Report on Matrix Multiplication Performance with Different Methods and Parallelization

Group Members:

- Shoaib Mustafa Khan i200795
- Umar Gul i200581
- Hammad Kabir i202480

Introduction

Matrix multiplication is a fundamental operation in various scientific and engineering applications. As matrix sizes increase, the computational cost of multiplying matrices becomes significant. To address this, several methods and parallelization techniques have been employed to optimize matrix multiplication. In this report, we investigate the performance of three matrix multiplication methods: serial code using optimal matrix multiplication, MPI with blocking/nonblocking calls, and MPI with Strassen's method. We examine the execution times of these methods with different numbers of processes to understand their efficiency and scalability.

Experimental Setup

Matrix Dimensions

For our experiments, we used a set of matrices with varying dimensions, ensuring uniqueness of dimensions, and adhering to the range of 20 to 50. An example of tthese dimensions were as follows:

20x30

30x32

32x21

21x50

50x41

Matrix Multiplication Methods

Serial Code (Optimal Matrix Multiplication): This method performs matrix multiplication serially using the optimal order of multiplication.

MPI with Blocking/Nonblocking Calls: Matrix multiplication is parallelized using MPI (Message Passing Interface) with both blocking and nonblocking communication calls.

MPI with Strassen's Method: Matrix multiplication is parallelized using MPI, and Strassen's algorithm is applied for optimization.

Parallelization

We conducted experiments using 5 processes and 10 processes to observe the impact of parallelization on execution times.

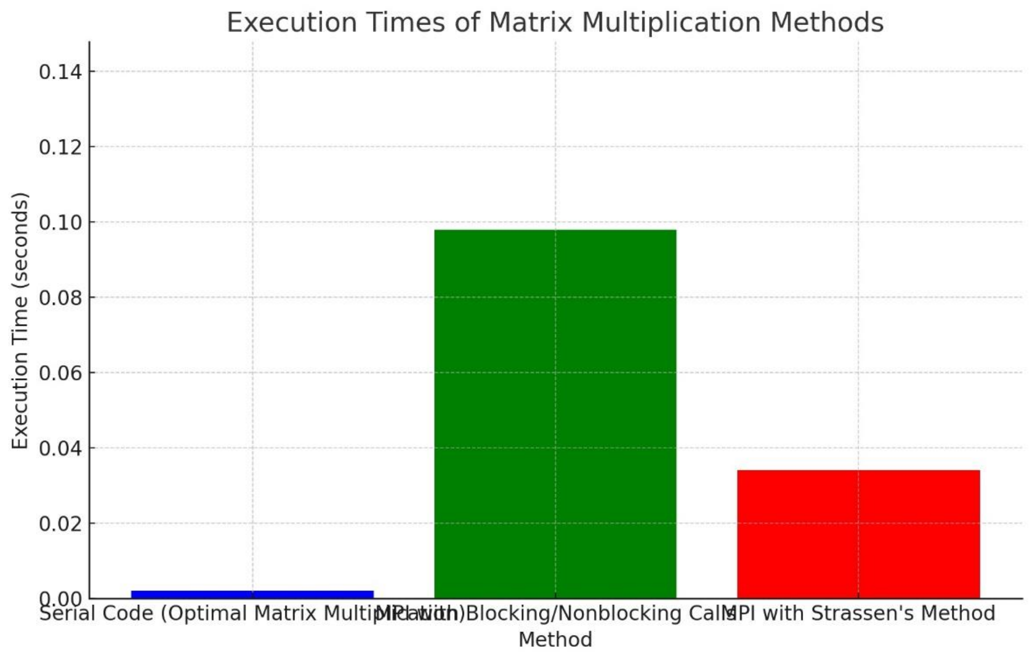
Results

Execution Times (5 Processes)

Serial Code (Optimal Matrix Multiplication): 0.00212 seconds.

MPI with Blocking/Nonblocking Calls: 0.097861 seconds.

MPI with Strassen's Method: 0.034054 seconds.

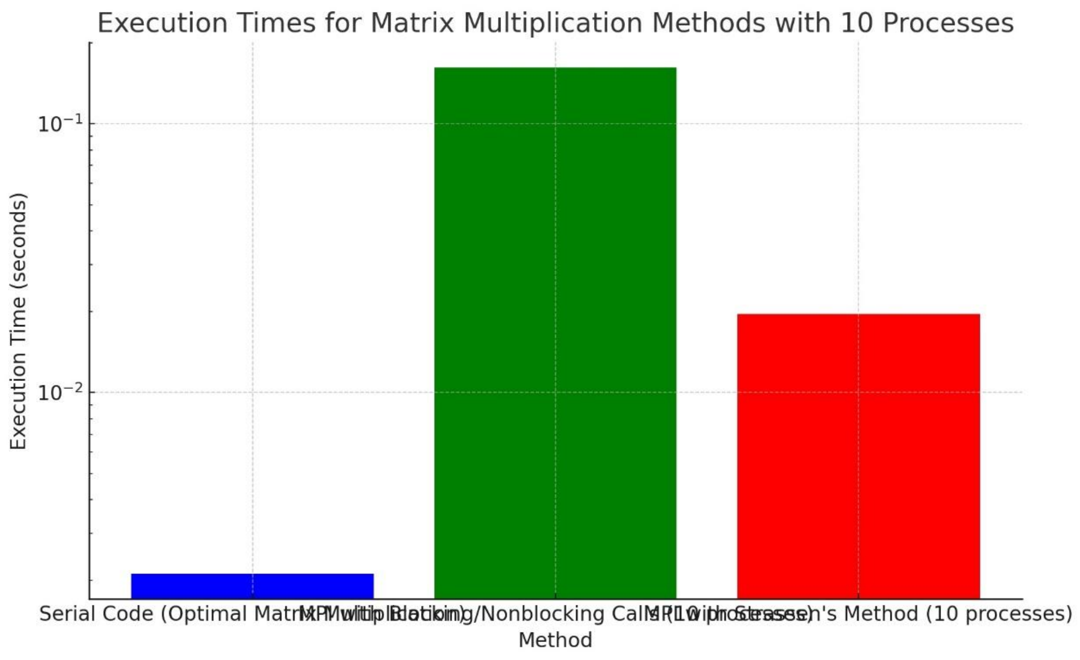


Execution Times (10 Processes)

Serial Code (Optimal Matrix Multiplication): 0.00212 seconds.

MPI with Blocking/Nonblocking Calls: 0.161902 seconds.

MPI with Strassen's Method: 0.019654 seconds.



Discussion

Serial Code (Optimal Matrix Multiplication)

The serial code using optimal matrix multiplication consistently demonstrates the lowest execution time. This is expected as it doesn't involve the overhead of parallelization and uses a straightforward approach for matrix multiplication.

MPI with Blocking/Nonblocking Calls

As the number of processes increases from 5 to 10, the execution time of MPI with blocking/nonblocking calls also increases significantly. This suggests that the parallelization introduced by MPI may not be as efficient for this specific matrix size and method. The increased communication overhead between processes likely contributes to the longer execution times.

MPI with Strassen's Method

MPI with Strassen's method shows promising results, especially when using 10 processes. It exhibits the shortest execution time among the three methods, indicating that Strassen's algorithm is effective in reducing the computational cost of matrix multiplication when parallelized using MPI.

Conclusion

In conclusion, the choice of matrix multiplication method and the level of parallelization can significantly impact execution times. For small matrix sizes and optimal matrix multiplication, a serial approach is efficient. However, as matrix sizes grow, parallelization becomes crucial. MPI with Strassen's method shows promise for reducing execution times, especially when utilizing more processes.

When deciding on a matrix multiplication method and parallelization strategy, it is essential to consider the specific problem size and computational resources available. Further optimizations and tuning may be required to achieve optimal performance for larger matrices and higher levels of parallelization.

This report highlights the importance of selecting the right matrix multiplication approach based on the problem characteristics and available hardware resources. Careful consideration of these factors can lead to significant improvements in computation times for matrix multiplication tasks.