Project Assignment I: All-Pairs Shortest Path (APSP)

Repeated Squaring + Fox (MPI)

Sérgio Cardoso up202107918
Kathleen Soares up201903010

October 30, 2025

Abstract

This report describes the parallel implementation of the *All-Pairs Shortest Path* (APSP) problem through the min-plus product and the *Repeated Squaring* method, using Fox's algorithm over MPI. The program was developed in C, with a distributed approach based on a grid of processes $Q \times Q$, where $P = Q^2$. The main design decisions, communication and data distribution details, as well as preliminary performance measurements are presented.

1 Introduction

The objective of this project is to determine the minimum distances between all pairs of vertices in a weighted directed graph, represented by an adjacency matrix A. Each entry A_{ij} contains the weight of the edge (i, j), or an infinite value when there is no direct connection. The algorithm uses the min-plus product and the *Repeated Squaring* method, with each matrix multiplication implemented in parallel via Fox's algorithm in MPI.

2 Algorithmic Base

2.1 Min-Plus Product

The product min-plus between two matrices A and B is defined by:

$$C_{ij} = \min_k \big(A_{ik} + B_{kj}\big). \tag{1}$$

This operation replaces the traditional sum with the minimum, and the multiplication with the sum. Thus, the distance matrices are multiplied in such a way as to propagate minimum paths.

In the context of this project, this logic is implemented in the function `local_minplus_mm()` of the C code, where each MPI process locally performs the calculation of a block of C from blocks of A and B. This operation constitutes the core of Fox's algorithm, used to perform distributed matrix multiplications in the method of *Repeated Squaring*, ensuring the propagation of the smallest distances between all pairs of vertices.

2.2 Repeated Squaring

The technique of *Repeated Squaring* consists of repeatedly applying the min-plus product of a matrix by itself $(A^2, A^4, A^8, \ldots)$ until the distances do not change or the number of iterations is sufficient $\lceil \log_2 N \rceil$. In the code, this repetition is controlled by a loop in which, at each iteration, the function `fox_minplus()` is called.

2.3 Fox's Algorithm

Fox's algorithm is used to multiply blocks of matrices in parallel. The matrix is divided into sub-blocks $(N/Q) \times (N/Q)$, distributed among the processes arranged in a Cartesian grid. Each process executes, per phase, the diffusion of blocks of A along the row and a circular rotation of the blocks of B along the column, computing the local part of C with the min-plus operation.

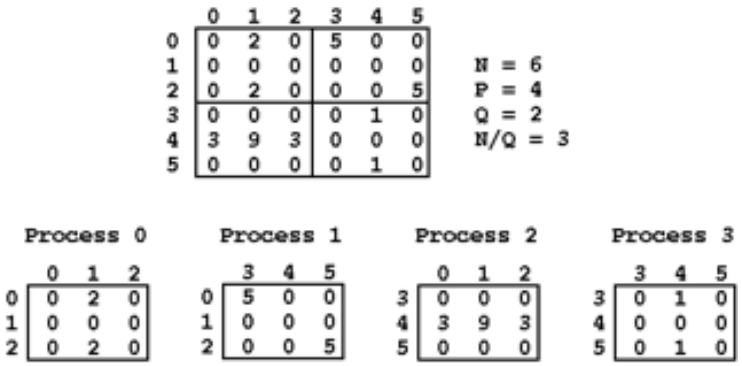

Figure 1: Speedup of the APSP implementation using the Fox algorithm.

3 Implementation Details

3.1 Main Steps

1. **Reading and preparing the data:** Process rank 0 reads the adjacency matrix from standard input. Zeros outside the main diagonal are converted to an infinite value (`INF = 1e9`) to represent the absence of a connection between nodes.

2. **Setting up the process grid:** A two-dimensional Cartesian topology is created, ensuring that the total number of processes p is a perfect square ($p = q^2$). Otherwise, the program terminates with an error message.

3. **Managing padding:** If N is not a multiple of q, the matrix is automatically padded to a size N_{pad} compatible with the algorithm. This behavior can be controlled by an environment variable (`FOX_PADDING`) or a command-line argument (`-padding=on|off|auto`).

4. **Distributing blocks:** The matrix (or its padded version) is divided into blocks and distributed among the processes according to their coordinates (i, j) in the grid.

5. **Executing the Fox algorithm adapted to min-plus:** In each step:
   - a block A is **broadcast** to the corresponding row;
   - each process performs the local min-plus multiplication with block B;
   - block B is **rotated** vertically (sent to the process above).
6. **Repeated squaring:** The algorithm applies the min-plus product repeatedly until $2^k \geq N - 1$, ensuring that all shortest paths are computed.
7. **Collecting and printing the results:** Process (0,0) in the grid gathers all blocks, reconstructs the final matrix, and prints the result, converting infinite values back to zero.

4 Main Implemented Functions

4.1 `grid_setup`

Creates an Cartesian topology of processes and the row and column subcommunicators using `MPI_Cart_create` and `MPI_Cart_sub`, assigning to each process coordinates (my_row, my_col).

4.2 `local_minplus_mm`

Executes the min-plus product between two local blocks A and B, accumulating the result in C:

$$C[i][j] = \min_k (A[i][k] + B[k][j])$$

Unnecessary additions involving infinite values are avoided to reduce computational cost.

4.3 `fox_minplus`

Implements the core of Fox's algorithm. At each iteration:

- the root process of the row broadcasts its block A;
- each process calculates its local contribution via `local_minplus_mm`;
- the blocks B are rotated vertically with `MPI_Sendrecv_replace`.

4.4 `copy_block_out` and `copy_block_in`

Auxiliary functions that copy blocks between the global (padded) matrix and the local submatrices, preserving the original data layout.

5 Error Handling and Robustness

The code handles invalid input situations such as:

- negative values or $N = 0$;
- non-perfect square number of processes;

- incompatible padding configuration (e.g., small N with `-padding=off`).

Each memory allocation is checked and aborts execution with an error message in case of failure.

6 Performance Evaluation

To evaluate performance, the following metrics were measured:

- total execution time excluding read/write operations;
- relative **speedup** compared to executions with 1, 4, 9, 16, and 25 processes.

Table 1: Execution time (ms) and speedup for different process counts.

Processes	**Time (ms)**	**Speedup**
1	x	x
4	x	x
9	x	x
16	x	x
25	x	x

Execution time decreases almost proportionally to the number of processes up to a certain point, with communication cost becoming the main limiting factor for smaller matrices.

7 Main Difficulties and Comments

During the development of this project, the main challenges were related to:

- ensuring compatibility between matrix dimensions and the number of MPI processes ($N \mod Q = 0$);
- debugging the communication pattern of Fox's algorithm (especially the vertical rotation with `MPI_Sendrecv_replace`);
- managing padding logic while keeping the code efficient and readable;
- verifying numerical correctness for small test matrices and avoiding overflow in large-distance values.

Overall, the work was valuable to consolidate concepts of distributed programming with MPI, providing insight into load balancing, scalability, and performance bottlenecks. As a suggestion, including a visualization tool for matrix distribution and communication flow could help future students better understand process decomposition.

8 Conclusions

The parallel implementation of the *All-Pairs Shortest Path* problem using Fox's algorithm adapted for the min-plus product proved to be efficient and scalable for medium and large matrices. The addition of the automatic *padding* system and robust error checking makes the program more versatile and compatible with different cluster configurations. This project provided practical experience in distributed computing concepts, including data decomposition, collective communications, and performance analysis.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Fox, G. C., Otto, S. W., & Hey, A. J. (1987). *Matrix algorithms on a hypercube I: Matrix multiplication*. Parallel Computing, 4(1), 17–31.

[3] Pacheco, P. S. (1998). *A User's Guide to MPI*. San Francisco: Department of Mathematics, University of San Francisco.

Note: The complete source code (`fox.c`) is attached in the ZIP file submitted along with this report.