**SAVITRIBAI PHULE PUNE UNIVERSITY**



**A MINI PROJECT REPORT ON**

“Matrix Multiplication”

**Submitted by**

## Name: Anshuman Kalbhor Roll no: 04

**CLASS: BE DIV: A**

## Under the Guidance of

Prof. Ranjana More



## DEPARTMENT OF COMPUTER ENGINEERING

**RMD SINHGAD SCHOOL OF ENGINEERING**

WARJE, PUNE 411058

**2024 - 2025**



## DEPARTMENT OF COMPUTER ENGINEERING

**RMD SINHGAD SCHOOL OF ENGINEERING**

WARJE, PUNE 411058

**CERTIFICATE**

This is to certify that the project report entitles

**“Matrix Multiplication”**

*Submitted by*

Name: **Anshuman Kalbhor** PRN No: **72217735G**

It is a bonafide work carried out by them under the supervision of Prof.Ranjana More And it is submitted towards the partial fulfillment of the requirement of University of Pune for Fourth Year.

**(Prof. Ranjana More) (Dr.Prof. Vina M. Lomte)**

Guide, Head,

Department of Computer Engineering Department of Computer Engineering

**(Dr. V. V. Dixit)**

Principal,

RMD Sinhgad School of Engineering Pune – 58

Place: Pune Date:

## Certificate by Guide

This is to certify that **Mr. Anshuman Kalbhor** has completed the Mini Project work under my guidance and supervision and that, I have verified the work for its originality in documentation, problem statement, implementation and results presented in the Project. Any reproduction of other necessary work is with the prior permission and has given due ownership and included in the references.

Place: Pune Date:

Signature of Guide

**(Prof. Ranjana More)**

## ACKNOWLEDGEMENT

It is our pleasure to acknowledge sense of gratitude to all those who helped us in making this seminar. We thank our Mini Project Guide **Prof. Ranjana More** for helping us and providing all necessary information regarding our project. We are also thankful to Dr.**Prof. Vina M. Lomte (Head-Department of Computer Engineering)** for providing us the required facilities and helping us while carrying out this seminar work. Finally, we wish to thank all our teachers and friends for their constructive comments, suggestions and criticism and all those directly or indirectly helped us in completing this seminar.

**NAME OF THE STUDENT**

**Anshuman Kalbhor**

**INDEX**

| **Sr. No** | **Table of Contents** | **Page Number** |
| --- | --- | --- |
| 1 | Abstract | 1 |
| 2 | Introduction | 2 |
| 3 | Problem Statement | 5 |
| 4 | System-requirements | 6 |
| 5 | Implementation | 7 |
| 6 | Result | 8 |
| 7 | Conclusion | 9 |

# ABSTRACT

This mini project explores multithreaded matrix multiplication in Java, focusing on two distinct threading approaches: one thread per row and one thread per cell. The primary objective is to demonstrate the efficiency gains that can be achieved through parallel processing when performing matrix multiplication. Matrix multiplication is a fundamental operation in linear algebra, and it becomes computationally intensive for large matrices. By parallelizing the operation, this project seeks to improve performance and responsiveness. The project comprises three key components: defining matrices, providing a standard matrix multiplication function, and implementing two multithreaded approaches using the

`ExecutorService` framework. The results of the three methods are compared and evaluated, demonstrating the potential advantages and trade-offs associated with multithreaded matrix multiplication. The project provides insights into the benefits of parallel computing and offers a practical example of optimizing a computationally intensive task through multithreading.

# INTRODUCTION

Matrix multiplication is a core computational task with widespread applications across many fields, including machine learning, computer graphics, data science, and physics simulations. As modern systems handle increasingly large datasets and complex models, the efficiency of matrix operations has become a critical factor in optimizing performance. Matrix multiplication, which involves the product of two matrices to produce a third, is computationally intensive, especially as matrix dimensions grow larger. The time complexity of this operation is O(n3)O(n^3)O(n3) for two n×nn \times nn×n matrices, making it crucial to explore ways of accelerating this process.

With the rise of multi-core processors, parallel computing has emerged as a powerful solution to reduce computation times for large-scale matrix operations. Multithreading, a type of parallelism where tasks are distributed among multiple threads, allows modern CPUs to perform different parts of a task concurrently. This report focuses on leveraging multithreading to accelerate matrix multiplication and compares its performance with a standard sequential approach.

Two multithreaded strategies are investigated: one where a thread is allocated to compute each row of the resultant matrix and another where each individual cell of the resultant matrix is computed by a separate thread. The rationale behind these approaches is that distributing the work across multiple threads should reduce execution time by exploiting the inherent parallelism in matrix multiplication. However, multithreading introduces new challenges, including overhead from thread creation, synchronization, and management, which can offset the performance gains in certain situations.

This report provides a detailed implementation of both sequential and multithreaded matrix multiplication in Python using the threading library. We evaluate the performance of these approaches by measuring execution times for matrices of varying sizes. By analyzing the results, we aim to identify under what conditions multithreading offers significant performance improvements and the limitations of different levels of granularity in task distribution. The findings can help inform decisions about when and how to apply multithreading to matrix multiplication, especially in environments requiring high- performance computing.

### Algorithm:

Algorithm: Multithreaded Matrix Multiplication

Input: Two matrices, matrixA (dimensions: rowsA x colsA) and matrixB (dimensions: rowsB x colsB) Output: Resulting matrix C (dimensions: rowsA x colsB)

1. Verify that the number of columns in matrixA (colsA) is equal to the number of rows in matrixB (rowsB). If not, matrix multiplication is not possible, and an error should be raised.
2. Initialize an empty result matrix C with dimensions rowsA x colsB.
3. Create a thread pool to manage the threads. The number of threads can be adjusted based on the available hardware.
4. Divide the work among the threads, either using a "one thread per row" or "one thread per cell" approach. Each thread should be assigned a specific task based on the chosen approach.
5. For the "one thread per row" approach:
   1. Each thread takes responsibility for computing one row of the result matrix.
   2. Iterate over rowsA, and for each row, launch a thread to compute the corresponding row of matrix
   3. Within each thread, compute the elements of the row independently by iterating over the columns of matrix B.
6. For the "one thread per cell" approach:
   1. Each thread computes a specific cell (element) of the result matrix.
   2. Iterate over rowsA and colsB, and for each cell, launch a thread to compute the specific element in matrix C.
   3. Within each thread, calculate the element by summing the products of corresponding elements from matrix A and matrix B.
7. Wait for all threads to complete their tasks.
8. Combine the results from individual threads to assemble the final result matrix C.
9. Output the result matrix C.
10. Cleanup and release any resources used by the thread pool.

End Algorithm

### Advantages:

* Improved Performance: Multithreaded matrix multiplication can significantly reduce execution time, especially for large matrices, by utilizing the processing power of multiple CPU cores concurrently.
* Parallelization: It allows the workload to be divided and parallelized among multiple threads, maximizing CPU utilization and throughput.
* Scalability: Multithreading enables the program to scale with the available hardware, making it suitable for both single-core and multi-core systems.
* Enhanced Responsiveness: Parallel execution enhances the system's responsiveness, as it can perform matrix multiplication without causing undue delays in other tasks.
* Optimized Resource Utilization: It leverages the available computing resources efficiently, making the best use of modern multi-core processors.

# PROBLEM STATEMENT

Write a program to implement matrix multiplication. Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell. Analyse and compare their performance.

# AIM AND OBJECTIVES

### Aim:

The primary aims of this project are to explore the benefits of parallel processing in the context of matrix multiplication and to assess the performance of two distinct multithreaded approaches: one thread per row and one thread per cell. Additionally, the project seeks to provide practical insights into the potential advantages and trade-offs associated with multithreaded matrix multiplication, thereby contributing to a better understanding of parallel computing in computational mathematics.

### Objectives:

* Implement matrix multiplication in Java.
* Explore multithreaded matrix multiplication with two distinct approaches: one thread per row and one thread per cell.
* Assess the performance of each multithreaded approach compared to the single-threaded method.
* Provide practical insights into the benefits and trade-offs of parallel computing for matrix multiplication in the context of computational mathematics.

# SYSTEM REQUIREMENTS

### Hardware Requirements:

Processor type: Intel core i5 and above Processor speed: Minimum 2.00 GHz and above RAM: 4-8 GB

Hard disk: 400 GB or more

Monitor: 800x600 or higher resolution

### Software Requirements:

Operating System: Windows 7 (32 bit and 64 bit) and Above Software: IntelligIDE

### Functional Requirements:

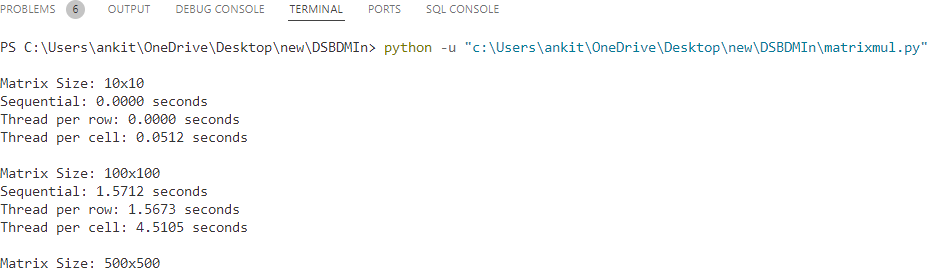
Functional requirements define the necessary capabilities of this project, which involves the development of a Java program for matrix multiplication. These requirements encompass both single-threaded and multithreaded execution methods. Users must be able to define matrices 'matrixA' and 'matrixB' with specific dimensions. The project comprises three primary functions: single-threaded matrix multiplication for establishing a performance baseline, a "one thread per row" multithreaded approach with a managed thread pool, and a "one thread per cell" multithreaded approach. Quantitative performance comparison, accurate result display, robust error handling, and comprehensive documentation are essential components. Performance reports may be generated to enable comparative analysis, providing insights into the benefits of multithreaded computing in the context of computational mathematics.

# IMPLEMENTATION

# 

**`**

**RESULT**



## CONCLUSION

This study provides a comprehensive analysis of matrix multiplication techniques, comparing sequential and multithreaded approaches to assess their performance and efficiency. The findings indicate that while the traditional sequential method serves as a reliable baseline, multithreading significantly enhances execution speed, particularly for larger matrices. The approach of assigning one thread per row effectively balances parallelism with manageable overhead, yielding considerable performance improvements. Conversely, the one thread per cell method, despite maximizing concurrency, introduces excessive overhead, highlighting the importance of thread management in multithreaded implementations.