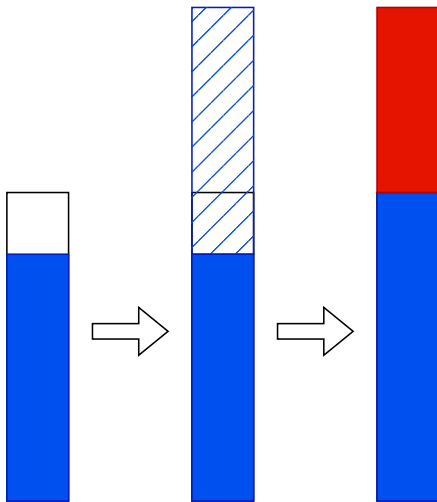
Bruck Allgather

- ▶ Well-known Allgather algorithm¹
- ▶ We can adapt it by merging between rounds
- ▶ However: when p is not a power of 2, we run into problems

¹J. Bruck et al. “Efficient algorithms for all-to-all communications in multiport message-passing systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 8.11 (1997), pp. 1143–1156. DOI: [10.1109/71.642949](https://doi.org/10.1109/71.642949).

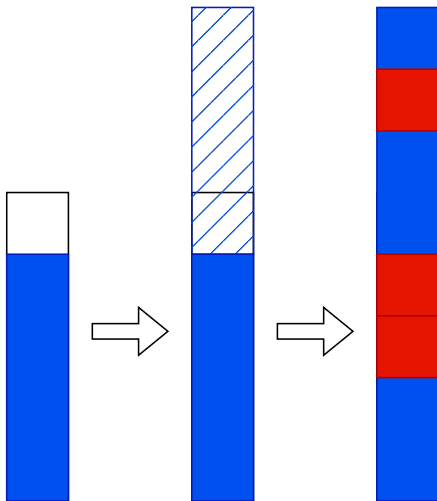
Bruck Allgather

When p is not a power of 2 for Allgather:



Bruck Allgather

When p is not a power of 2 for Allgather-merge:

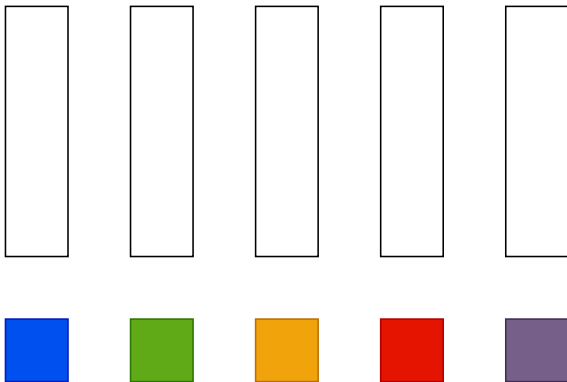


Existing Allreduce Algorithm

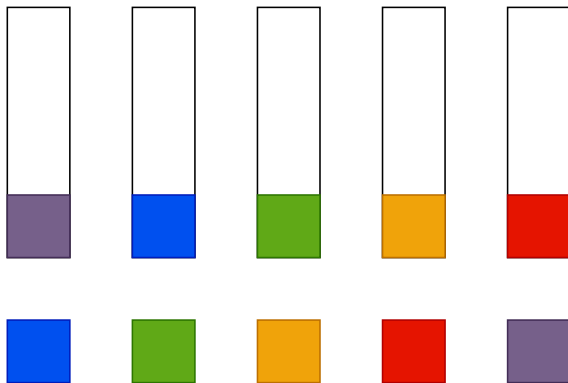
Jesper Larsson Träff. “Optimal, Non-pipelined Reduce-scatter and Allreduce Algorithms with an Application to All-to-all Communication”. In: *ACM Trans. Parallel Comput.* 12.4 (Dec. 2025). ISSN: 2329-4949. DOI: 10.1145/3760789. URL: <https://doi.org/10.1145/3760789>

- ▶ Switch to roughly-halving skip sequence
- ▶ Keep input separate from working array
- ▶ Calculate per-round adjustment term ε
- ▶ If $\varepsilon = 0$, merge input and working array and send results
- ▶ Else, just send working array

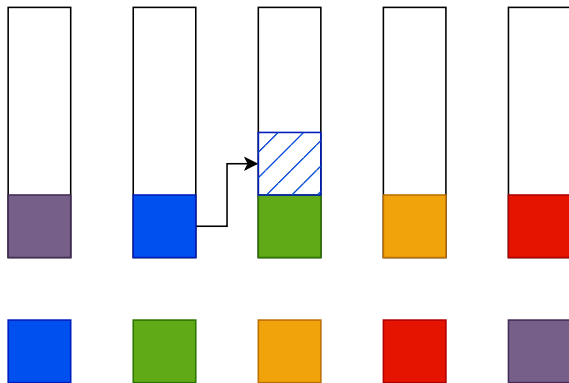
Existing Allreduce Algorithm



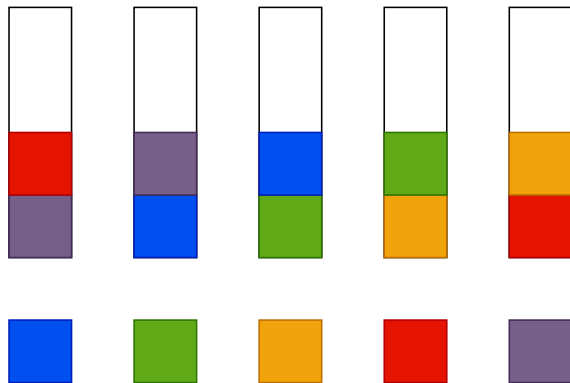
Existing Allreduce Algorithm



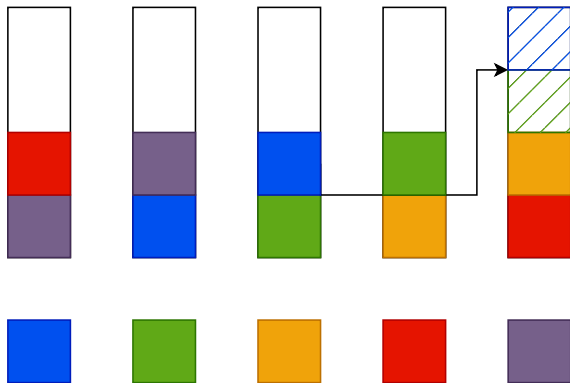
Existing Allreduce Algorithm



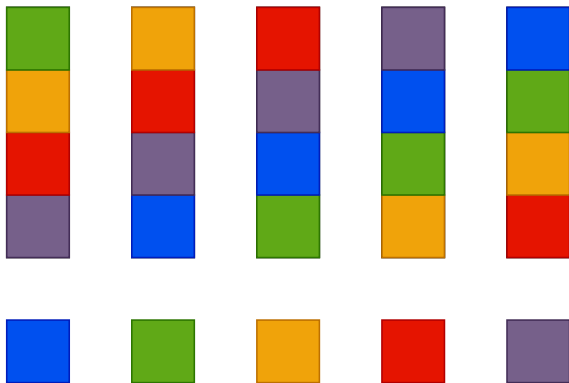
Existing Allreduce Algorithm



Existing Allreduce Algorithm



Existing Allreduce Algorithm



Circulant I

$$q = \log_2 p$$

skip sequence s_k is roughly halving

procedure ALLGATHERMERGECIRCULANTI

Calculate skip sequence s_k

for $k = 0, \dots, q - 1$ **do**

$\varepsilon \leftarrow s_{k+1} \bmod 2$

▷ *Adjustment term*

if $\varepsilon = 1$ **then**

 SendRecv(working array)

 TwoWayMerge(new, old)

else

 TwoWayMerge(input, working array) into buffer

 SendRecv(buffer)

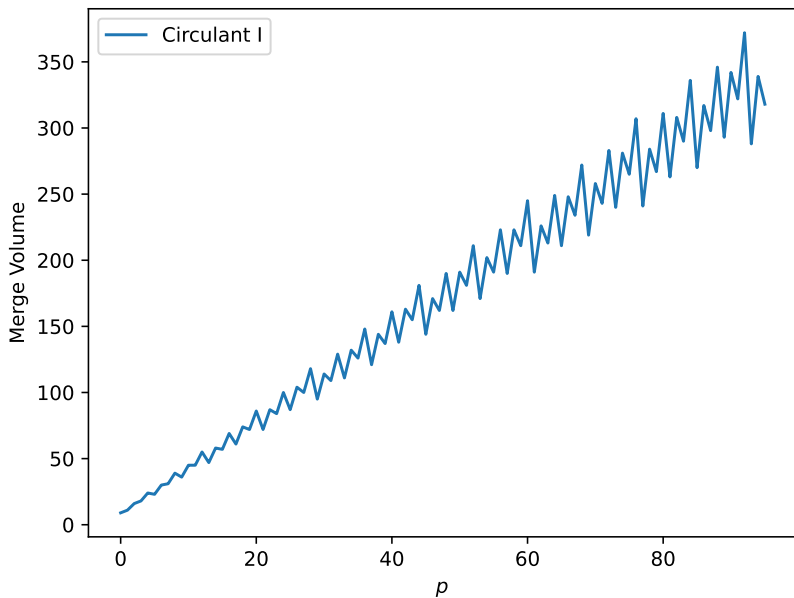
 TwoWayMerge(new, old)

TwoWayMerge(input, working array)

Circulant I: Complexity

- ▶ Communication cost: $\alpha \lceil \log_2 p \rceil + \beta \frac{p-1}{p} m$
Same as baseline; this is optimal
- ▶ Merge volume: $4m$
 - ▶ Regular merge: $m + m/2 + m/4 \dots \approx 2m$
 - ▶ Additional merge if $\varepsilon = 0$: $m/2 + m/4 \dots \approx m$
 - ▶ Final merge: m

Circulant I: Merge Volume



Circulant II

- ▶ When $\varepsilon = 0$, instead of an extra two-way merge at the sender, do a three-way merge at the receiver
- ▶ Merge volume for receiver stays the same, but extra merge for sender disappears

Circulant II

procedure ALLGATHERMERGECIRCULANTII

for $k = 0, \dots, q - 1$ **do**

$\varepsilon \leftarrow s_{k+1} \bmod 2$

 ▷ *Adjustment term*

if $\varepsilon = 1$ **then**

 SendRecv(working array)

 TwoWayMerge(new, old)

else

 SendRecv(working array + input)

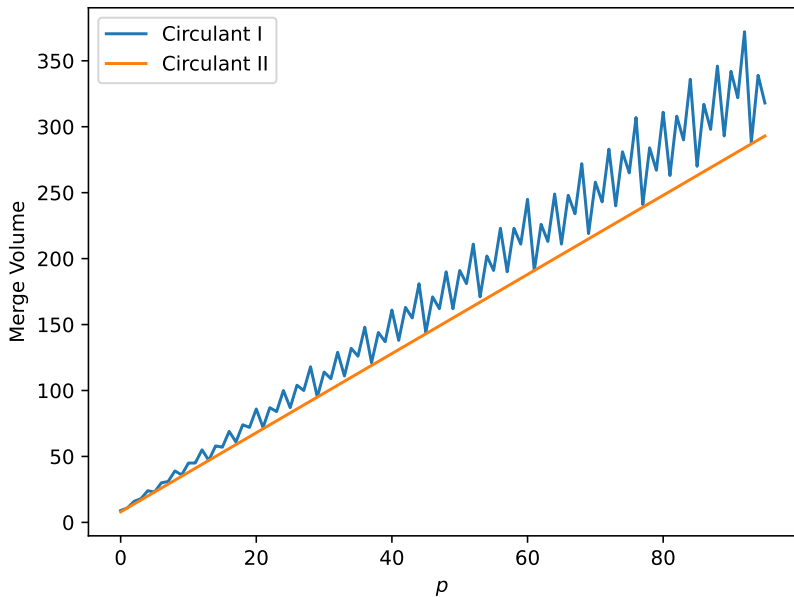
 ThreeWayMerge(new, new input, old)

TwoWayMerge(input, working array)

Circulant II: Complexity

- ▶ Communication cost stays the same
- ▶ Merge volume: $3m$
 - ▶ Regular merge: $m + m/2 + m/4... \approx 2m$
 - ▶ Final merge: m

Circulant II: Merge Volume



Circulant III: Complexity

- ▶ Final merge causes significant merge cost
- ▶ We can avoid this by integrating it into the last round

Circulant III

procedure ALLGATHERMERGECIRCULANTIII

for $k = 0, \dots, q - 1$ **do**

$\varepsilon \leftarrow s_{k+1} \bmod 2$

 ▷ *Adjustment term*

if $\varepsilon = 1$ **then**

 SendRecv(working array)

if last round **then**

 ThreeWayMerge(new, old, input)

else

 TwoWayMerge(new, old)

else

 SendRecv(working array + input)

if last round **then**

 TwoWayMerge(new input, input) into buffer

 ThreeWayMerge(new, buffer, old)

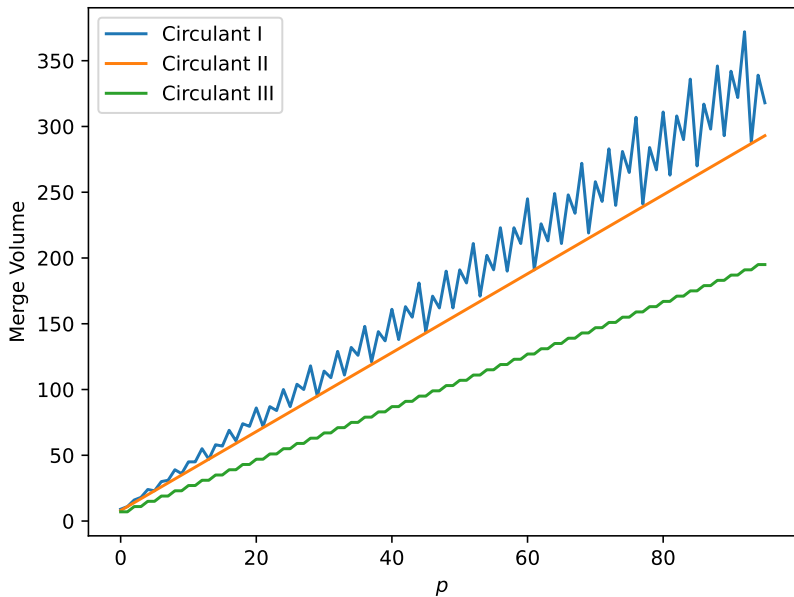
else

 ThreeWayMerge(new, new input, old)

Circulant III: Complexity

- ▶ Communication cost stays the same
- ▶ Merge volume: $2m$
 - ▶ Regular merge: $m + m/2 + m/4... \approx 2m$

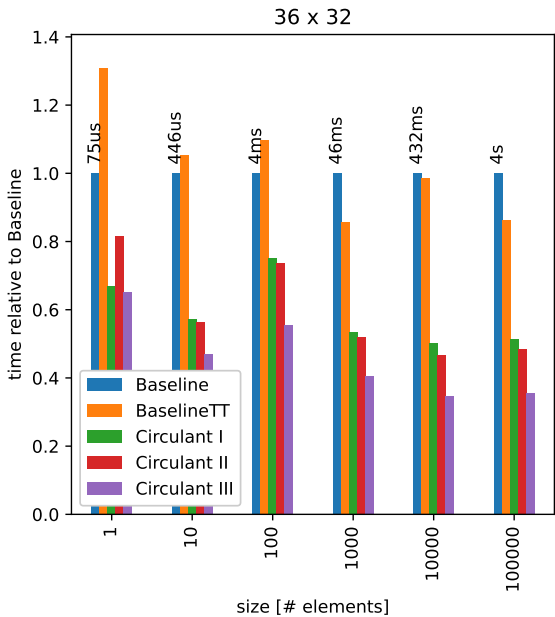
Circulant III: Merge Volume



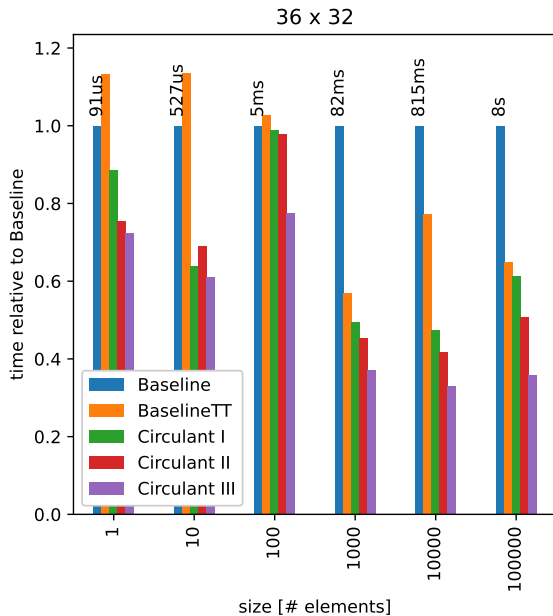
Benchmarks: Setup

- ▶ Hydra at TU Wien
- ▶ $36 \times 32 = 1152$ processes
- ▶ Connected with Intel OmniPath
- ▶ Three different data initializations

Benchmarks: int



Benchmarks: double



Conclusion

- ▶ We demonstrated the practical applicability of Träff's 2025 Allreduce algorithm
- ▶ We provide an efficient implementation of Allgather-merge as a MPI-style Collective
- ▶ It turns out that when adapting the general algorithm for our specific case, it is worth considering its unique characteristics

References

- [1] J. Bruck et al. “Efficient algorithms for all-to-all communications in multiport message-passing systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 8.11 (1997), pp. 1143–1156. DOI: 10.1109/71.642949.
- [2] Jesper Larsson Träff. “Optimal, Non-pipelined Reduce-scatter and Allreduce Algorithms with an Application to All-to-all Communication”. In: *ACM Trans. Parallel Comput.* 12.4 (Dec. 2025). ISSN: 2329-4949. DOI: 10.1145/3760789. URL: <https://doi.org/10.1145/3760789>.