

***PDC Project***

***Group Members: Asad Ullah Khan (K21-4945)***

***Rohail Rashid Khan (K21-3152)***

***Agha Fardeen (K21-3371)***

***Topic: Matrix Operations***

***Teacher: Sir Nadeem Kafi***

***Introduction***

In this Parallel and Distributed Computing project, we leverage OpenMP and MPI to enhance the efficiency of matrix operations. Matrices play a crucial role in various applications, but as their size grows, so does computational complexity. Traditional methods struggle, prompting the use of parallel computing. OpenMP handles shared-memory parallelism, while MPI excels in distributed-memory systems. Our goal is to create a powerful framework for matrix operations, from basic multiplication to complex tasks, optimizing performance on both shared and distributed architectures. Join us in this journey to parallelize operations, optimize resources, and witness a substantial speed boost, gaining essential skills in the realm of Parallel and Distributed Computing. Let the parallelization adventure begin!

***Matrix Operations***

**Matrix Multiplication:** Matrix multiplication, a fundamental operation in scientific computing, becomes a formidable challenge with increasingly large matrices. To tackle this, we employ a hybrid approach, harnessing the strengths of both OpenMP and MPI.OpenMP shines in shared-memory parallelism, allowing us to distribute the workload across multiple threads within a single node. For matrix multiplication, this means concurrent processing of rows and columns, optimizing performance on multicore systems.MPI, designed for distributed-memory environments, takes the collaboration to the next level. It enables seamless communication between nodes in a cluster, essential for handling large matrices that exceed the capacity of a single machine.The synergy between OpenMP and MPI in matrix multiplication yields a scalable solution, breaking down the task into manageable chunks that are processed concurrently within nodes and coordinated across the entire cluster. This approach not only accelerates computation but also ensures efficient utilization of resources, making it a powerful strategy for tackling large-scale matrix operations in the realm of parallel and distributed computing.

**Matrix Addition and Subtraction:** Matrix addition and subtraction, fundamental operations in linear algebra, are expedited through the parallel prowess of OpenMP. In the realm of shared-memory parallelism, OpenMP shines by concurrently handling the element-wise addition and subtraction of matrices on multiple threads within a single node.By distributing the workload across cores, OpenMP ensures efficient utilization of multicore architectures, leading to a significant acceleration in the computation of matrix addition and subtraction. This approach is particularly advantageous when dealing with large matrices, where the traditional sequential methods may fall short in terms of speed and efficiency.The simplicity of matrix addition and subtraction belies the complexity that arises with sizable matrices. OpenMP's ability to parallelize these operations provides a valuable tool for scientists, engineers, and researchers dealing with data-intensive applications. This parallel approach not only expedites computation but also unlocks the potential for enhanced scalability, making it an invaluable asset in the arsenal of parallel computing techniques.

**Matrix Inversion and Transposition:** Matrix inversion and transposition, critical operations in linear algebra, can be expedited through the parallel capabilities of OpenMP. In the realm of shared-memory parallelism, OpenMP excels by concurrently handling the intricate computations involved in these operations on multiple threads within a single node.Matrix inversion, a computationally intensive task, benefits from parallel processing as OpenMP divides the workload across cores, accelerating the inversion process. This is particularly valuable in applications where rapid and parallelized computation of the inverse matrix is essential.Similarly, matrix transposition, which involves swapping rows and columns, becomes more efficient with OpenMP's parallel approach. The concurrent processing of elements across threads ensures a speedy transformation, making it well-suited for handling large matrices.OpenMP's simplicity and efficiency in parallelizing matrix inversion and transposition operations provide a valuable tool for researchers and practitioners in fields such as data analysis, numerical simulations, and scientific computing. This parallel approach not only accelerates computation but also enhances scalability, making it a powerful asset in the toolkit of parallel computing techniques.

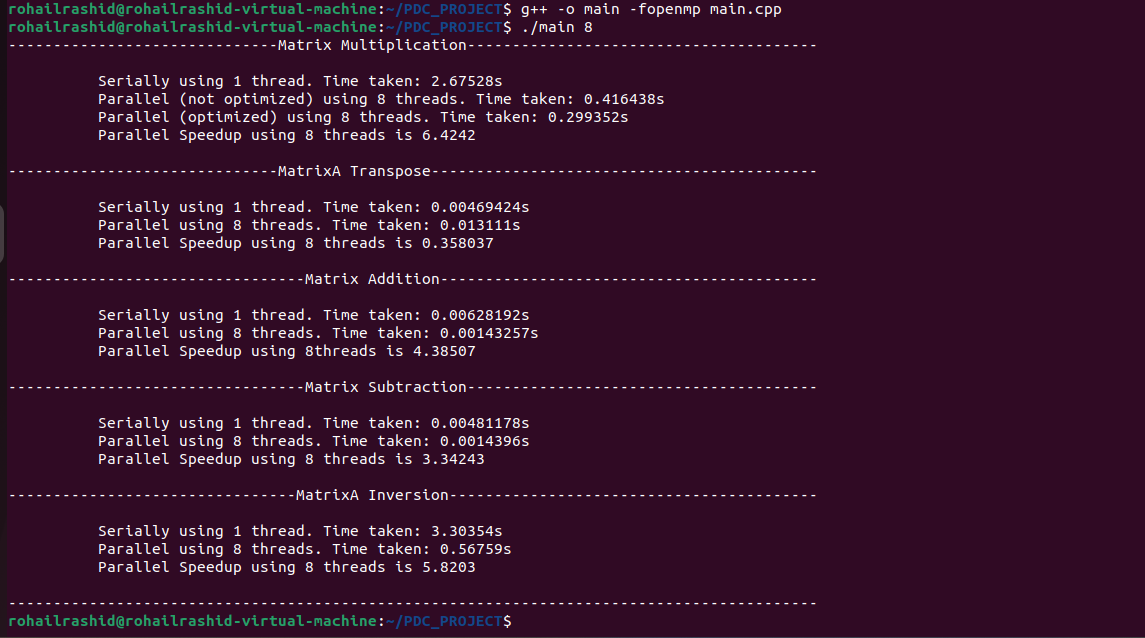
**Computations at Matrix Multiplication:**

We created a complex transformation function that transforms our matrix using a complex formula involving numerous mathematical functions so our matrix multiplication becomes compute intensive and works for a larger no of time in serial about 40 mins. This is necessary as computational problems of such large-scale benefits from parallel computing. Well for the dataset we used maximum 5000x5000 two matrices that equates to about 25 million data values in a single matrix.

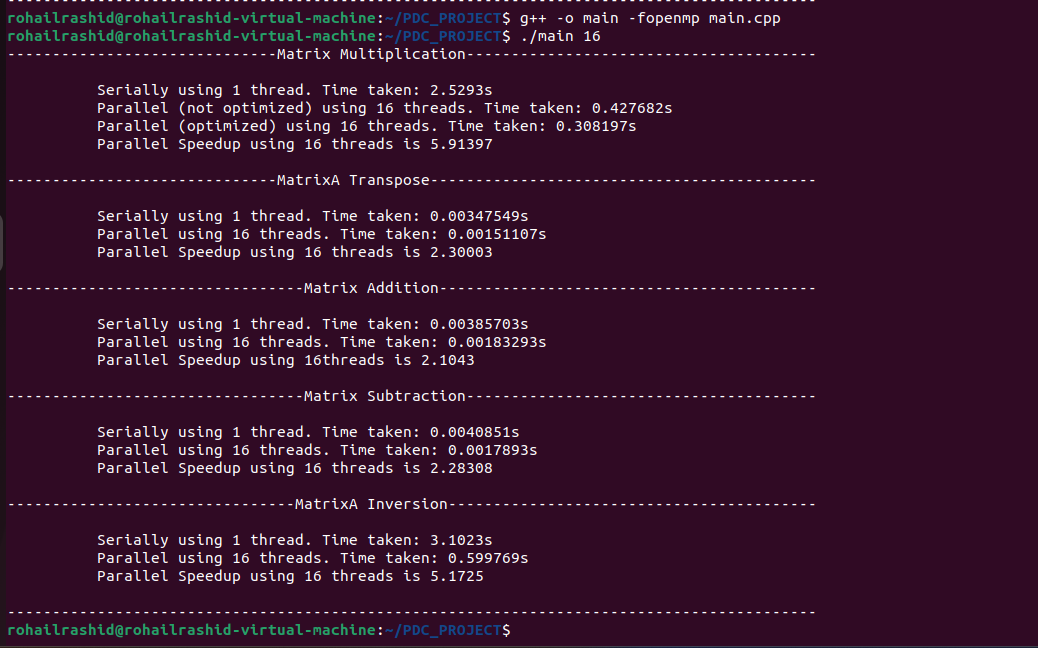
# 

# **Matrix Operations using Openmp and Size 500**

***Size 500 threads 8***

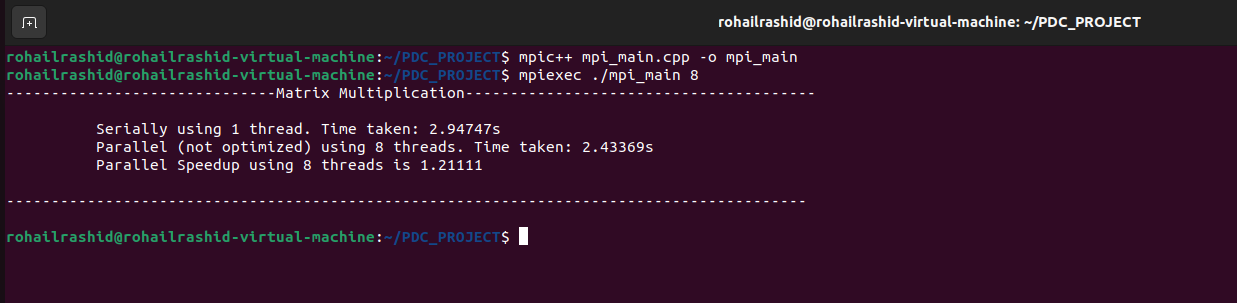
****

***Size 500 threads 16***

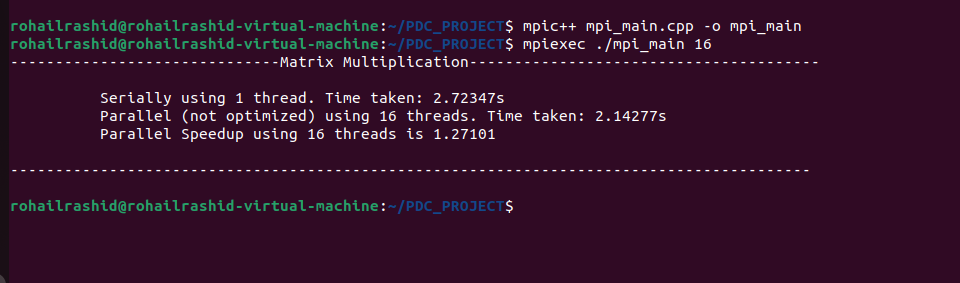
****

**Matrix Operations using MPI and Size 500**

***Size 500 processes 8***

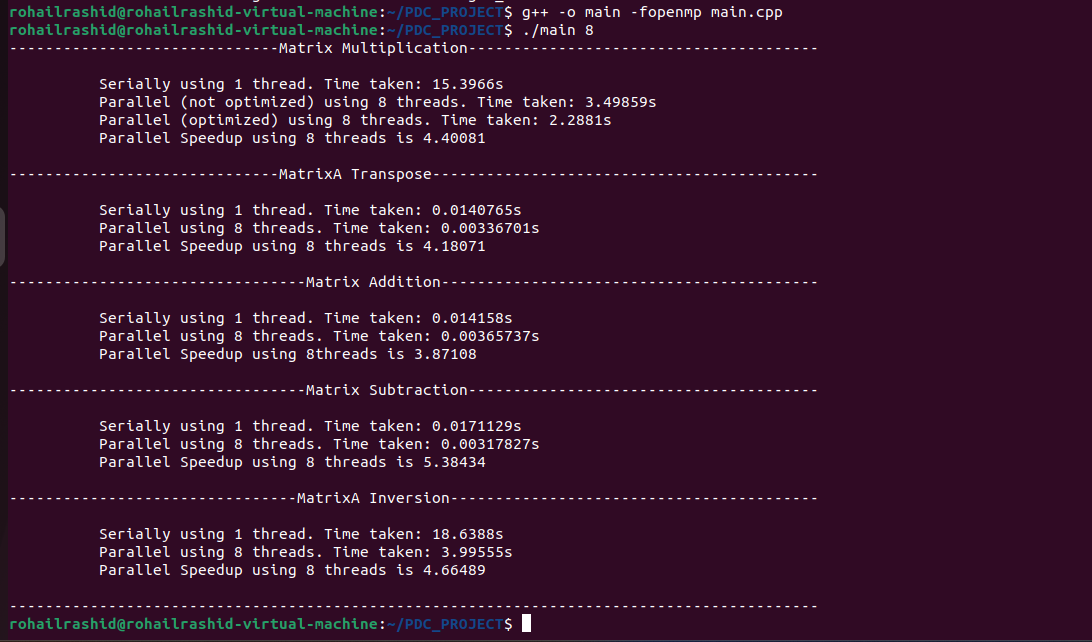
****

***Size 500 processes 16***

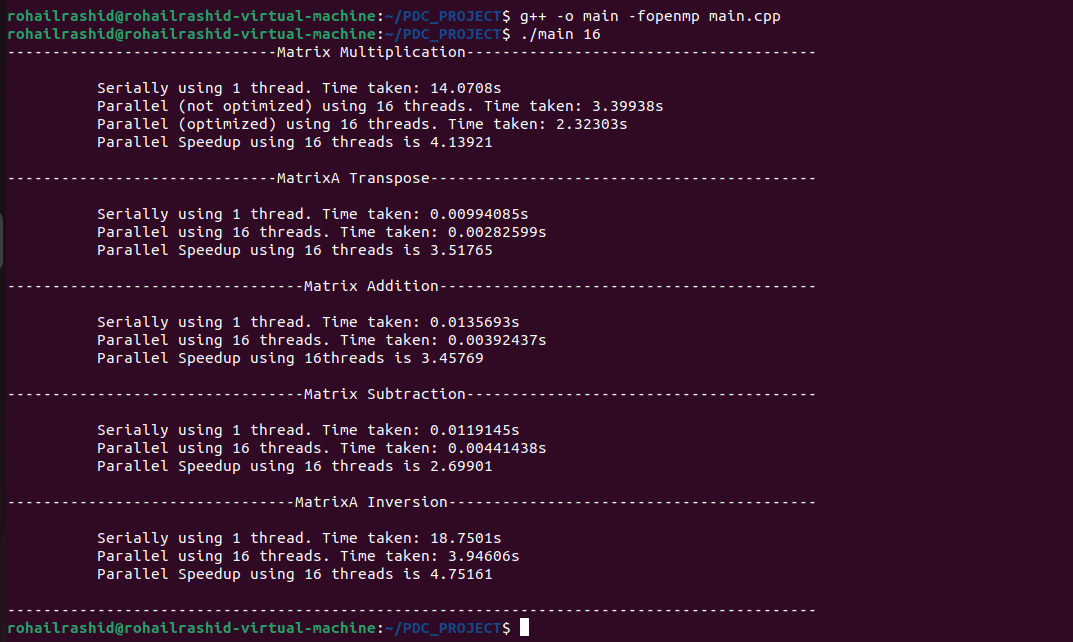
****

**Matrix Operations using openmp and Size 1000**

***Size 1000 threads 8***

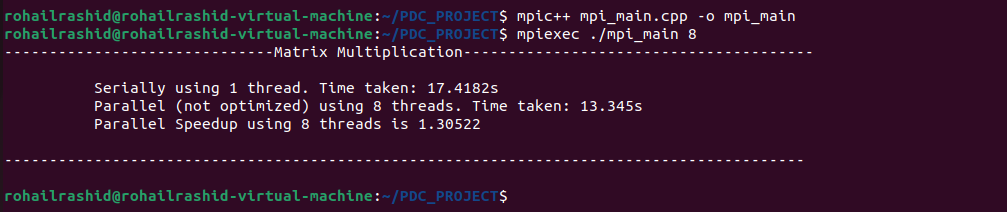
****

***Size 1000 threads 16***

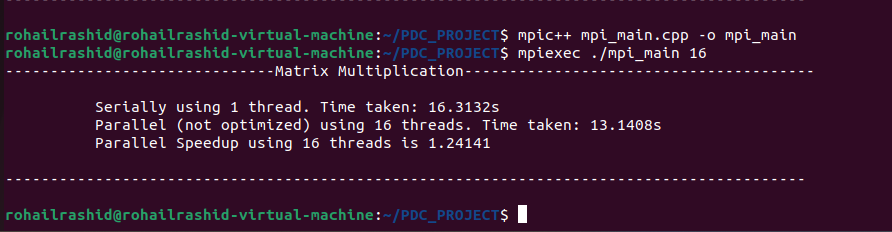
****

**Matrix Operations using MPI and Size 1000**

***Size 1000 processes 8***

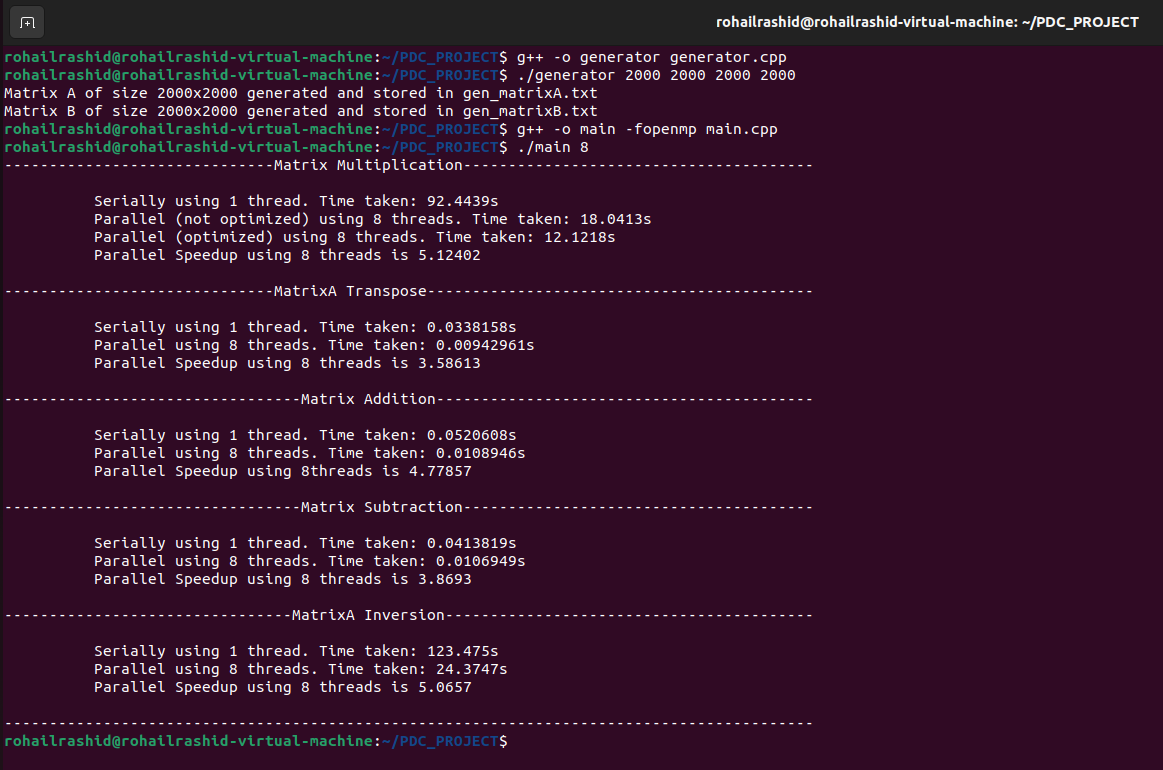
****

***Size 1000 processes 16***

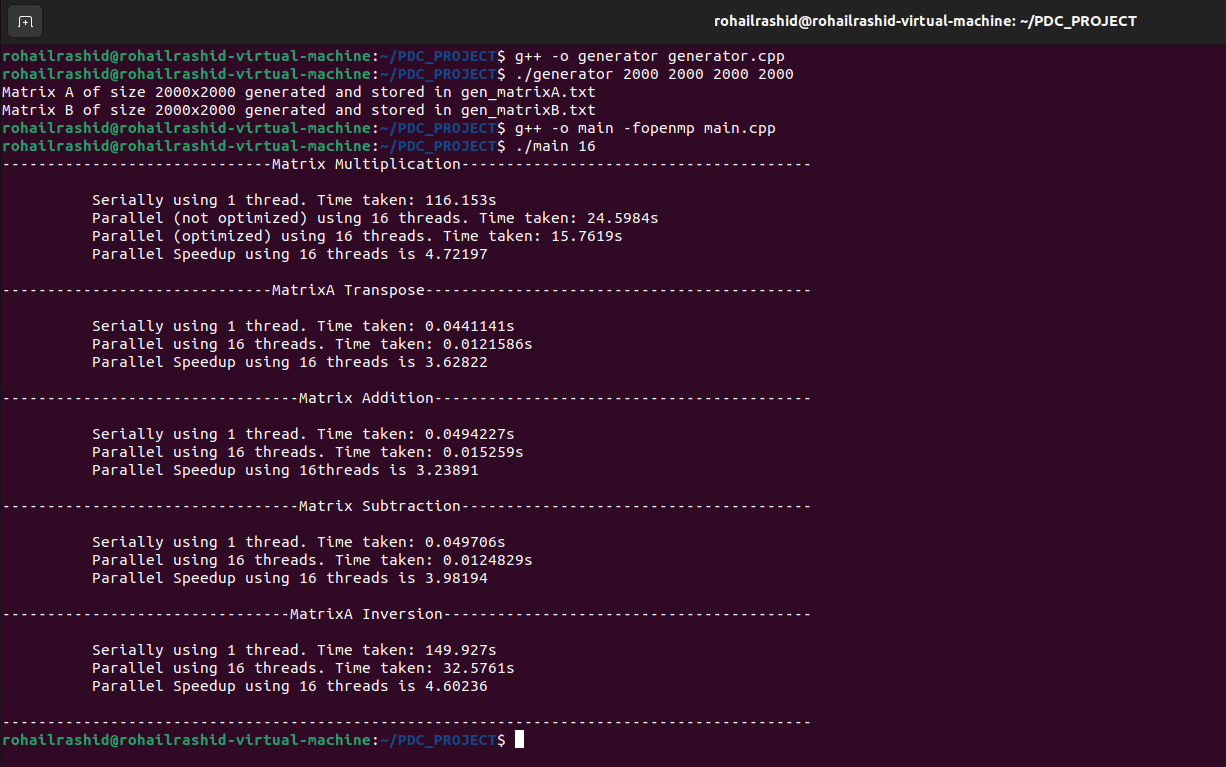
****

**Matrix Operations using openmp and Size 2000**

***Size 2000 threads 8***

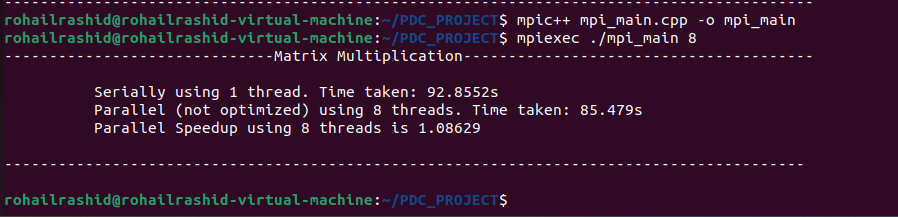
******

***Size 2000 threads 16***

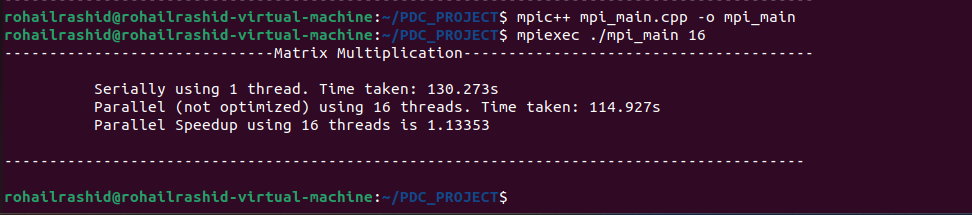
****

**Matrix Operations using MPI and Size 2000**

***Size 2000 processes 8***

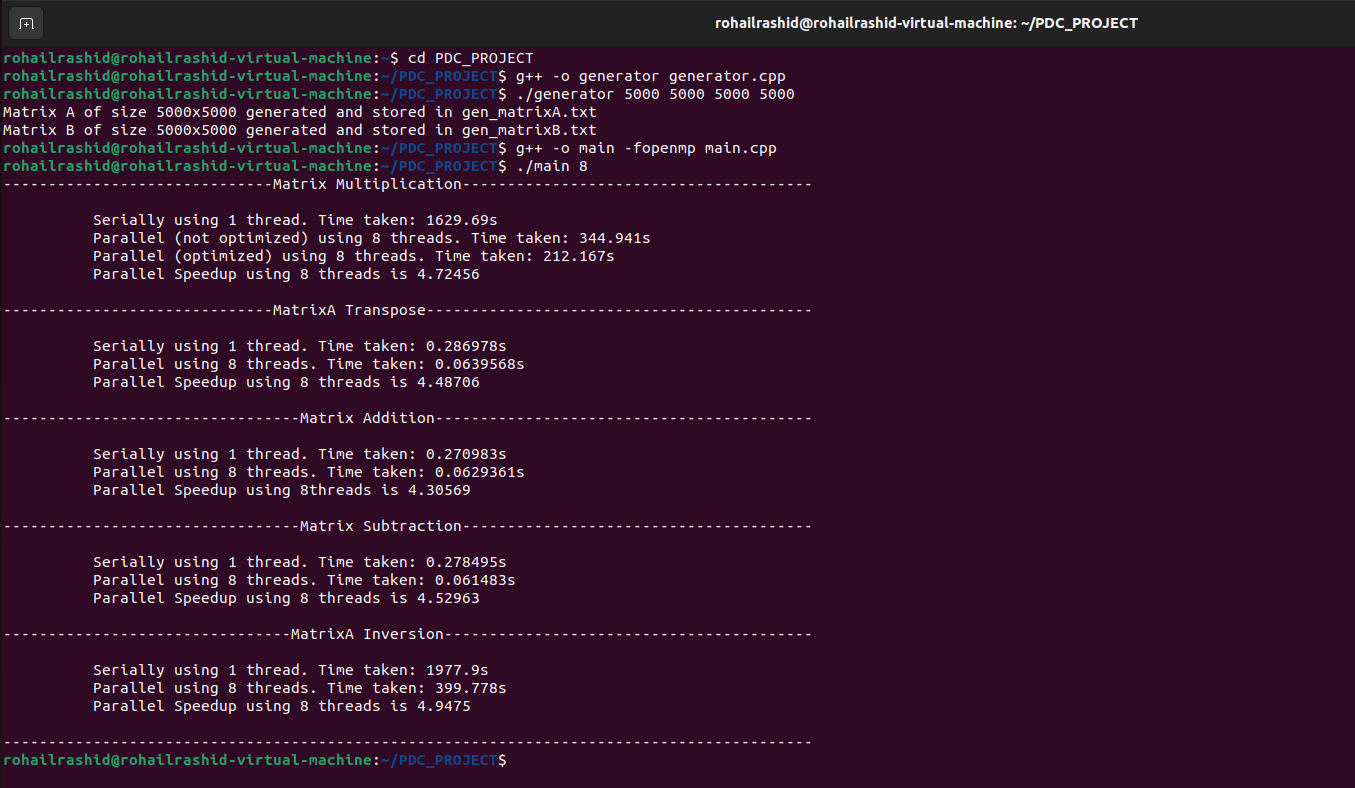
******

***Size 2000 processes 16***

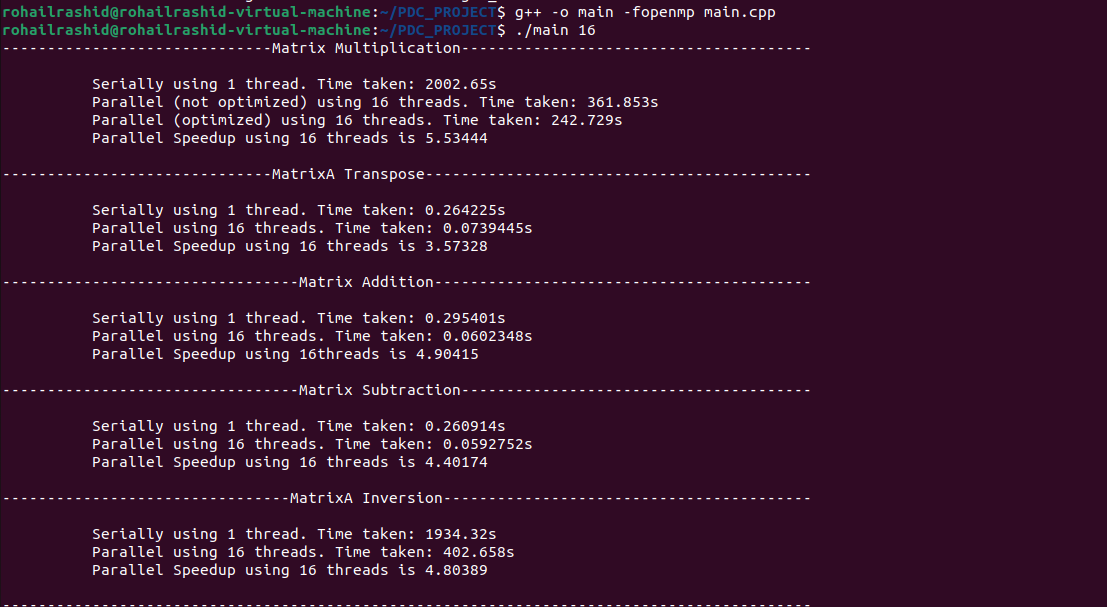
****

**Matrix Operations using openmp and Size 5000**

***Size 5000 threads 8***

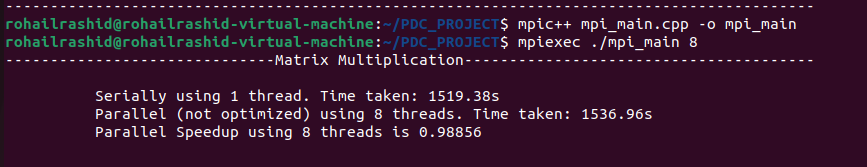
******

***Size 5000 threads 16***

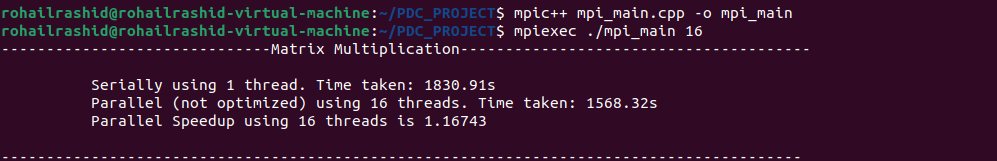
****

**Matrix Operations using MPI and Size 5000**

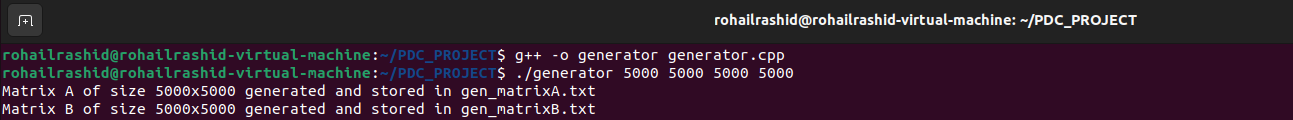
***Size 5000 processes 8***

******

***Size 5000 processes 16***

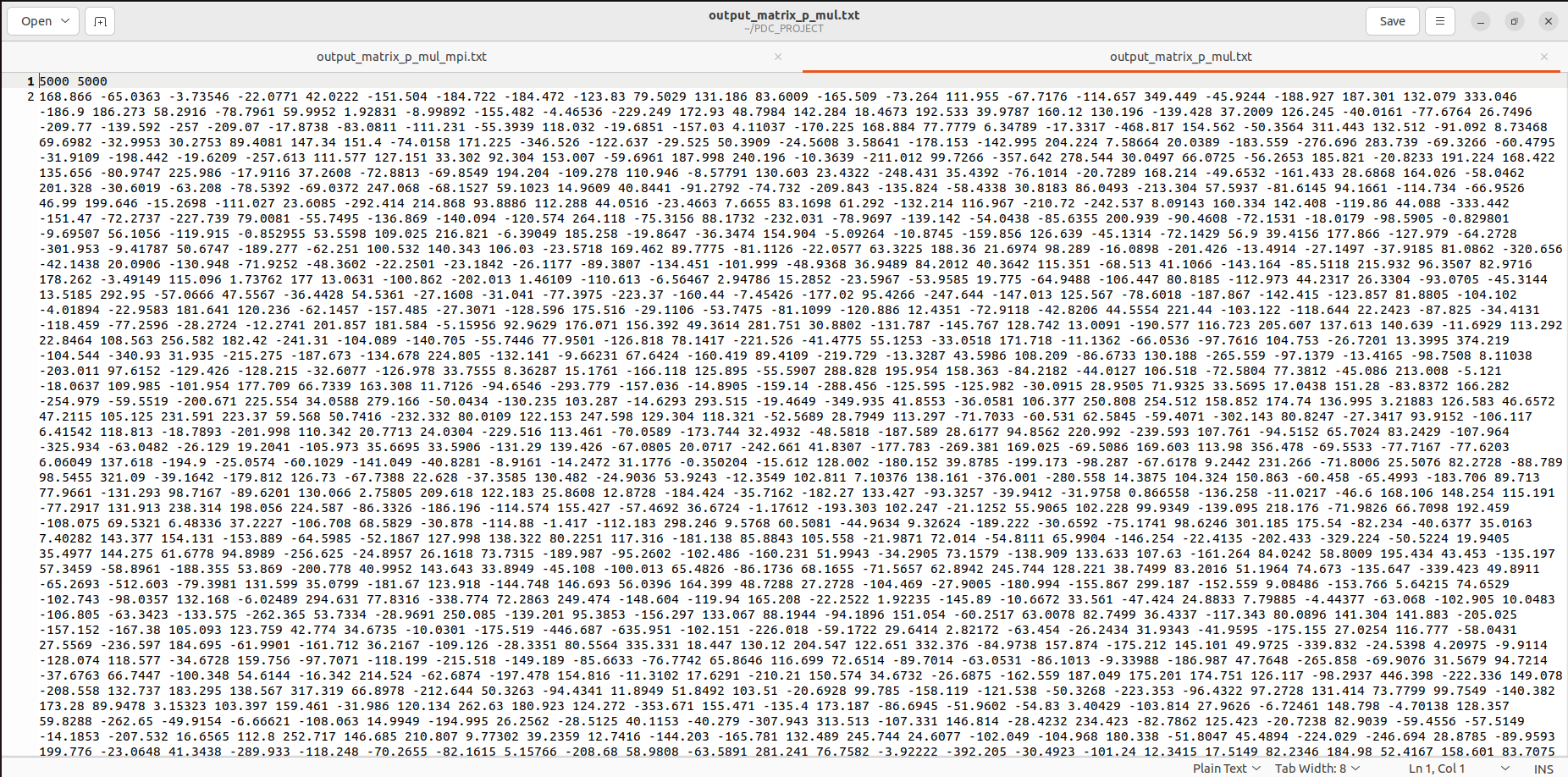
****

***Matrix Generation***

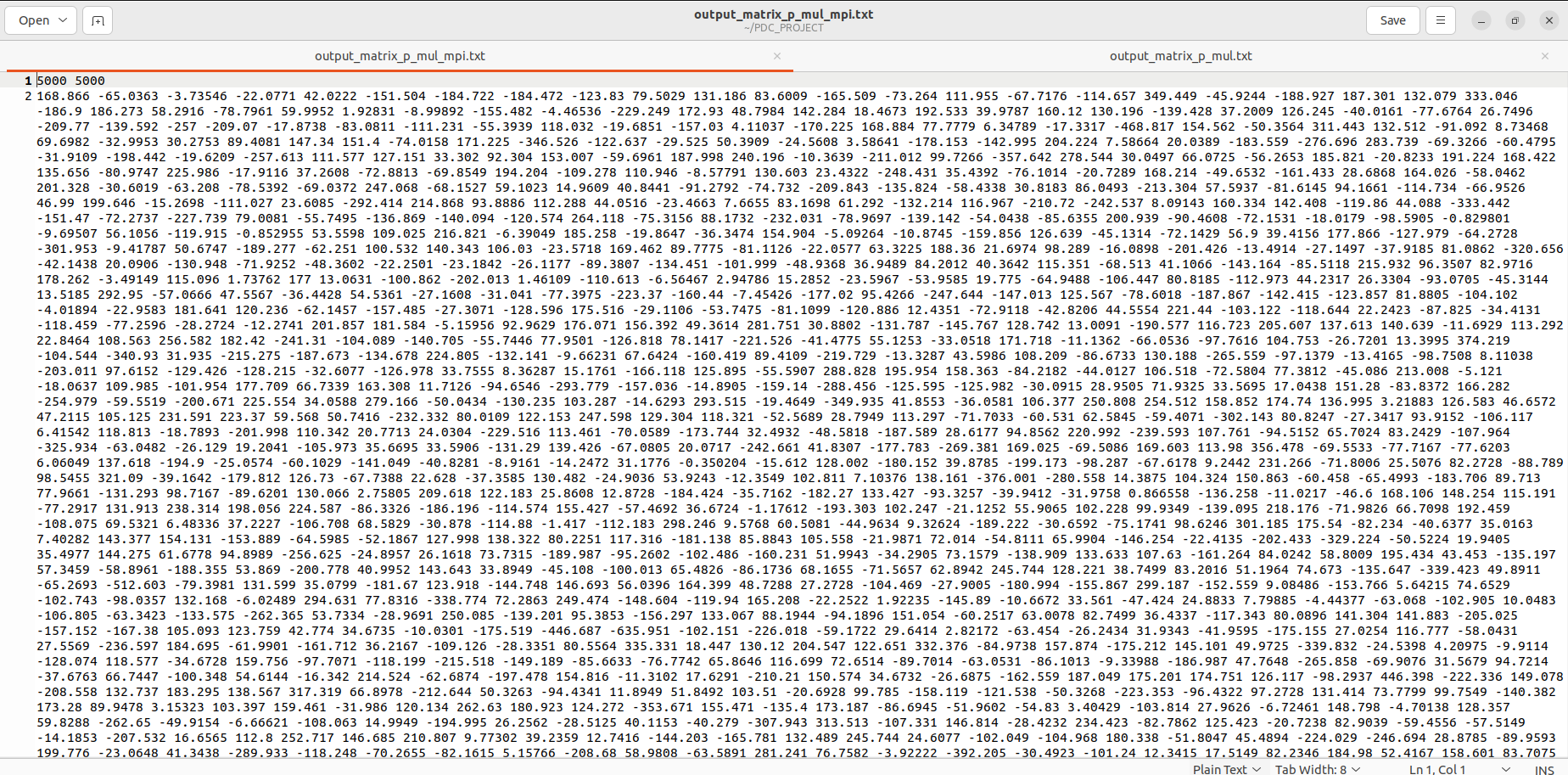
****

***Results***

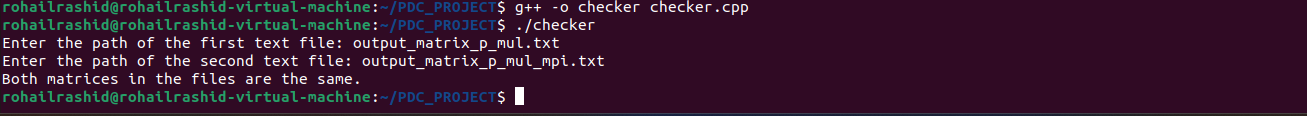
This is OpenMP matrix multiplication result

****

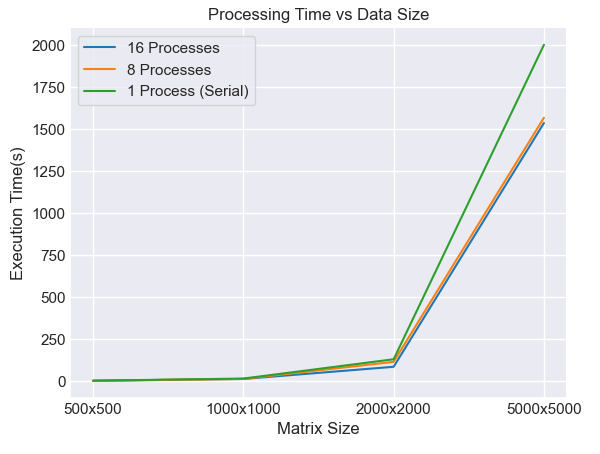
This is MPI matrix multiplication result

****

Here we have validated both open mp and mpi results and they are correct



***Analysis***



The above graph shows that as the size of the matrix increases the processing time too increases drastically.

It can also be noted that serial takes more time than parallel using 8 or 16 processes therefore as no of processes increases the processing time reduces.

**Note: In the graph the execution time using multiple processes is the MPI time.**

***Issues faced in Conversion***

We converted our parallel matrix multiplication part of openmp into mpi. We faced few issues such as parallel time was larger than serial so we had to write a proper mpi code that would satisfy the correct execution time and make it beneficial to use mpi.

***Conclusion***

In conclusion, the collaborative use of MPI and OpenMP in matrix operations marks a significant leap in computational efficiency. MPI's prowess in distributed-memory systems complements OpenMP's shared-memory parallelism, optimizing tasks like multiplication, inversion, and transposition. This dual strategy enhances speed and scalability, particularly beneficial for large matrices. The parallel approach not only accelerates computation but also ensures effective resource utilization. Our exploration has deepened our understanding of matrix operations and equipped us with essential skills in Parallel and Distributed Computing. This synergistic blend stands as a powerful solution, unlocking new levels of performance in the dynamic landscape of computational mathematics.

***References***

* <https://vaibhaw-vipul.medium.com/matrix-multiplication-optimizing-the-code-from-6-hours-to-1-sec-70889d33dcfa>
* <https://www.c-sharpcorner.com/article/parallel-programming-with-openmp-in-cpp-matrix-multiplication-example/>
* <https://www.christianbaun.de/CGC1718/Skript/CloudPresentation_Shamima_Akhter.pdf>
* <https://cse.buffalo.edu/faculty/miller/Courses/CSE633/thanigachalam-Spring-2014-CSE633.pdf>