Report

Using MPI to efficiently distribute GEMM computations

Made by

Jules Gourio
Félix Foucher de Brandois

ENSEEIHT - Formation ModIA, 5th year *2024-2025*

1 Introduction

In this project, we focus on the distributed implementation of the General Matrix Multiplication (GEMM) algorithm using the Message Passing Interface (MPI). The goal is to efficiently distribute the computation of matrix multiplication across multiple processes, leveraging parallel computing to enhance performance. GEMM is a fundamental operation defined as:

$$C = \alpha \cdot op_A(A) \times op_B(B) + \beta \cdot C \tag{1}$$

where op are either identity or transposition, α and β scalars and A, B, C matrices.

For simplicity, we restrict our study to the case where $\alpha = 1$, $\beta = 0$, and op_A and op_B are the identity operation, reducing the operation to $C = A \times B$.

The matrices A, B, and C are dense and distributed across a network of nodes using a 2D Block-Cyclic (2DBC) distribution pattern. This distribution ensures that each block of the matrices is assigned to a specific node based on its position in a logical $p \times q$ grid.

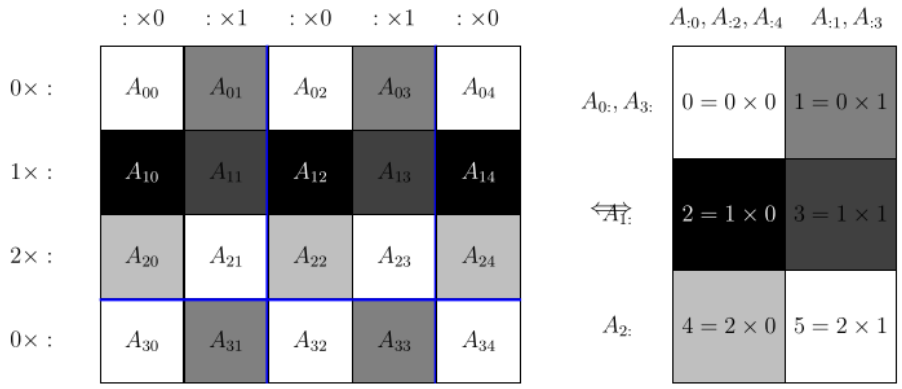

Figure 1: A split matrix distributed on nodes ⇔ the grid of nodes and their affected column/row combination of blocks

The goal of this project is to implement three variations of a distributed GEMM algorithm using different MPI communication strategies and then benchmark their performance across a range of problem sizes and node configurations.

1.1 Blocking peer-to-peer communications

This variant uses `MPI_Ssend` and `MPI_Recv` to transmit required blocks between processes. Each node sends the blocks it owns to other nodes in its row (for A) or column (for B), then performs local computation. Communication and computation are strictly serialized.

Using VITE, we can visualize the blocking peer-to-peer communication pattern for GEMM. We set the number of nodes to $p = q = 2$, the matrix size to $m = n = k = 10$ and the number of blocks per node to $b = 5$.

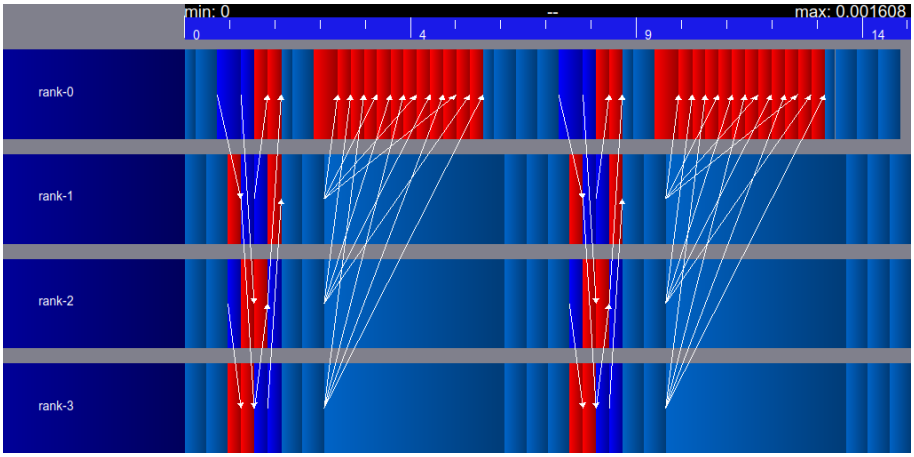

Figure 2: Blocking peer-to-peer communication for GEMM

Clear vertical barriers appear where all processes must wait for communication to complete before proceeding to computation. Some processes finish their communication earlier but must wait for others, leading to inefficient resource utilization.

1.2 Blocking collective communications

Here, `MPI_Bcast` is used instead of multiple individual sends. A node that owns a block of A or B broadcasts it to all nodes in the same row or column communicator. This reduces the number of messages and synchronizes communication across each dimension.

Using VITE, we can visualize the blocking collective communication pattern for GEMM in the same configuration as before.

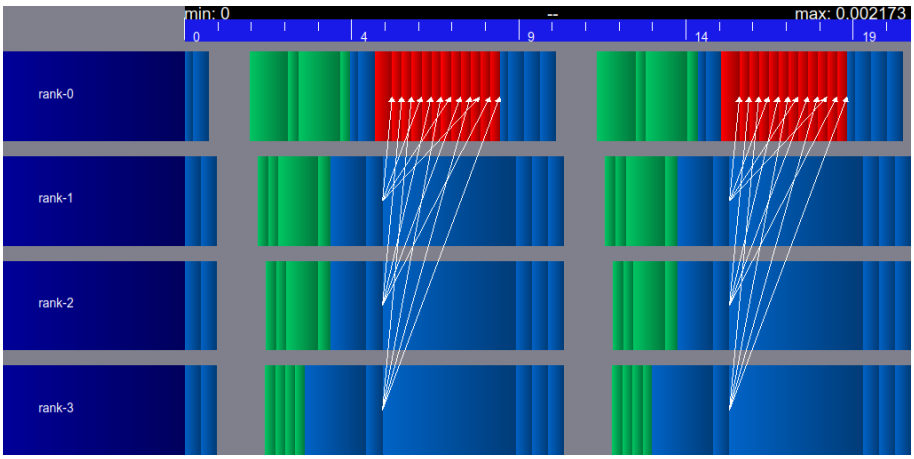

Figure 3: Blocking collective communication for GEMM

`MPI_Bcast` reduces the number of individual messages by leveraging optimized collective communication trees.

1.3 Non-blocking peer-to-peer communications

This approach uses `MPI_Irecv`, `MPI_Issend`, and `MPI_Wait` to overlap communication with computation. Nodes post non-blocking receives and sends in advance (controlled by a lookahead parameter), allowing them to proceed with computation while communication is ongoing.

Using VITE, we can visualize the non-blocking peer-to-peer communication pattern for GEMM in the same configuration as before.

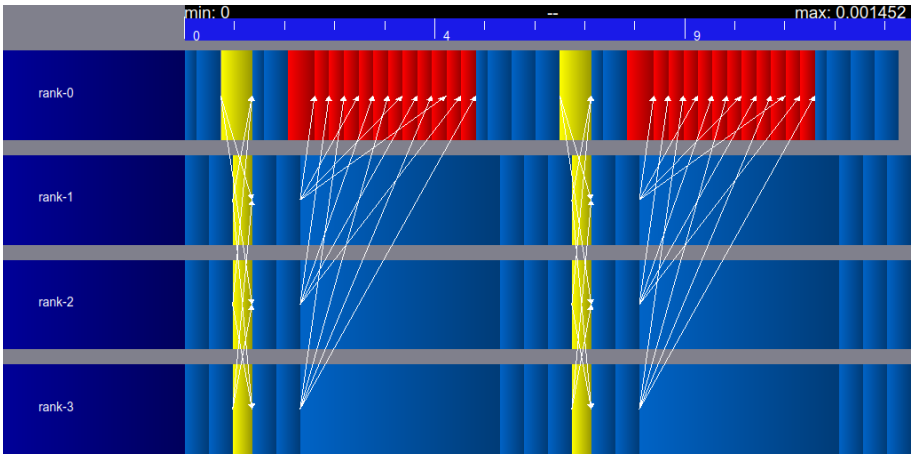

Figure 4: Non-blocking peer-to-peer communication for GEMM

Communication and computation phases overlap significantly, reducing total execution time.

2 Benchmarking

To evaluate the performance of our implementations, we conducted a series of benchmarks. All benchmarks were performed using SimGrid, a simulation framework for MPI programs. Each experiment was repeated 5 times, and performance values were collected from the trace outputs.
For each simulation run, we tried different `lookahead` values for the non-blocking peer-to-peer implementation. The following results show the best performance achieved for each configuration.

2.1 Raw performance

We evaluated the raw performance of each implementation by measuring the number of Gflop/s achieved for different matrix sizes ($m = n = k$) on a fixed grid ($p = q = 2$).

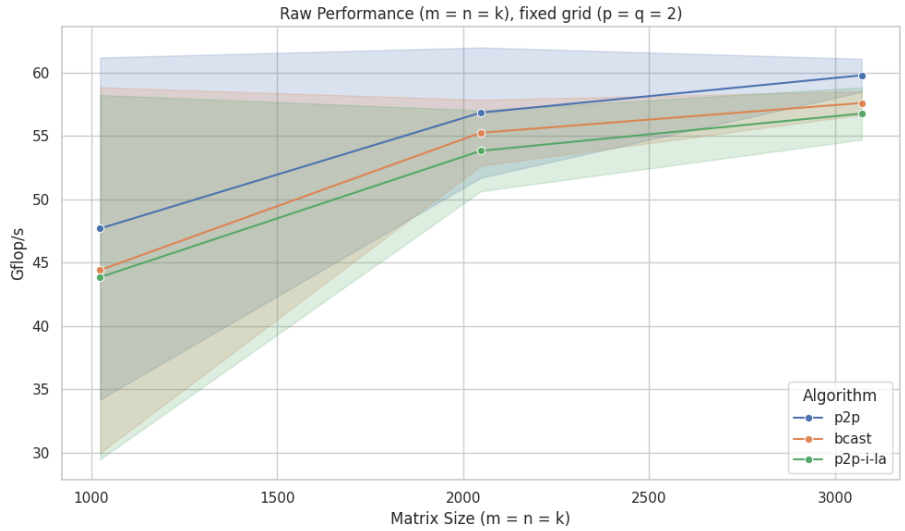

Figure 5: Raw performance of the three implementations for different matrix sizes.

Blocking peer-to-peer (p2p) consistently outperforms both blocking variants. All implementations show increasing performance with larger matrix sizes, indicating better computational-to-communication ratios.

2.2 Strong scaling

We evaluated strong scaling by fixing the problem size ($m = n = k = 2048$) and varying the number of nodes ($p \times q$).

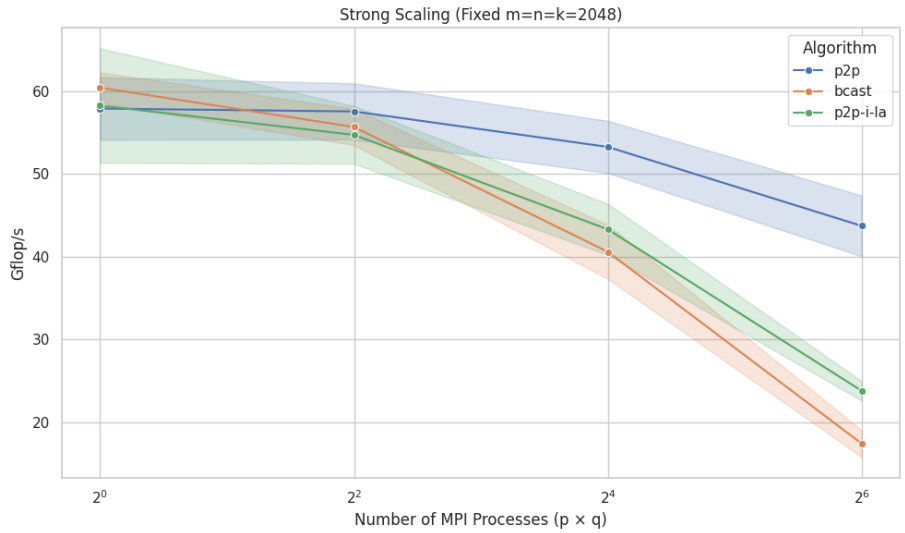

Figure 6: Strong scaling performance of the three implementations for a fixed problem size.

All implementations show declining efficiency with increasing process count, indicating parallel overhead. The fixed problem size means each process has less work, making communication overhead more prominent.

2.3 Weak scaling

We evaluated weak scaling by increasing both the problem size and the number of nodes proportionally.

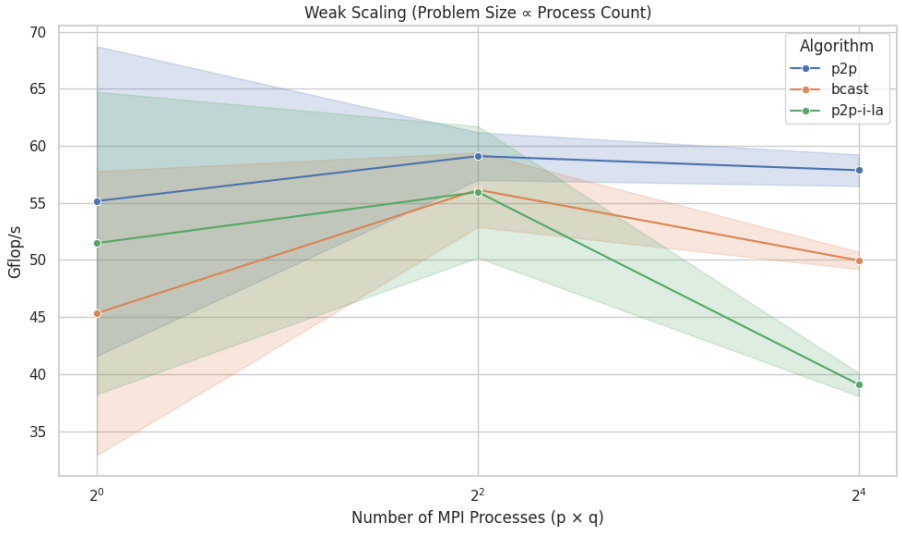

Figure 7: Weak scaling performance of the three implementations for increasing problem size and number of nodes.

Weak scaling should maintain constant performance as both work and resources increase proportionally. However, we observe that only the blocking peer-to-peer (p2p) implementation remains constant, while the other two implementations show decreasing performance.

3 Conclusion

Our benchmarking study of three distributed GEMM implementations reveals several important findings about MPI communication strategies in parallel matrix multiplication. The blocking peer-to-peer implementation consistently demonstrated the best performance across all tested scenarios, achieving a better scaling overall. Surprisingly, the non-blocking peer-to-peer implementation and the collective broadcast approach showed inferior performance compared to the basic blocking implementation, which contradicts theoretical expectations and common parallel computing wisdom that communication-computation overlap should provide performance benefits.