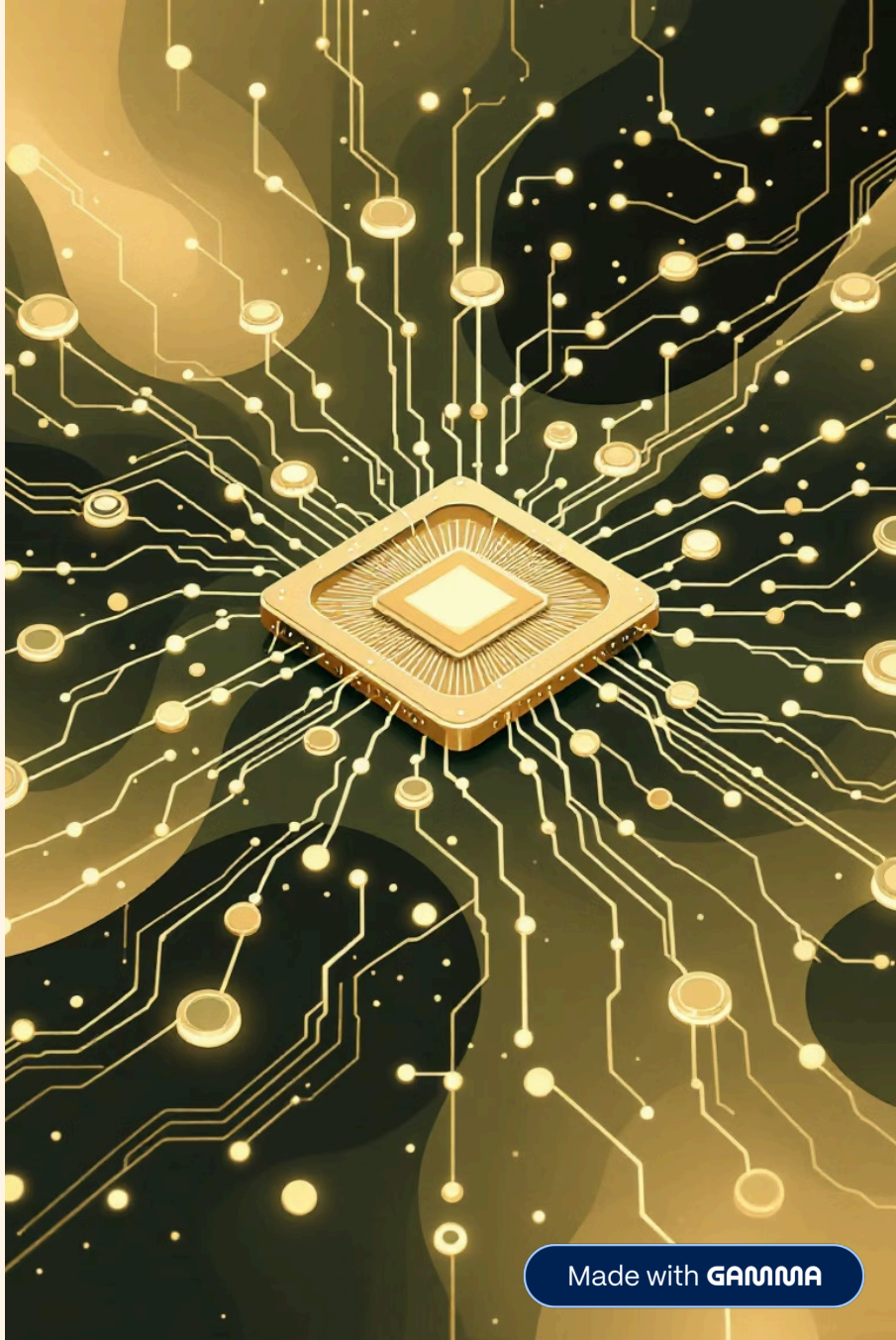
Optimal, Non-pipelined Reduce-scatter and Allreduce Algorithms

Breaking the "Power-of-Two" Constraint in MPI Collectives

Implementation Guide & Algorithmic Analysis



The Core Problem: Efficient Collective Operations

Collective operations, such as `MPI_Allreduce`, are fundamental to high-performance parallel computing. They enable synchronized data aggregation and distribution across all participating processors.

The Goal: Uncompromising Efficiency

Logarithmic Latency

(Speed → O(log2(p))

Linear Bandwidth

(Volume → O(p)

Any Processor Count

(p processors)

The Challenge: Overcoming Trade-offs

Standard algorithms often force a compromise between achieving optimal speed and maintaining generality across various processor configurations.

Standard Approaches: Strengths and Weaknesses

Ring Algorithm

Advantage: Works for any number of processors (p).

Disadvantage: Suffers from slow linear latency (O(p)), making it inefficient for large processor counts.

Hypercube/Recursive Doubling

Advantage: Achieves fast logarithmic latency (O(log p)), ideal for rapid data exchange.

Disadvantage: Restricted to scenarios where p is a perfect power of two, limiting applicability.

This disparity highlights a significant gap: the need for an algorithm that combines the speed of Hypercube with the universality of the Ring algorithm.

The Proposed Solution: A Novel Reduce-Scatter Algorithm

Proposed Reduce-Scatter

Our proposed Reduce-Scatter algorithm utilizes a specific graph structure: a $\lceil \log_2 p \rceil$-regular circulant graph. This design enables a balanced distribution of communication across processors.

Key Mechanism: The "Roughly Halving" Skip Sequence

The core innovation lies in a dynamic skip sequence, ensuring efficient data partitioning regardless of the processor count.

- The skip value `s` is iteratively updated by `s <- ceil(s/2)`.
- This "roughly halving" strategy ensures convergence and balanced message sizes.

This approach guarantees **optimal latency and bandwidth** for **any** number of processors, effectively breaking the traditional "power-of-two" constraint.

Phase 1: Reduce-Scatter - The Rotated Copy (Initialization)

```
procedure PARTITIONED_ALLREDUCE (V[p], W)
  W <- V[r]
  for i = 1,..., p-1 do
    R[i] <- V[(r + i) mod p]
  end for
```

- This initialization phase performs a critical "rotated copy" of the data.
- `W` serves as an alias for `R[0]`, pointing to the initial block on the current processor.
- The buffer `R` is allocated to hold `p` blocks of data.
- The input block `V[i]` on processor `r` is ultimately destined for the final result block `W[i]` on processor `i`.

Invariant: The partial result buffer `R[i]` on processor `r` consistently holds the partial sum for the block that will eventually reside on processor `(r+i) mod p`.

Phase 1: Reduce-Scatter - The Communication Loop

```
s <- p
while s > 1 do
  s', s <- s, ceil(s/2)
  t, f <- (r+s) mod p, (r-s+p) mod p
  Send(R[s...s'-1], t) | Recv(T[0...s'-s-1], f)
```

- **Skip Calculation:** The variable `s'` temporarily stores the previous skip value, while `s` is updated to **ceil(s/2)**, embodying the "roughly halving" strategy.
- **Communication:** Each processor sends its relevant partial sums, covering "far" destinations (`R[s ... s'-1]`), to the calculated target processor `t`.
- **Simultaneous Receive:** Concurrently, it receives `s' - s` blocks into a temporary buffer `T` from the source processor `f`, maximizing communication efficiency.

This loop is the heart of the algorithm, orchestrating the parallel reduction and scattering of data across the network.

Phase 1: Reduce-Scatter - Local Reduction Logic

```
  W <- W + T
  for i = 1,..., s'-s-1 do
    R[i] <- R[i] + T[i]
  end for
end while
end procedure
```

- **Immediate Reduction:** Upon receiving data, the blocks in temporary buffer `T` are immediately reduced into the local `R` buffer, ensuring partial sums are updated efficiently.
- `W` represents the partial sum that contributes to the local processor's final result block, undergoing continuous accumulation.

Termination: The loop concludes when `s=1`. At this point, the invariant ensures that `W` holds the complete, globally reduced result for its specific block, aggregated from all `p` processors.

Phase 2: Allgather (The Reverse Process)

To complete the Allreduce operation, Phase 2 **reverses the Reduce-Scatter process**, distributing the final reduced blocks to all processors.

```
while S != [empty_stack] do
  s' <- POP(S)
  f, t <- (r+s) mod p, (r-s+p) mod p
  Send(R[0...s'-s-1], t) | Recv(R[s...s'-1], f)
  s <- s'
end while
```

- **Stacked Skips:** This phase utilizes a stack `S` that stores the skip values used during Phase 1. Popping these values retrieves the sequence in reverse: `1, 2, 3, 6, 11...` (a doubling sequence).
- **Direct-to-Buffer:** Crucially, unlike Phase 1, data is received directly into the main buffer slots (`R[s...s'-1]`), eliminating the need for a temporary buffer `T`.

This careful reversal ensures that each processor obtains the complete and correct final reduction for all blocks.

Comparison of Collective Reduction Algorithms

Algorithm	Latency	Volume	Works for any p?	Commutativity
Ring	$O(p)$ (Linear)	$O(p)$ (Optimal)	Yes	Yes
Hypercube	$O(\log p)$ (Optimal)	$O(p)$ (Optimal)	No (only $p = 2^k$)	No
This Paper (Alg 1: Reduce-Scatter)	$\lceil \log_2 p \rceil$ (Optimal)	$p - 1$ (Optimal)	Yes	Yes
This Paper (Alg 2: Allreduce)	$2\lceil \log_2 p \rceil$ (Optimal)	$2(p - 1)$ (Optimal)	Yes	Yes

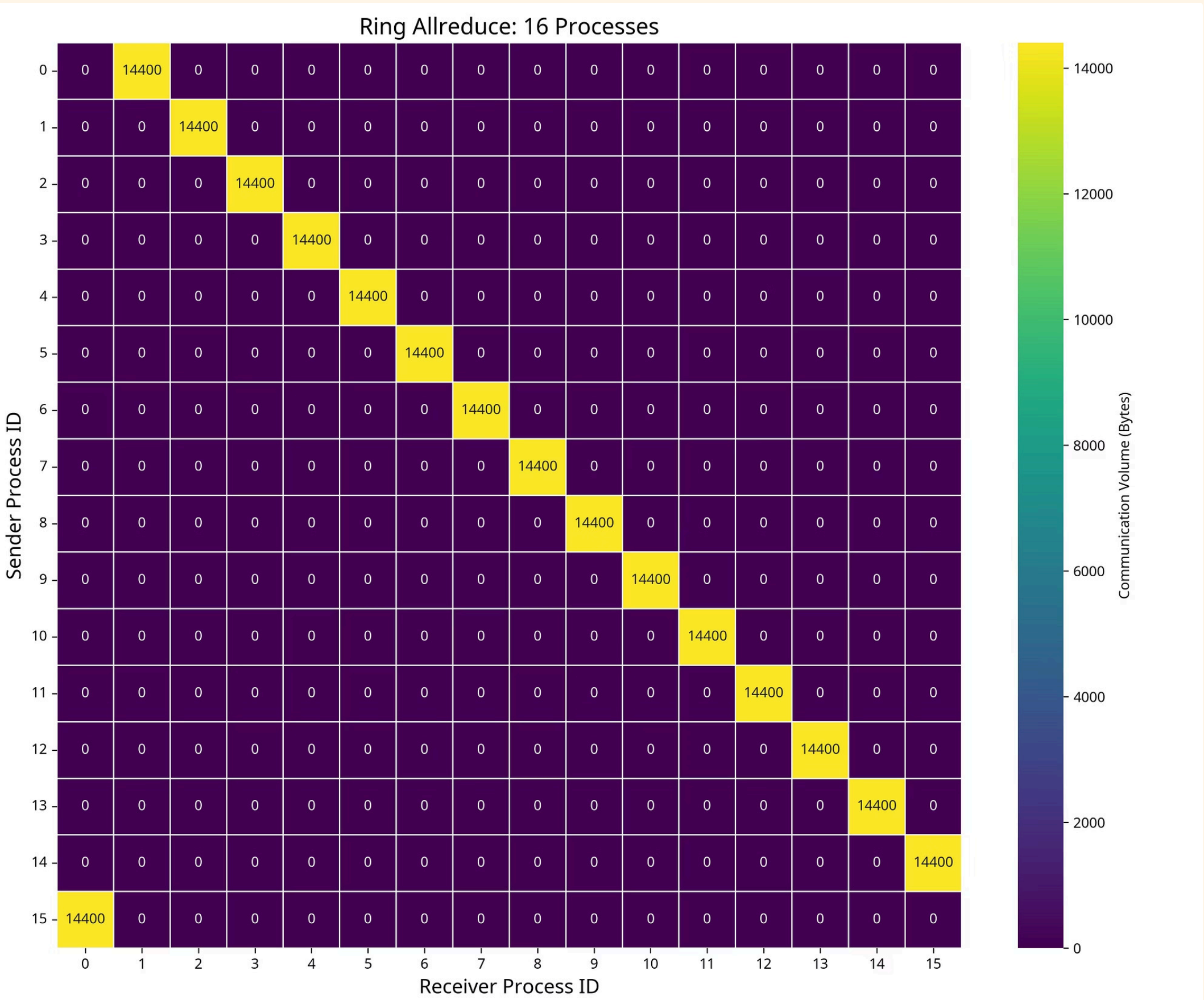
Conclusion: The Best of Both Worlds

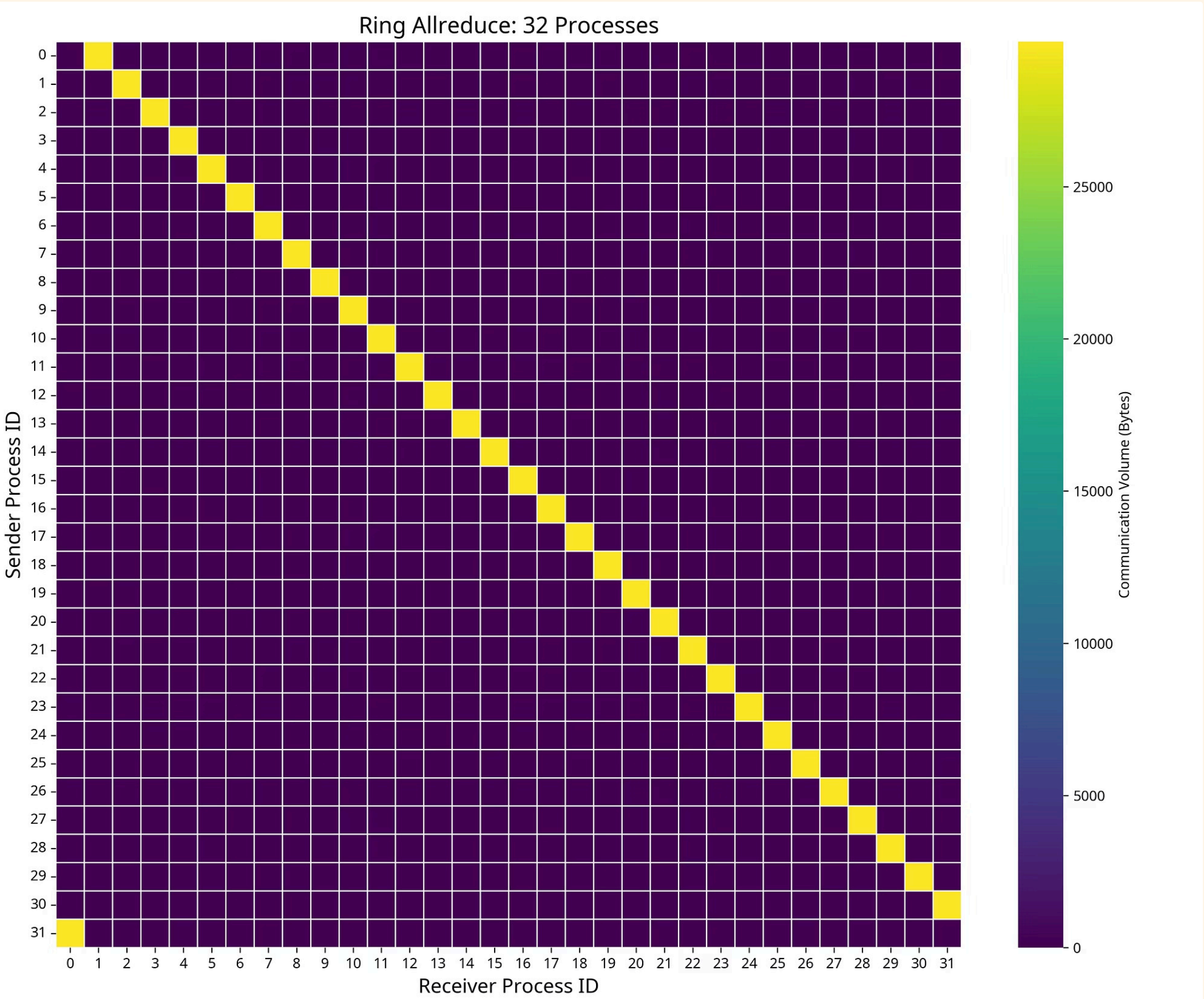
Our proposed algorithm effectively **breaks the traditional trade-off** between latency and generality in MPI collective operations. It combines the optimal logarithmic latency of Hypercube-based approaches with the universal applicability of the Ring algorithm.

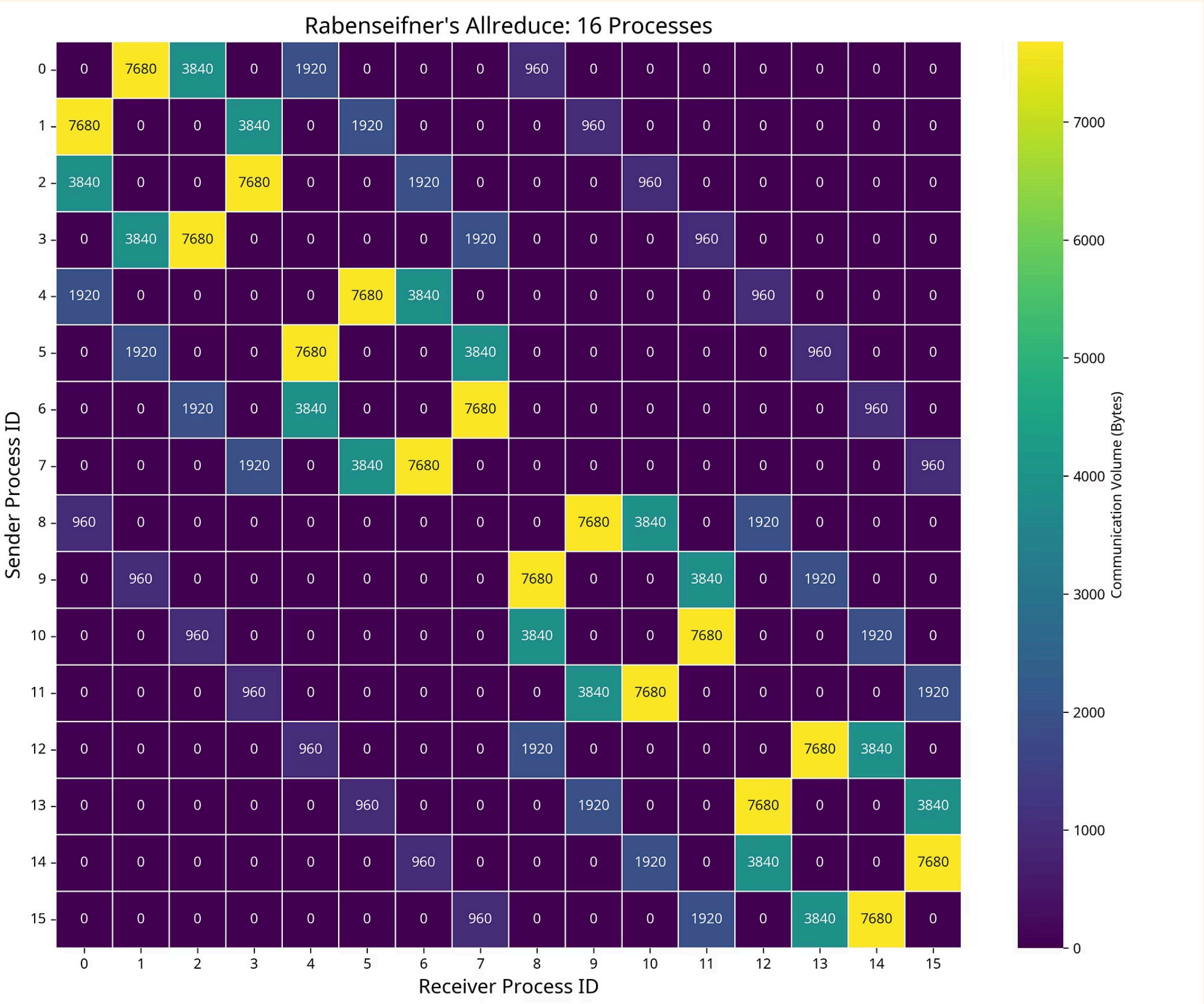
Constraints & Limitations

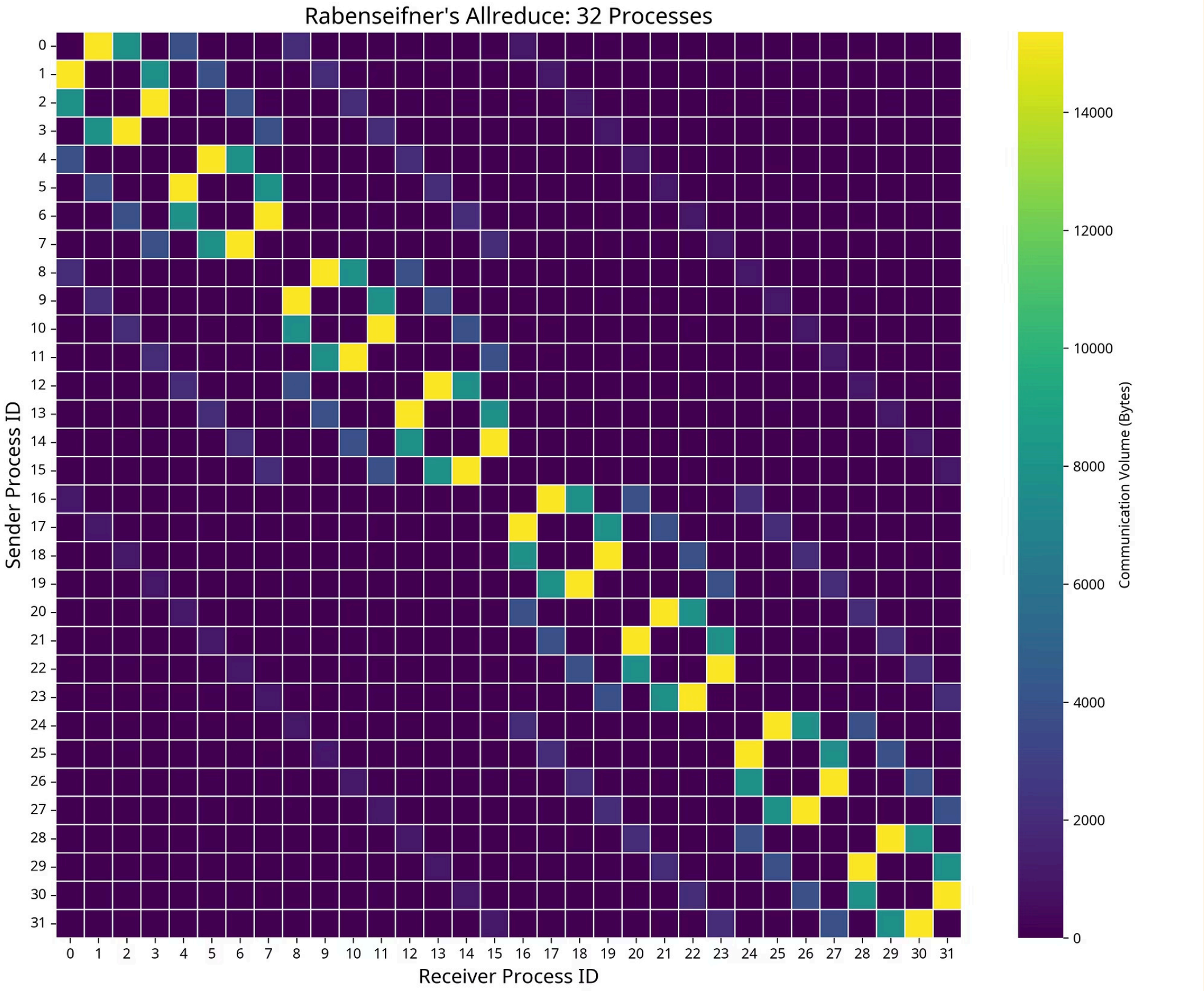
Commutative Operators Only

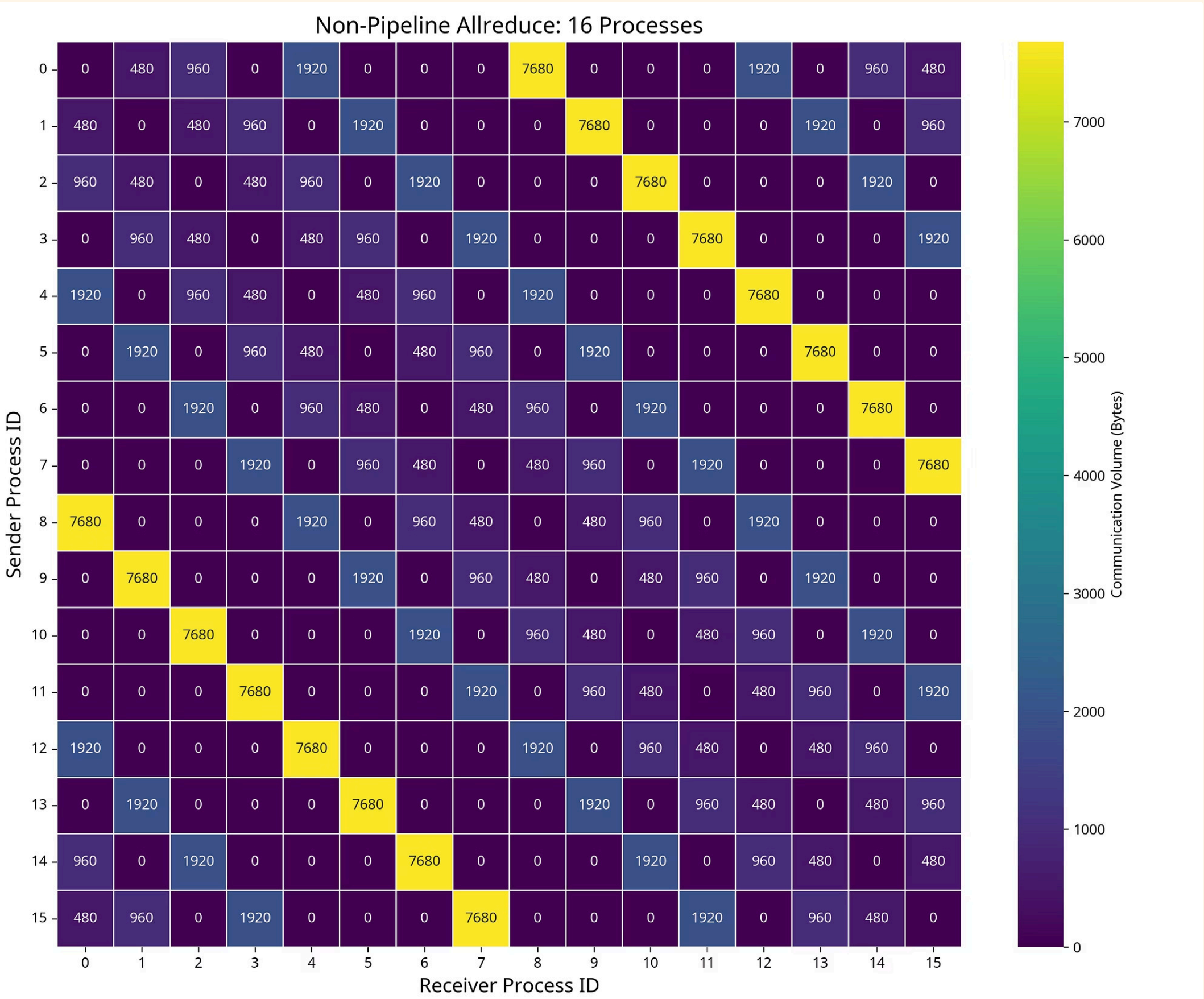
The algorithm relies on the operator being commutative (e.g., sum, maximum, minimum). This is because the order of reduction is non-standard due to the dynamic skip sequence, which could lead to incorrect results with non-commutative operations.

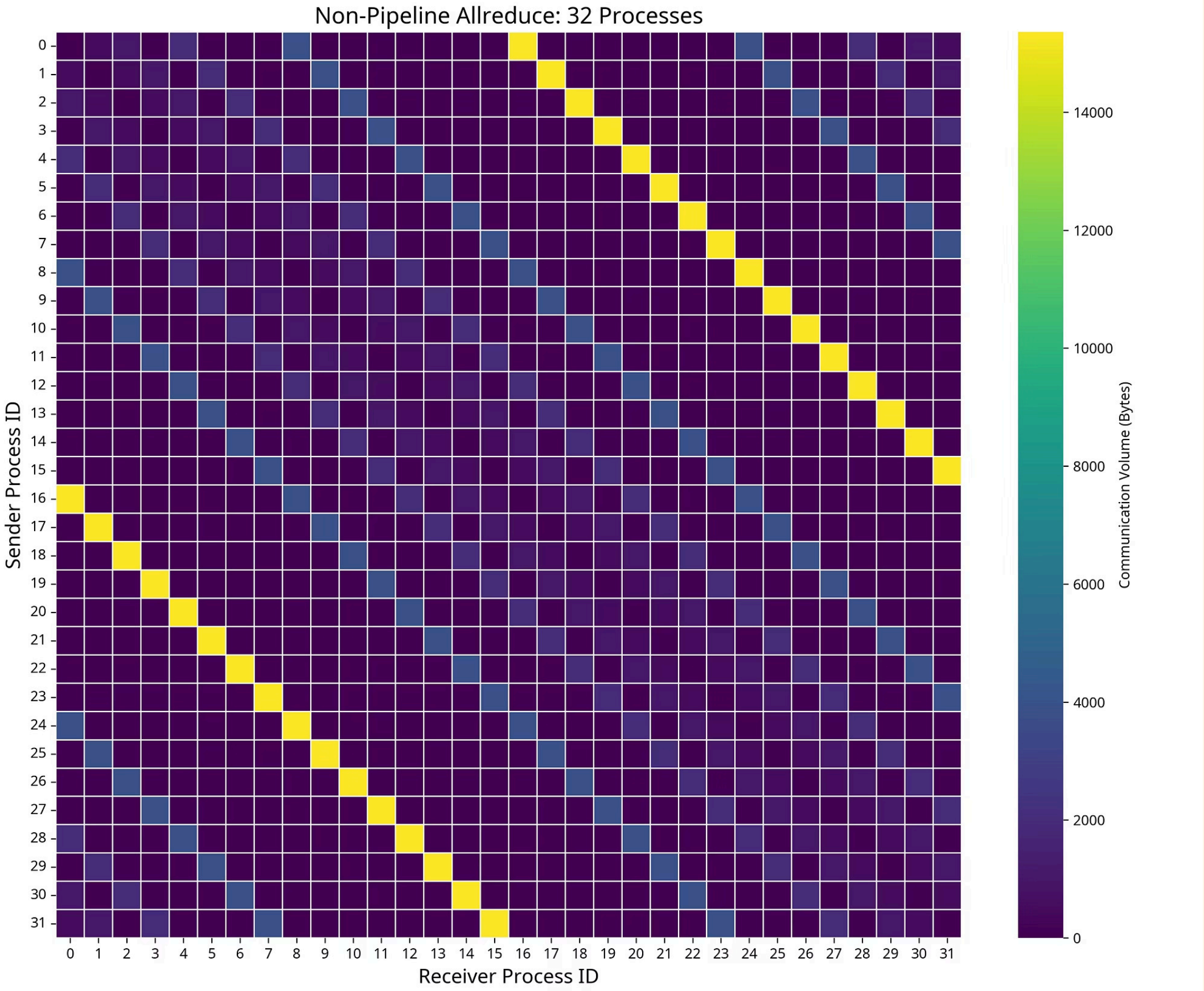


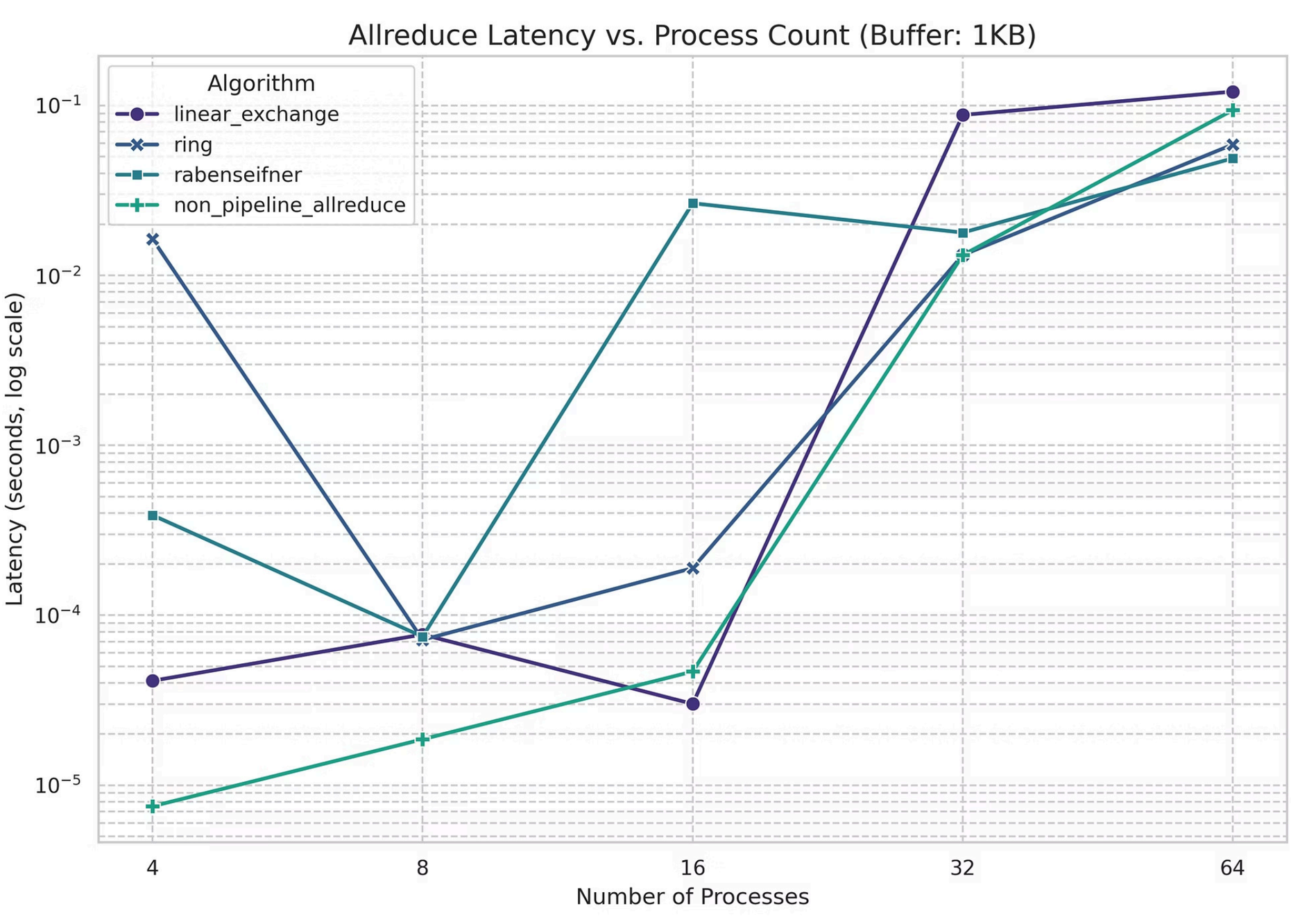


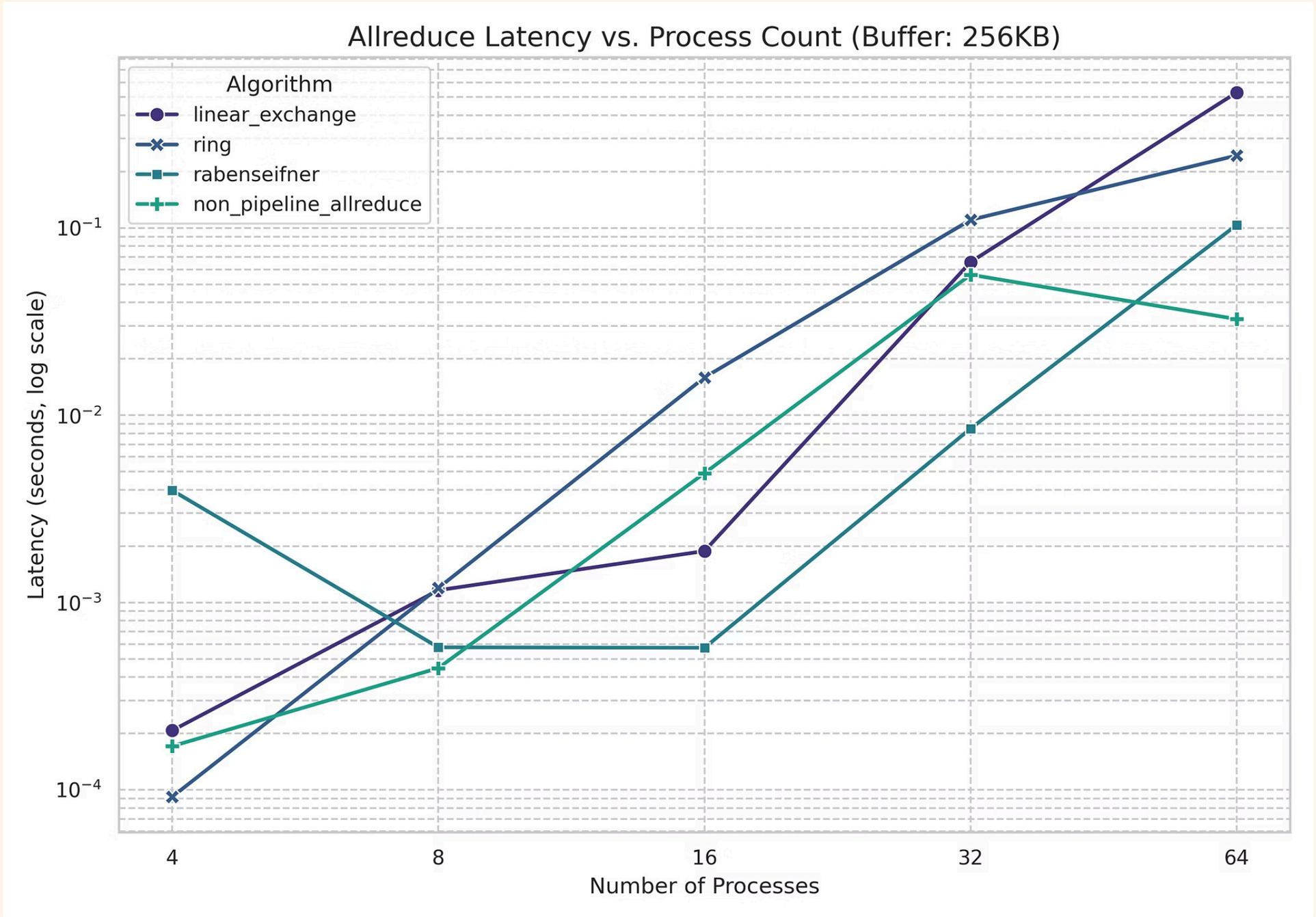


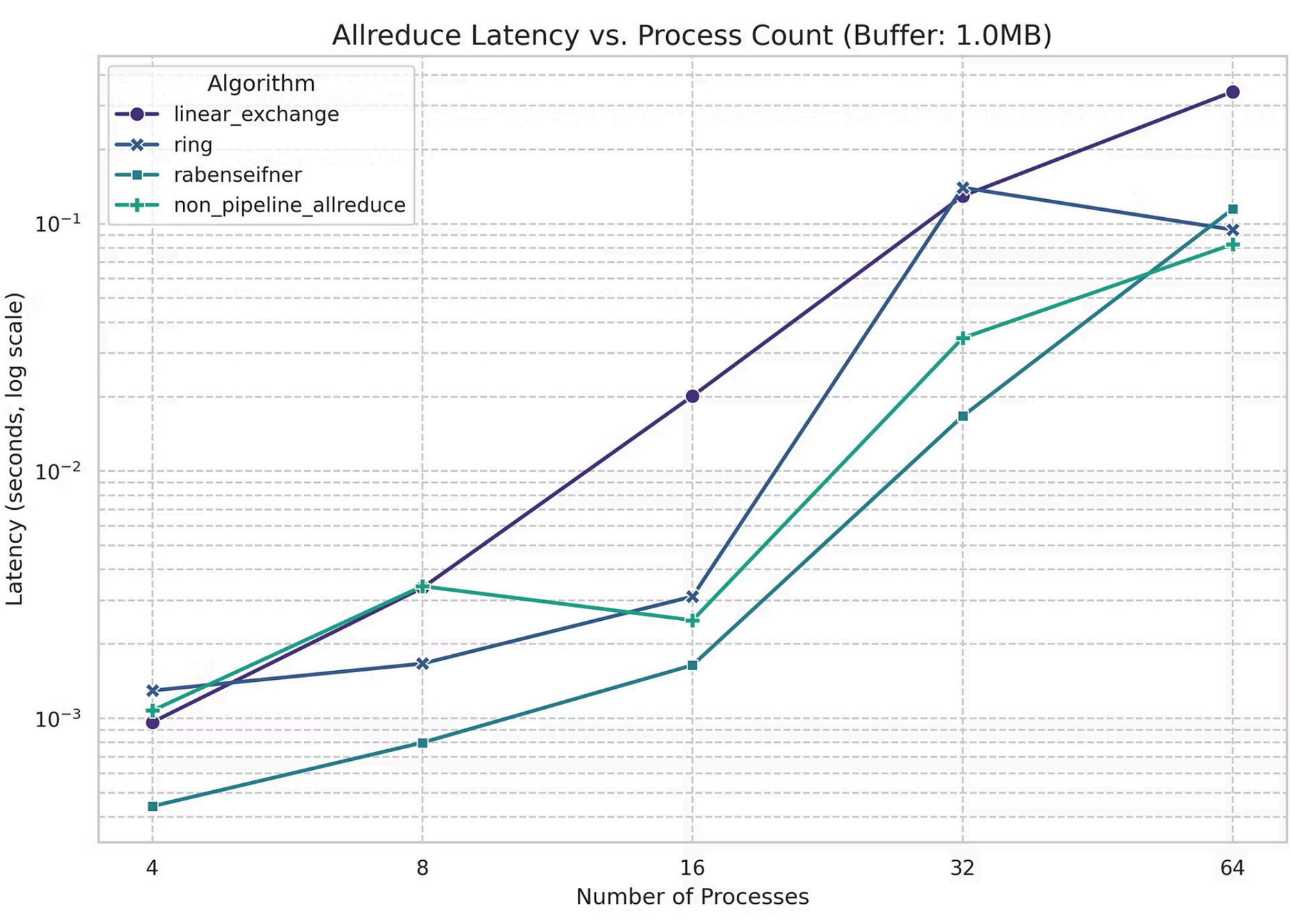


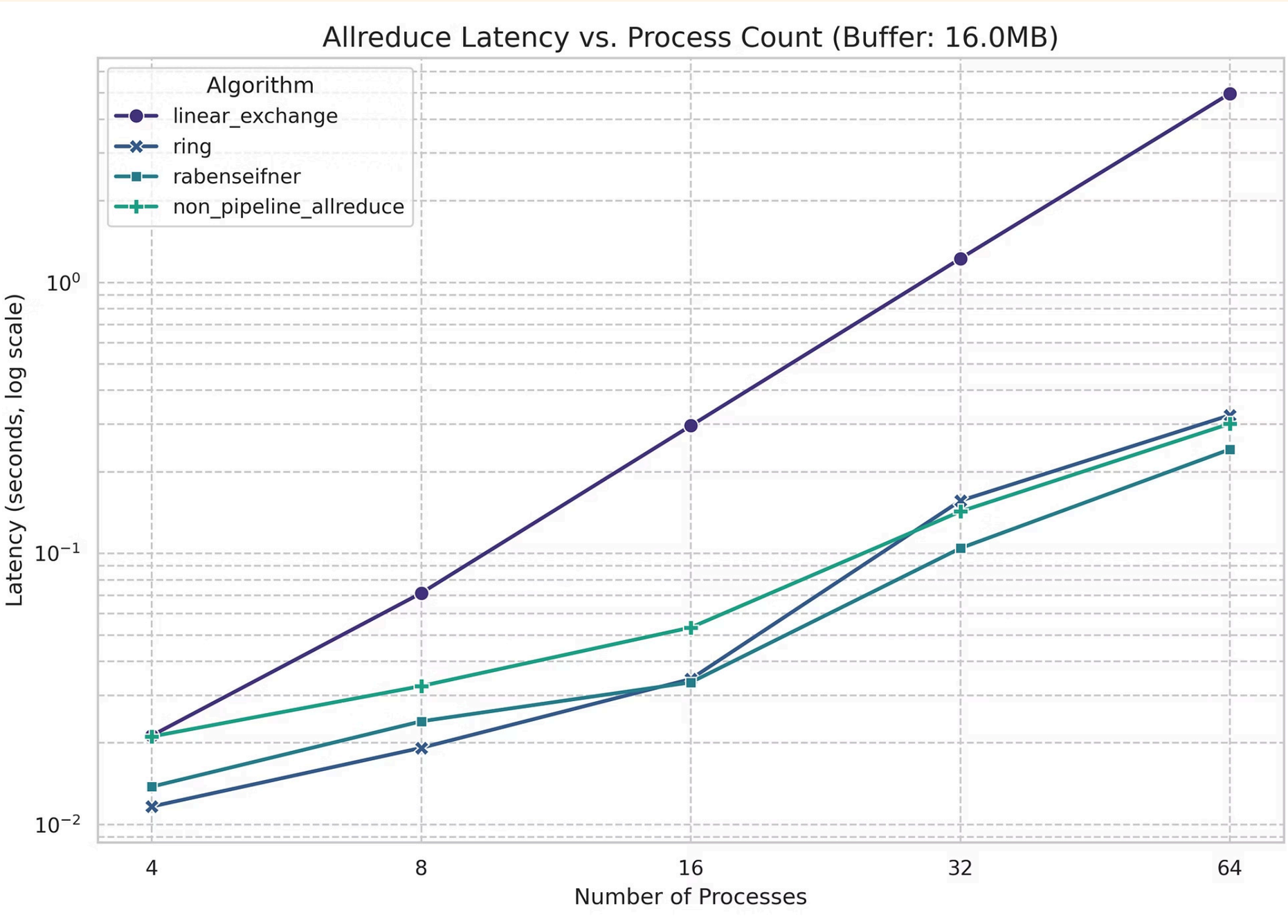


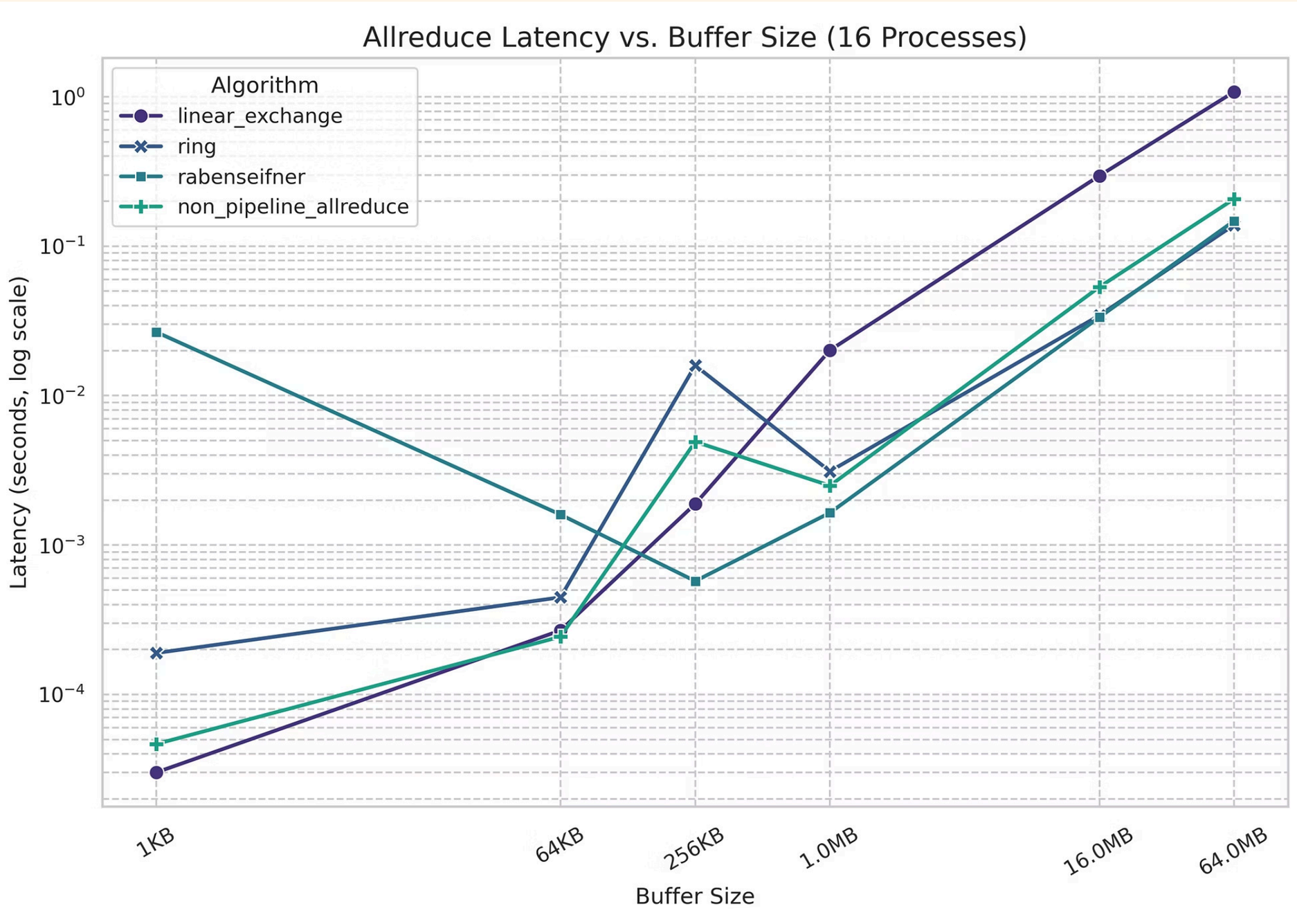


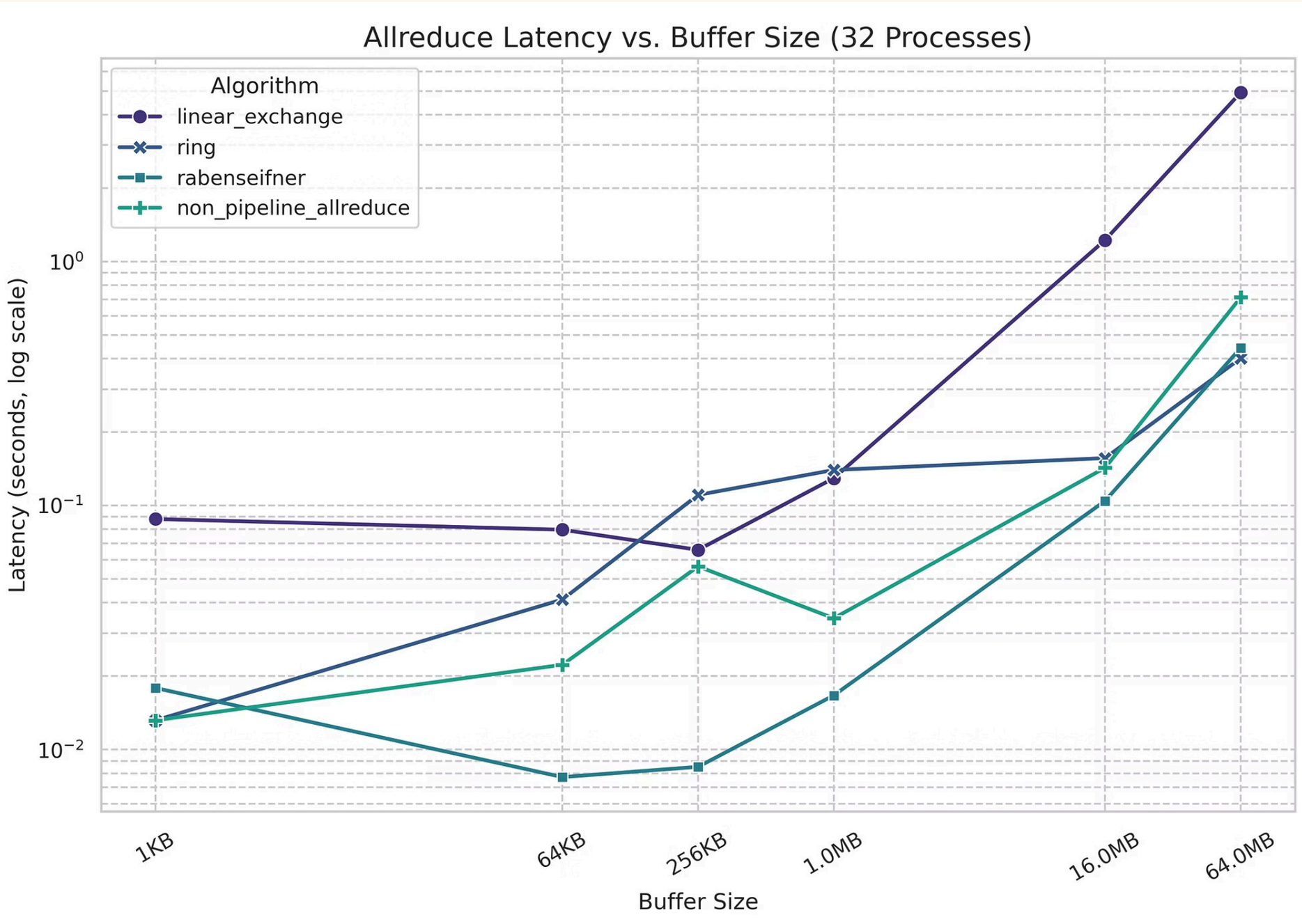


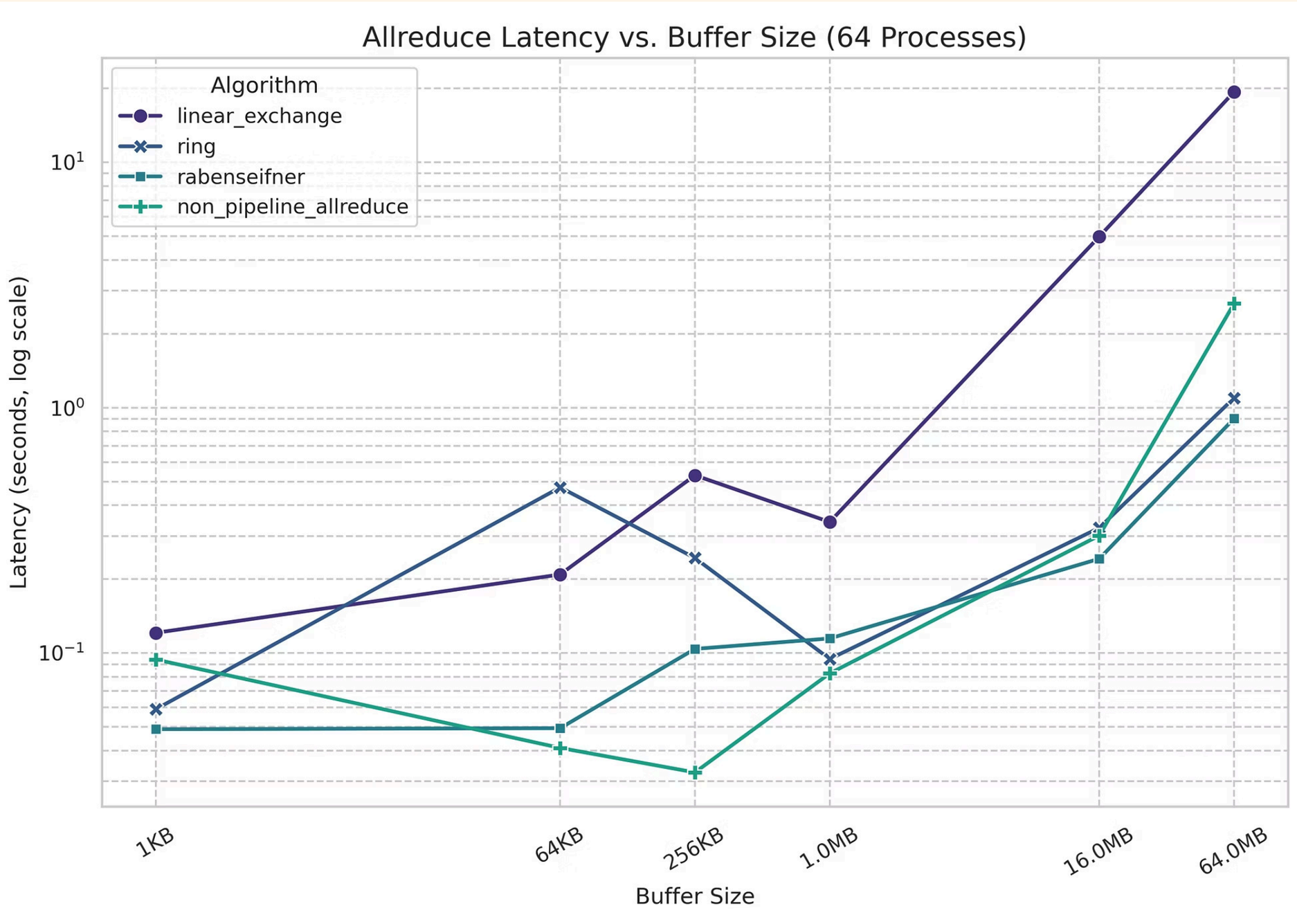












Theorem 1: Proving Optimality

Mathematical Proof for Latency and Bandwidth

The Claims

Latency: Optimal $\lceil \log_2 p \rceil$ rounds. The "roughly halving" scheme guarantees convergence in logarithmic steps.

Bandwidth: Optimal $p - 1$ blocks. Each processor sends and receives exactly the minimum data required.

The Proof - Telescoping Series

Show that the number of blocks sent in round k is

$$(s_k - s_{k+1})$$

The Total Volume is the sum over all rounds:

$$\sum (s_k - s_{k+1})$$

Expand this as a Telescoping Series:

$$= (s_0 - s_1) + (s_1 - s_2) + \cdots + (s_{q-1} - s_q)$$

Cancel intermediate terms to get the result:

$$= s_0 - s_q = p - 1$$