National Textile University, Faisalabad

Department of Computer Science

Name:	Fahad Ali
Class:	BSCS-A 6th
Registration No:	22-NTU-CS-1154
Activity:	Assignment
Course Name:	Parallel and Distributed Computing
Submitted To:	*Sir. Nasir Mahmood*
Submission Date:	8st March, 2025

Project & Git Initialization:

Commit 1: Initialize project structure

Created a Git repository and added files like .gitignore to ignore unnecessary files and READMD.md with a brief project overview.

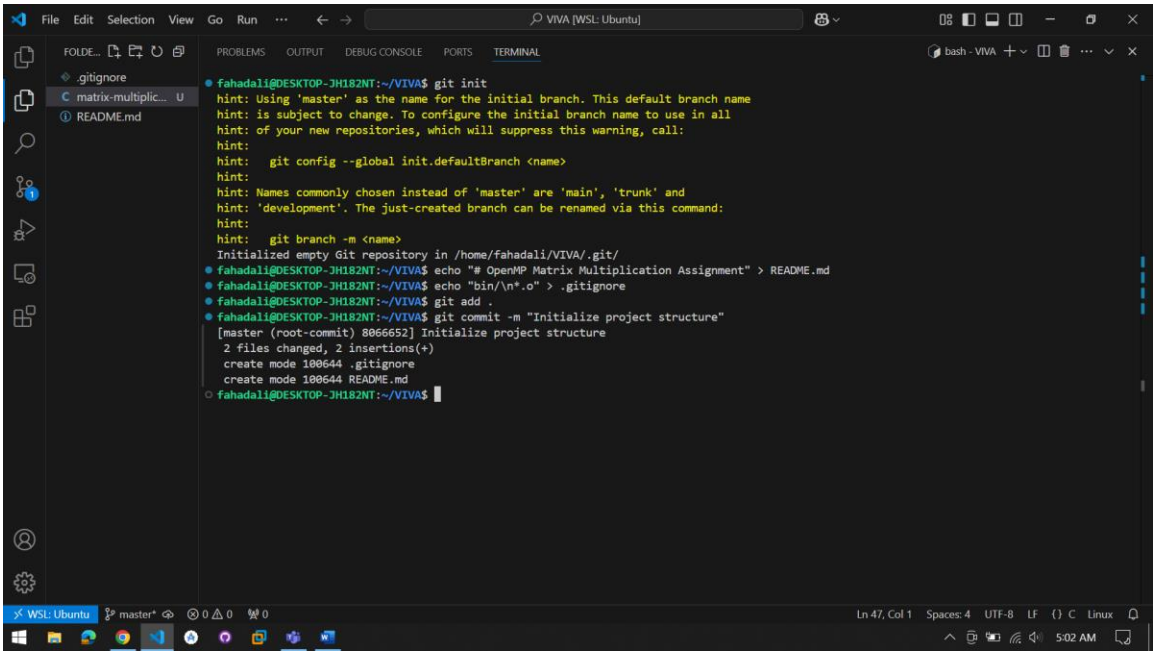

Figure 1

Implemented the code of the Sequential Matrix Multiplication

Commit 2: adding the sequential matrix multiplication code

Here I implemented the matrix multiplication using sequential method without using the OpenMP. And executed the program 10 times and recorded the results along with the average of the execution time.

Here, I'm using 2 matrixes of dimensions 2 with a size of 500 each.

The multiplication is done here using traditional loop method.

Code:

Two functions multiply_matrices() and get_execution_time() is used here. The first function is responsible for multiplying the 2 matrixes. The other function is responsible for assigning random values to the matrixes and then calculating the execution time.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 500
void multiply_matrices(int A[N][N], int B[N][N], int C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

double get_execution_time() {
    int A[N][N], B[N][N], C[N][N];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i][j] = rand() % 10;
            B[i][j] = rand() % 10;
        }
    }
    clock_t start = clock();
    multiply_matrices(A, B, C);
    clock_t end = clock();

    return (double)(end - start) / CLOCKS_PER_SEC;
}
int main() {
    double total_time = 0.0;
    int runs = 10;

    for (int i = 0; i < runs; i++) {
        total_time += get_execution_time();
    }
    printf("Average Execution Time (Sequential): %.6f seconds\n", total_time /
runs);
```

```
    return 0;
}
```

Output:

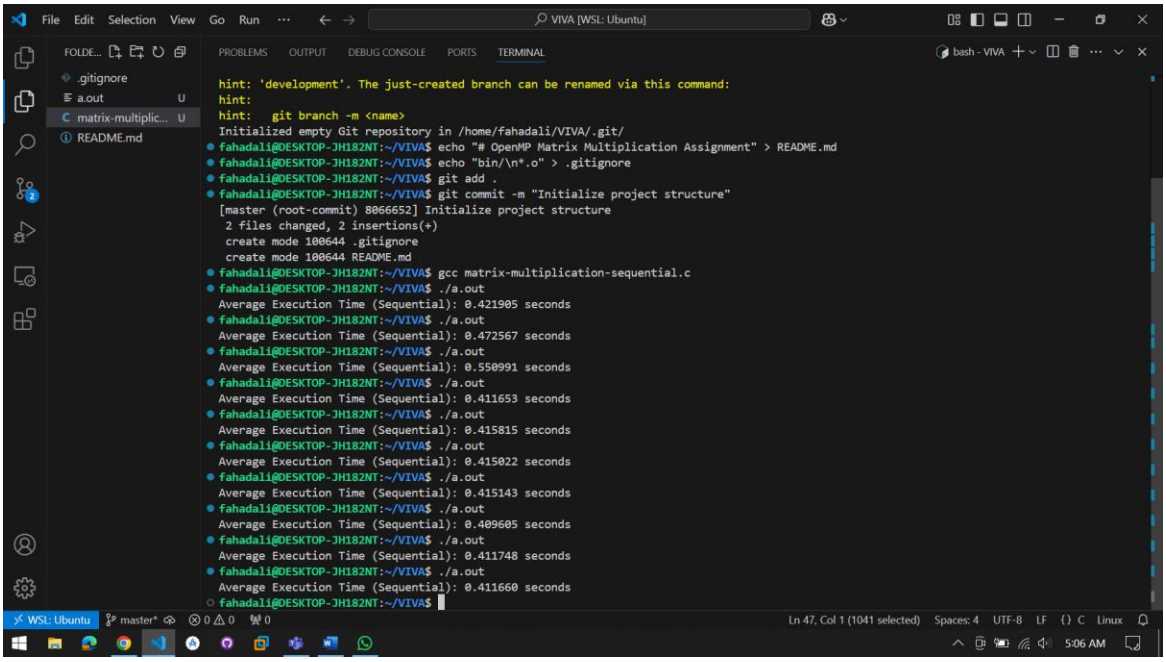

Figure 2

As you can see I've compiled the .c file and then executed it using **./a.out** command. And on console I got the Average Execution time which is basically the average of time required to perform matrix multiplication 10 times. This program was tested for 10 times and the range of Average Execution time we got is **(0.411653s – 0.550991s)** note mostly the out was near to **0.4s.**

Git:

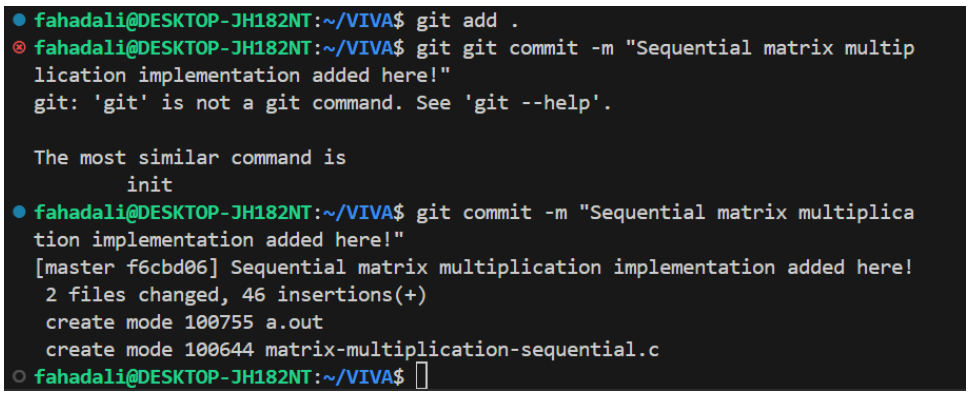

Figure 3

Implemented the code of OpenMP Matrix multiplication

Commit 3: OpenMP parallelization matrix multiplication

Here we can see by using OpenMP, the execution time has decreased and it makes the multiplication more efficient and increases its overall performance.

Used the **#pragma omp parallel for** for loop parallelization.

The use of both static and dynamic scheduling along the correct variable scope is ensured here.

Code:

Two functions multiply_matrices_parallel() and get_execution_time_parallel() is used here. The first function is responsible for multiplying the 2 matrixes using schedule and parallel loop. The other function is responsible for assigning random values to the matrixes and then calculating the execution time.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

#define N 500
#define NUM_THREADS 4

void multiply_matrices_parallel(int A[N][N], int B[N][N], int C[N][N]) {
    #pragma omp parallel for schedule(dynamic) num_threads(NUM_THREADS)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}



double get_parallel_execution_time() {
```

```
    int A[N][N], B[N][N], C[N][N];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i][j] = rand() % 10;
            B[i][j] = rand() % 10;
        }
    }

    double start = omp_get_wtime();
    multiply_matrices_parallel(A, B, C);
    double end = omp_get_wtime();

    return end - start;
}

int main() {
    double total_time = 0.0;
    int runs = 10;

    for (int i = 0; i < runs; i++) {
        total_time += get_parallel_execution_time();
    }

    printf("Average Execution Time (Parallel): %.6f seconds\n", total_time /
runs);
    return 0;
}
```

Output:

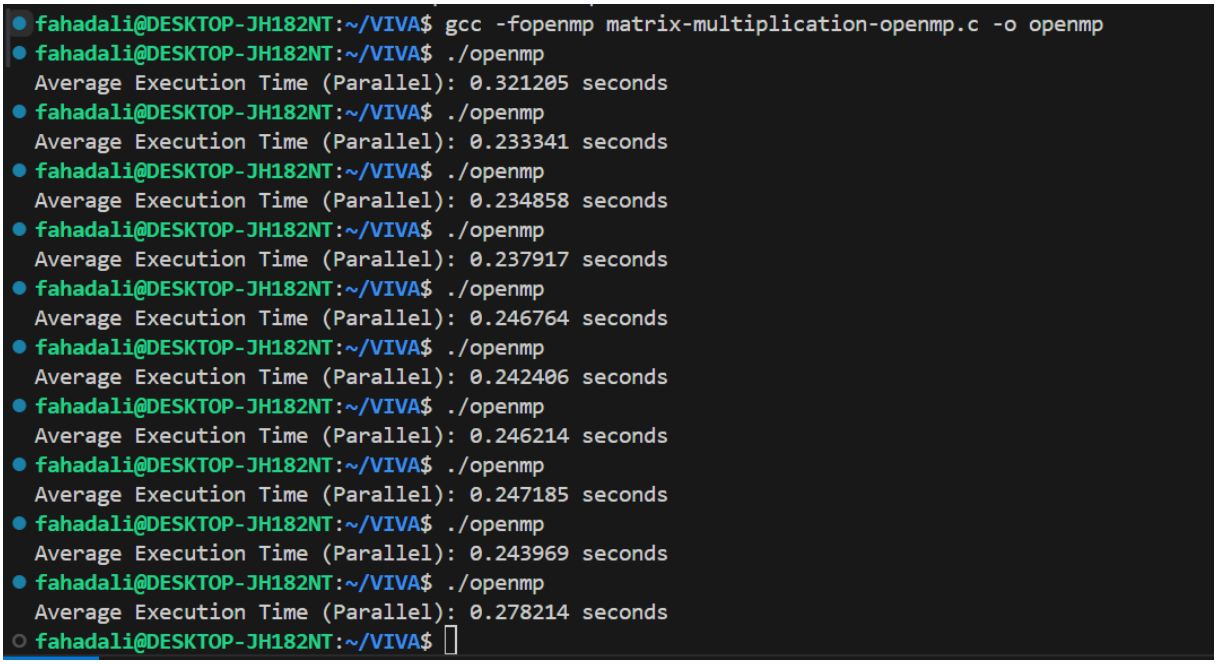

Figure 4

As you can see I've compiled the .c file and then executed it using **./openmp.out** command. And on console I got the Average Execution time which is basically the average of time required to perform matrix multiplication 10 times. This program was tested for 10 times and the range of Average Execution time we got is **(0.233341s – 0.278214s)** note mostly the out was near to **0.23s.**

Git:

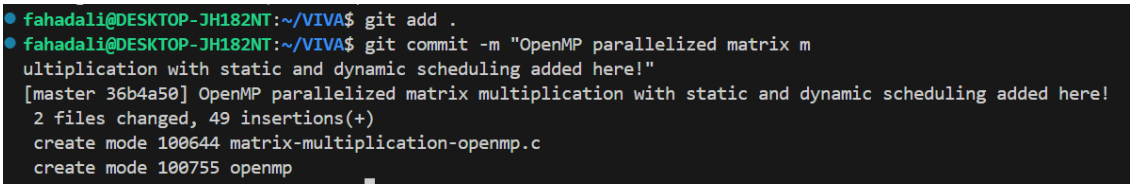

Figure 5

Performance Evaluation

Here we can conclude that the use of OpenMP made the matrix multiplication much faster than using the traditional looping method. This shows how the use of parallel computing and distribution can increase overall performance by utilizing multiple threads efficiently.