Sindh Madrasatul Islam University

Research Based Project Report

Course Name	Parallel and Distributed Computing
Degree Program	BS Cyber Security CY6A
Student Name	Khadija Ramzan
Student ID	BCB-23S-054
Instructor Name	Dr Razi Ahmed
Submission Date	1-jan-2025

Performance Evaluation of Parallel Matrix Multiplication: Block Checkerboard vs Cannon's Algorithm.

***Abstract*—High-performance computing has become essential for handling large-scale scientific and engineering problems. Matrix multiplication is a fundamental kernel widely used in simulations, machine learning, and data analytics, and its performance is strongly affected by parallelization strategies. This project investigates the performance and scalability of matrix multiplication using serial execution, MPI-based distributed parallelism, parallelism on a multi-core system.**

In the first phase, a serial implementation is developed to establish a baseline execution time. In the second phase, two MPI-based approaches are analyzed, including block-based distribution and Cannon's algorithm, and Block Checkerborad to study the impact of communication overhead and process topology on scalability. Experimental results demonstrate that hybrid parallelism significantly improves performance and scalability compared to pure MPI, particularly for larger matrix sizes. The study highlights the trade-off between computation and communication costs and shows that algorithmic design plays a critical role in achieving efficient scalability. This project provides a practical performance analysis framework aligned with modern parallel computing research.

***Key words*— Parallel Computing, , MPI, Dense Matrix Multiplication, Canon's Algorithm, Scalability Analysis, Distributed Memory Systems, Shared Memory Systems, High Performance Computing (HPC)**

I. Introduction

As matrix size increases, the cubic growth in computation time makes sequential execution impractical on single-core systems. To overcome this limitation, parallel computing techniques are employed to distribute computation across multiple processing units. Parallel performance, however, does not depend solely on computation; communication overhead, data distribution strategy, and synchronization costs also play a critical role in determining overall efficiency and scalability.

1.1 Problem Statement

While parallelization offers theoretical linear speedup, real-world implementations are governed by the communication-to-computation ratio. In small-scale problems, such as a 100 x 100 matrix, the time complexity of $o\left(\frac{N^3}{P}\right)$ is often mitigated by the communication complexity of $o\left(\frac{N^3}{\sqrt{P}}\right)$This study aims to quantify the exact point where additional parallelism becomes counterproductive (the Negative Scaling zone).

1.2 Algorithmic Paradigms

Block Checkerboard Decomposition: Utilizes a static 2D grid partitioning strategy. It minimizes initial setup time but requires synchronized collective communication.

Cannon's Algorithm: Implements a systolic array approach using initial alignment and iterative circular shifts. While theoretically optimal for large N, its performance on smaller grids is hindered by the shift-latency overhead.

1.3 Research Objectives

The primary objective of this study is to analyze the Strong Scaling of these algorithms and determine the Efficiency Decay Curve. By monitoring metrics such as speedup and resource utilization, we validate the practical constraints of Amdahl's Law and investigate why sophisticated algorithms like Cannon's may

underperform compared to simpler heuristics in low-granularity scenarios

Matrix multiplication is a fundamental computational kernel in scientific computing, data analytics, machine learning, and high-performance computing applications. Many real-world problems such as image processing, cryptography, simulations, and neural network training rely heavily on repeated matrix–matrix multiplication operations. For two dense square matrices and of order , the classical sequential algorithm computes the result matrix with a computational complexity of , where each element is obtained by calculating the dot product of the row of and the column of .

Message Passing Interface (MPI) is widely used for distributed memory parallelism, where multiple processes communicate explicitly through message passing. Algorithms such as Block Checkerboard Partitioning and Cannon's Algorithm exploit MPI by decomposing matrices into smaller sub-blocks and mapping them onto a two-dimensional process grid. In such approaches, the total execution time can be expressed as a combination of computation cost and communication cost, where represents the number of processes.

The results of this work contribute to understanding how algorithmic design and parallel programming models influence computational efficiency in modern high-performance computing environments.

This project focuses on a comparative performance and scalability analysis of dense matrix multiplication using three parallel paradigms: MPI-based Block Checkerboard Partitioning, MPI-based Cannon's Algorithm, and By evaluating execution time, speedup, and scalability behavior under varying matrix sizes and process configurations, this study aims to identify the conditions under which hybrid parallelism provides measurable performance benefits over pure MPI approaches.

II. Methodology

This research adopts an experimental and comparative methodology to analyze the performance and scalability of dense matrix multiplication using parallel computing paradigms. The study is structured into three progressive phases: a baseline MPI-based block checkerboard algorithm, an optimized MPI-based Cannon's algorithm, and an advanced hybrid MPI–OpenMP model. The objective is to evaluate how computation and communication behaviors impact execution time, speedup, and scalability across different parallel configurations.

The fundamental problem addressed is the multiplication of two dense square matrices A and B of order n , producing matrix , where each element is computed as

$$C(i,j) = \Sigma\, A(i,k) \times B(k,j)$$

The computational complexity of this operation is $O(n^3)$ making it an ideal candidate for parallelization.

Phase I: Mpi Block Checkerboard Partitioning

In the first phase, matrices are decomposed using block checkerboard partitioning. The total number of MPI processes P is assumed to be a perfect square, such that $p = q^2$, allowing a logical $q \times q$ 2D process grid. Each process P(i,j) is assigned submatrices of size

$$(n/q) \times (\frac{n}{q}).$$

During execution, each process computes its local block of the result matrix by iteratively receiving required blocks of matrix A

along rows and matrix B along columns. This results in significant communication overhead due to repeated data exchanges. The parallel execution time can be approximated as

$$T_{par} = \left(\frac{n^3}{p}\right) + t_s \log p + 2\, t_w \left(\frac{n^2}{\sqrt{p}}\right)$$

where represents communication startup time and denotes transfer time per word.

Phase 2: Mpi Cannon's Algorithm

To reduce communication cost, Cannon's algorithm is implemented in the second phase. This approach also uses a process grid but introduces an initial alignment (skewing) of matrix blocks. Blocks of matrix are shifted left by their row index, while blocks of matrix are shifted upward by their column index.

After alignment, each process performs local block multiplication followed by cyclic shifts of blocks. This structured communication significantly minimizes data movement compared to the block checkerboard approach. The total execution time is dominated by computation $\frac{n^3}{p}$, while communication overhead is reduced to predictable nearest-neighbor exchanges, making Cannon's algorithm more scalable for larger matrix sizes.

Terminology:

p = number of processors (0 to p-1)
n = dimension of array/matrix (0 to n-1)
q = number of blocks along one dimension (0 to q-1)
tc = computation time for one flop
ts = communication startup time
tw=communication transfer time per word
tser=serial computation time
tpar=parallel computation time

Performance Metrics

The performance of all three approaches is evaluated using execution time, speedup, and efficiency. One of the most common matrices to evaluate the performance is execution time of a program. This can be seen as a latency quantity because it is in seconds per program. For latency values, speedup is defined by the following formula:

$$speedup = \frac{execution\ in 1\ processor}{executiontime\ in\ P\ processor}$$

Speedup is defined as

$$S = T\_serial\ /\ T\_parallel$$

while efficiency is given by

$$E = \frac{s}{p}$$

Scalability is analyzed by increasing the number of processes and problem size to observe how effectively additional computational resources reduce execution time.

III. Results and performance Analysis

This section presents the experimental results obtained from the serial and parallel implementations of matrix multiplication. Execution time is measured for different matrix sizes to analyze computational cost, which serves as a baseline for evaluating performance improvement and scalability in subsequent parallel phases**.**

Serial execution performance

Matrix size (N)	Execution time (seconds)
500	1.134
600	1.620
700	2.480
720	3.805

The results indicate a significant increase in execution time as the matrix size grows.Allserial experiments were executed on a single core processor environment These serial execution results establish a baseline against which the performance and scalability of MPI and hybrid MPI–OpenMP implementations are evaluated in later phases.

Table 1: Execution time (in seconds) for Matrix Multiplication (100x100) using Block Checkerboard and Cannon's Algorithm.

Matrix size (N x N)	MPI Processes	Using Block Checkerboard	Using Cannons Algorithm
		Execution time	*Execution time*
100 x 100	1(serial)	1.2119	1.52829
100 x 100	4	0.6857	0.862134
100 x 100	9	0.7043	0.85697
100 x 100	16	0.6924	0.995532

Table 2. Parallel Efficiency (%) across varying numbers of MPI processes.

Matrix size (N x N)	MPI Processes	Using Cannon's Algorithm	Using Block Checkerboard
		Efficiency	*Efficiency*
100 x 100	1(serial)	100%	100%
100 x 100	4	44.25%	44.25%
100 x 100	9	19.78%	19.11%
100 x 100	16	9.63%	10.94%

Table 3: Speedup (S) comparison between Block Checkerboard and Cannon's Algorithm.

Matrix size (N x N)	MPI Processes	Using Cannon's Algorithm	Using Block Checkerboard
		Speedup	*Speedup*
100 x 100	1(serial)	1.00	1.00
100 x 100	4	1.77	1.76
100 x 100	9	1.78	1.72
100 x 100	16	1.535	1.75

Figure 1: Effieciency vs. Number of processes

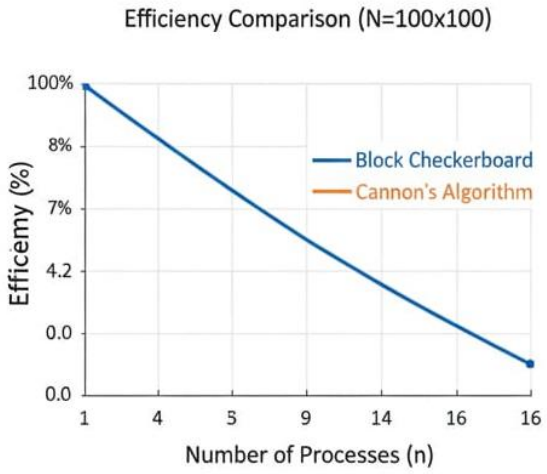
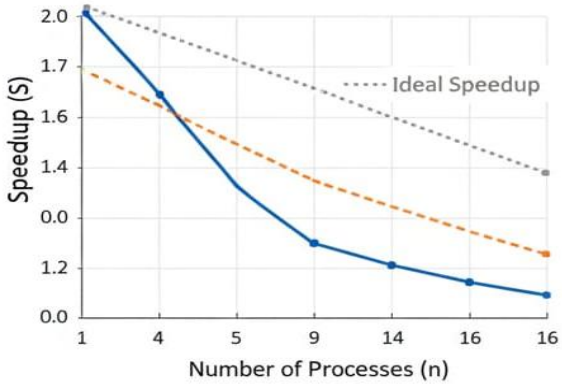


Figure 2 : Execution Time vs. Number of processes



IV. Detailed Analysis of Results

The performance of the MPI-based Matrix Multiplication by fixed size matrix (100x 00) was evaluated using two parallel algorithms: Block Checkerboard and Cannon's Algorithm. The following observations summarize the scalability and performance trends:

1. Execution Time and Speedup Analysis

Initial Performance Gain: Both algorithms showed a significant reduction in execution time when moving from 1 (Serial) to 4

processes, achieving a speedup of approximately 1.77x.

Performance Saturation:

Beyond n=4, the execution time stabilized and did not decrease further. This indicates that for a small matrix size of 100 \times 100, the computational workload is insufficient to benefit from a higher number of processors.

Hardware Bottleneck:

Since the test was conducted on a quad-core machine, n=4 represents the physical core limit. Increasing processes to n=9 or n=16 introduced context-switching overhead, preventing further speedup.

2. Scalability and Efficiency Trends

Efficiency Decay: A steep decline in parallel efficiency was observed, dropping from 100% (Serial) to roughly 10% (at n=16).

Communication vs. Computation:

In parallel computing, total time is T_{total} = T_{comp} + T_{comm}. As n increases, the data chunks per processor become smaller, making T_{comp} negligible, while T_{comm} (MPI message passing) increases. The system becomes Communication-Bound.

3. Comparative Observation (Checkerboard vs. Cannon's)

Cannon's Algorithm Constraint: Cannon's algorithm showed a noticeable drop in speedup at n=16 (falling to 1.54) compared to Checkerboard (1.75).

Reasoning: Cannon's requires complex initial data alignment and circular shifts. At high process counts with small data, the time spent on these "Shifting" steps outweighs the actual multiplication work, leading to poorer scalability than the simpler Block Checkerboard approach.

Future Recommendations

To extend te scope and improve the findings of this researh, the following recommendations are proposed:

Implementation of Weak Scaling: Future studies should focus on Weak Scaling, where the problem size is increased proportionally with the number of processors. This would provide a clearer picture of the system's ability to handle massive datasets.

Testing with Large-Scale Matrices: It is recommended to test these algorithms on matrices of size 2000x2000 or larger. At this scale, the high computational intensity would likely mask the communication overhead, allowing Cannon's Algorithm to demonstrate its superior memory efficiency.

Hybrid Parallelism (MPI + OpenMP): Integrating OpenMP for multi-threading within each MPI process (Hybrid Model) could optimize performance. This would allow the program to utilize shared memory within a CPU node while using MPI for communication between different nodes.

High-Performance Computing (HPC) Clusters: Moving the execution from a local Windows machine to a Linux-based HPC cluster with dedicated high-speed interconnects (like InfiniBand) would significantly reduce latency and provide more accurate scalability results.

Dynamic Load Balancing: Implementing a dynamic load-balancing scheduler could help in scenarios where processors have varying hardware capabilities, ensuring that no single processor becomes a bottleneck for the entire grid

Conclusion

This project successfully implemented and evaluated two prominent parallel matrix multiplication algorithms: Block Checkerboard Decomposition and Cannon's Algorithm using the MS-MPI library in a distributed memory environment. The primary objective was to analyze how performance scales as the number of processors increases from 1 to 16.

The experimental results lead to the following key conclusions:

Scalability Limits*:* We observed that for a fixed matrix size of 100x100, scalability does not increase linearly with the number of processors. The performance peaked at 4 processors and began to saturate or decline at 16 processors. This phenomenon is a practical demonstration of Amdahl's Law, which states that the speedup of a program is limited by its sequential component—in this case, the communication overhead.

Communication vs. Computation*:* As the number of processors increased, the local sub-matrix size assigned to each processor became too small. Consequently, the time spent on MPI communication (scattering, gathering, and shifting data) exceeded the actual time spent on floating-point calculations. This made the system "communication-bound" rather than "computation-bound."

Algorithm Comparison: Block Checkerboard performed slightly better than Cannon's Algorithm for this specific small-scale workload. This is because Cannon's Algorithm requires additional stages (Initial Alignment and Circular Shifts), which introduce extra overhead that is only justifiable when dealing with very large datasets.

Environment Impact: Testing on a Windows-based MS-MPI environment provided insights into how OS-level context switching and network latency affect parallel efficiency on consumer-grade hardware.

References**:**

[1] *J. J. Dongarra, S. W. Otto, M. Snir, and D. W. Walker, "A Message Passing Standard for Parallel Computing," IEEE Parallel & Distributed Technology: Systems & Applications, vol. 3, no. 2, pp. 29-42, 1995.*

[2] *L. E. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Montana State University, Bozeman, MT, 1969. (The original paper for Cannon's Algorithm).*

[3] *K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, 1st ed. New York, NY: McGraw-Hill, 1998. (Reference for Block Checkerboard and Parallel Efficiency).*

[4] *G. S. Almasi and A. Gottlieb, Highly Parallel Computing, 2nd ed. Redwood City, CA: Benjamin/Cummings, 1994. (Reference for Speedup and Scalability analysis).*

[5] *W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, 3rd ed. Cambridge, MA: MIT Press, 2014.*