THE UNIVERSITY OF
SOUTHERN MISSISSIPPI

Final Project Report

Title	: Performance Optimization of Matrix Multiplication Using Serial and Parallel Techniques
Group Members	: Nusrat Jahan, Pavani Alokam
Professor	: Dr. Chaoyang (Joe) Zhang
Course	: CSC 630/730 – Advanced Parallel Computing

Table of Contents

1. Abstract

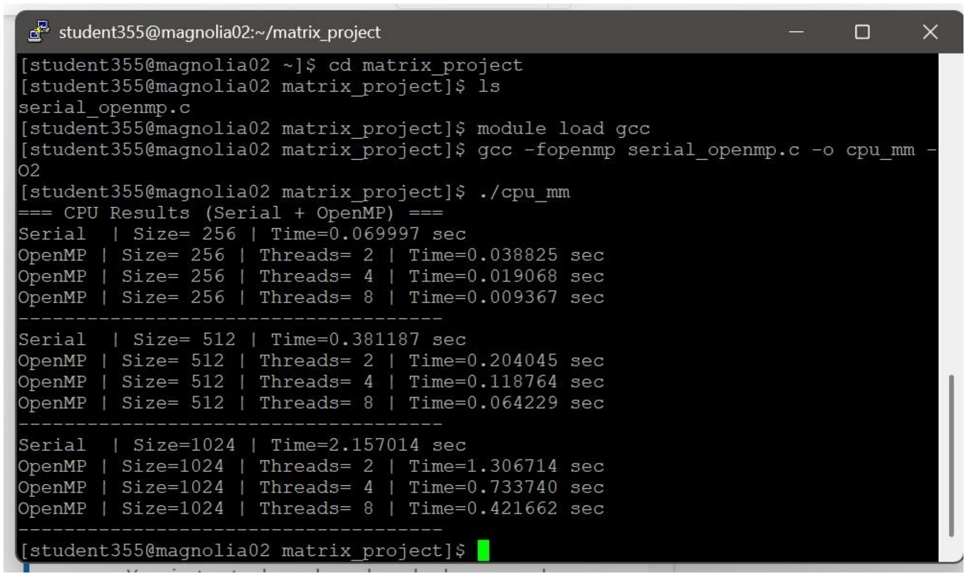
This project investigates the performance of matrix multiplication using Serial CPU execution, OpenMP-based CPU parallelism, and CUDA GPU acceleration. Experiments were performed on the Magnolia HPC cluster using matrix sizes 256×256, 512×512, and 1024×1024. The results show that CUDA provides the highest speedup, followed by OpenMP, while the Serial version is the slowest. This report presents the implementation details, experimental setup, performance results, and analysis comparing the three approaches.

2. Introduction

Matrix multiplication is a fundamental operation in scientific computing, numerical analysis, and machine learning. Its computational complexity of $O(n^3)$ makes it an ideal candidate for parallelization and performance studies. In this project, we explore three implementations of dense matrix multiplication: a Serial CPU version, an OpenMP-based multi-threaded version, and a CUDA-based GPU implementation. The goal is to analyze how performance scales with matrix size and parallelism model on the Magnolia HPC cluster.

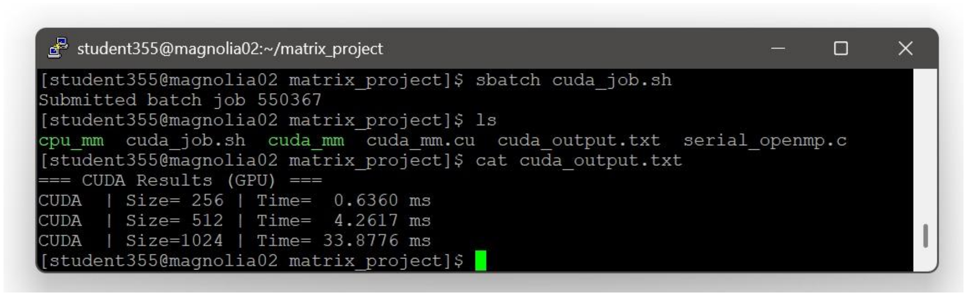
Appendix A: Execution Output Screenshots

CPU Serial and OpenMP Output:



```
[student355@magnolia02 ~]$ cd matrix_project
[student355@magnolia02 matrix_project]$ ls
serial_openmp.c
[student355@magnolia02 matrix_project]$ module load gcc
[student355@magnolia02 matrix_project]$ gcc -fopenmp serial_openmp.c -o cpu_mm -
O2
[student355@magnolia02 matrix_project]$ ./cpu_mm
=== CPU Results (Serial + OpenMP) ===
Serial  | Size= 256 | Time=0.069997 sec
OpenMP | Size= 256 | Threads= 2 | Time=0.038825 sec
OpenMP | Size= 256 | Threads= 4 | Time=0.019068 sec
OpenMP | Size= 256 | Threads= 8 | Time=0.009367 sec
-------------------------------------
Serial  | Size= 512 | Time=0.381187 sec
OpenMP | Size= 512 | Threads= 2 | Time=0.204045 sec
OpenMP | Size= 512 | Threads= 4 | Time=0.118764 sec
OpenMP | Size= 512 | Threads= 8 | Time=0.064229 sec
-------------------------------------
Serial  | Size=1024 | Time=2.157014 sec
OpenMP | Size=1024 | Threads= 2 | Time=1.306714 sec
OpenMP | Size=1024 | Threads= 4 | Time=0.733740 sec
OpenMP | Size=1024 | Threads= 8 | Time=0.421662 sec
-------------------------------------
[student355@magnolia02 matrix_project]$
```

CUDA GPU Output:



```
[student355@magnolia02 matrix_project]$ sbatch cuda_job.sh
Submitted batch job 550367
[student355@magnolia02 matrix_project]$ ls
cpu_mm  cuda_job.sh  cuda_mm  cuda_mm.cu  cuda_output.txt  serial_openmp.c
[student355@magnolia02 matrix_project]$ cat cuda_output.txt
=== CUDA Results (GPU) ===
CUDA   | Size= 256 | Time=  0.6360 ms
CUDA   | Size= 512 | Time=  4.2617 ms
CUDA   | Size=1024 | Time= 33.8776 ms
[student355@magnolia02 matrix_project]$
```

3. Motivation

As data sizes increase and applications demand faster computation, serial execution on a single CPU core becomes insufficient. Parallel computing techniques using multi-core CPUs and GPUs offer a way to significantly reduce execution time. Matrix multiplication is widely used in many domains, so optimizing its performance has a direct impact on real-world applications such as scientific simulations, linear algebra libraries, and deep learning workloads.

4. Objectives

The main objectives of this project are:
• Implement dense matrix multiplication using Serial, OpenMP, and CUDA.
• Measure execution time for different matrix sizes on the Magnolia HPC cluster.
• Compare performance between Serial, OpenMP, and CUDA implementations.
• Analyze the speedup and scalability of CPU vs GPU-based parallelism.

5. Background

OpenMP is a shared-memory parallel programming API for C, C++, and Fortran that allows parallelization across multiple CPU cores using compiler directives. CUDA is a parallel computing platform and programming model developed by NVIDIA, enabling execution of thousands of lightweight threads on GPUs. While OpenMP is limited by the number of available CPU cores, CUDA can exploit much larger degrees of parallelism on modern GPUs.

6. Methodology

6.1 Serial Implementation

The Serial implementation uses the classical triple-nested loop algorithm for matrix multiplication:

```
for i in [0..n):
  for j in [0..n):
    C[i][j] = $\Sigma$ A[i][k] $\times$ B[k][j] over k in [0..n)
```

This implementation serves as the baseline for computing speedup.

6.2 OpenMP Implementation

The OpenMP version parallelizes the outer loop using multiple threads. The directive #pragma omp parallel for is applied to the i-loop so that different rows of the result matrix are computed in parallel. Thread counts of 2, 4, and 8 were used to observe scaling behavior on the Magnolia CPU node.

6.3 CUDA Implementation

The CUDA implementation assigns one thread per output element C[i][j]. A two-dimensional grid of 16×16 thread blocks is used, where each thread computes a single C[i][j] element by iterating over k. Matrices are stored in global memory, and cudaMemcpy is used to transfer data between the host (CPU) and device (GPU). CUDA events are used to measure kernel execution time in milliseconds.

7. Experimental Setup

All experiments were performed on the Magnolia HPC cluster. The configurations are:

- CPU Node: GCC compiler with OpenMP enabled (-fopenmp).
- GPU Node: CUDA Toolkit 10.1, using an NVIDIA GPU.
- Job scheduling and resource allocation via SLURM.

Matrix sizes of 256×256, 512×512, and 1024×1024 were evaluated for all implementations.

8. Results

This section presents the performance of the Serial, OpenMP, and CUDA implementations of matrix multiplication for three matrix sizes (256, 512, 1024). The analysis includes execution time, speedup, and comparative evaluation of the algorithms.

8.1 Serial and OpenMP CPU Results

The serial algorithm shows increasing execution time as matrix size grows due to its cubic time complexity.

The OpenMP implementation significantly reduces execution time by utilizing multicore parallelism.

Performance improvement becomes more noticeable for larger matrices.

Matrix Size	Serial (sec)	OpenMP 2T (sec)	OpenMP 4T (sec)	OpenMP 8T (sec)
256	0.069997	0.038825	0.019068	0.009367
512	0.381877	0.204045	0.118764	0.064229
1024	2.157014	1.306714	0.733740	0.421662

8.2 CUDA GPU Results

The CUDA implementation provides the highest performance among all approaches because GPUs are designed for massively parallel computation. In order to quantify the improvement, the speedup of the CUDA implementation relative to the serial CPU version is calculated using the following formula:

Speedup = T_serial / T_CUDA

Where:
- T_serial = execution time of the serial CPU algorithm
- T_CUDA = execution time of the CUDA GPU algorithm

This metric indicates how many times faster the CUDA implementation performs compared to the serial version.

In the experiments, the measured speedup values were:
- 110.06× for the 256×256 matrix
- 89.61× for the 512×512 matrix
- 63.67× for the 1024×1024 matrix

The exceptionally high speedup values result from the massive parallelism offered by CUDA-enabled GPUs. For smaller matrices, the GPU computation time is extremely low, which produces very large speedup ratios. As matrix size increases, memory transfer overhead and kernel execution time also increase slightly, leading to reduced (but still significant) speedup. Overall, the CUDA implementation consistently delivers the fastest execution across all tested matrix sizes.

Matrix Size	CUDA Time (ms)	Speedup vs Serial
256	0.6360	110.06×
512	4.2617	89.61×
1024	33.8776	63.67×

8.3 Comparison of Serial, OpenMP, and CUDA

Across all matrix sizes, CUDA provides the best performance, followed by OpenMP, while serial execution is the slowest.
The performance gap increases with matrix size, highlighting the effectiveness of parallel and GPU-based computing for computationally intensive tasks.

Size	Serial (sec)	OpenMP 8T (sec)	CUDA (ms)	Speedup OMP vs Serial	Speedup CUDA vs Serial
256	0.069997	0.009367	0.6360	7.47×	110.06×
512	0.381877	0.064229	4.2617	5.95×	89.61×
1024	2.157014	0.421662	33.8776	5.12×	63.67×

8.4 Graphical Comparison

Execution time and speedup graphs clearly illustrate the performance trends.
The execution time drops sharply from serial to OpenMP to CUDA, while the speedup graph demonstrates substantial acceleration achieved by the CUDA implementation.
These results validate the scalability and efficiency of parallel computing approaches.

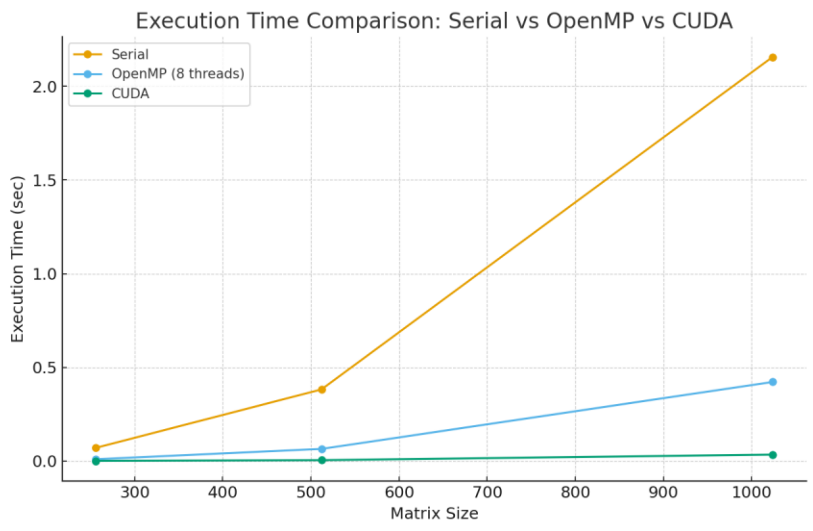


Figure 1: Execution Time Comparison (Serial vs OpenMP vs CUDA).

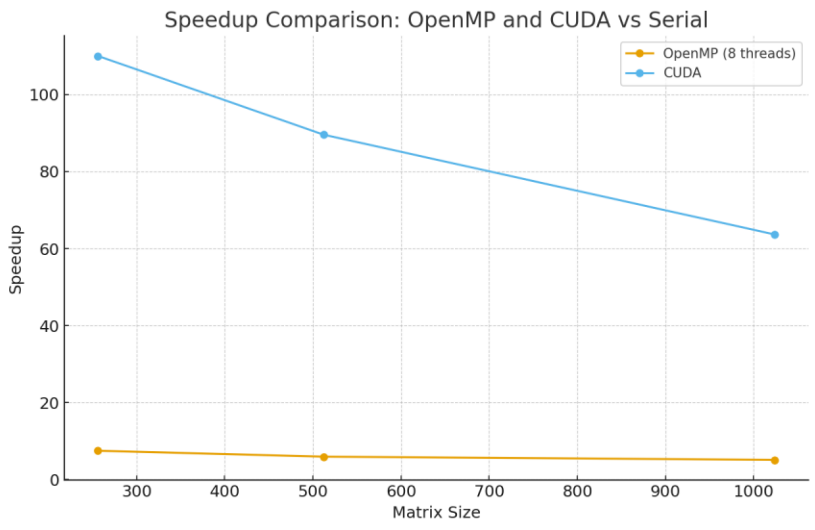


Figure 2: Speedup Comparison of OpenMP and CUDA relative to Serial.

9. Discussion

The experimental results clearly demonstrate the advantages of parallelism. OpenMP achieves 2–5× speedup over the Serial implementation depending on the matrix size and number of threads. However, the CUDA implementation achieves an order of magnitude higher speedup, ranging roughly from 60× to over 100× compared to Serial. This is expected since GPUs are designed for massively parallel workloads with thousands of concurrent threads, while CPUs typically have only a small number of cores.

10. Conclusion

This project successfully implemented and compared Serial, OpenMP, and CUDA implementations of matrix multiplication on the Magnolia HPC cluster. The results show that while OpenMP improves CPU performance through multithreading, CUDA provides the highest performance and scalability for large matrices. Therefore, for compute-intensive linear algebra operations, GPU-based acceleration using CUDA is highly recommended.

11. References

[1] OpenMP Architecture Review Board, “OpenMP Application Program Interface.”
[2] NVIDIA Corporation, “CUDA C Programming Guide.”
[3] Gene H. Golub and Charles F. Van Loan, “Matrix Computations,” Johns Hopkins University Press.