Matrix Multiplication Project Report

Student Name: Aline Uwimana

Student ID: LS2425234

Submission Date: March 25,2025

System Configuration

TABLE OF CONTENT

-**CPU Model**: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz

-**Memory Size**: 3.7Gi

-**Operating System Version**: P5S5GQC 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux.

-**Compiler Version**: gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

-**Python Version**: Python 3.10.12

Implementation Details

1. **Input**:
   - The user is prompted to input two matrices (First and Second matrices).
   - The dimensions of the matrices must be compatible for multiplication (i.e, the number of columns in the first matrix must equal the number of rows in the second matrix).
2. **Matrix Multiplication**: The multiplication of two matrices is performed using the The following formula:

$$c_{ij} = \sum_{k=0}^{n-1} A_{ik} \times B_{kj}$$

where: - (A) is the first matrix. - (B) is the second matrix. - (C) is the result matrix.

3. **Output**: -The program will display the resulting matrix after multiplication .

C Language Implementation

-**Source Code**:

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

```c
#define MAX 10  // Maximum size for matrix dimensions

// Function to multiply two matrices
void multiplyMatrices(int firstMatrix[][MAX], int secondMatrix[][MAX], int resultMatrix[][MA
    // Ensure the number of columns in the first matrix is equal to the number of rows in th
    if (firstCol != secondRow) {
        printf("Matrix multiplication is not possible. The number of columns in the first ma
        return;
    }

    // Initialize resultMatrix to 0
    for (int i = 0; i < firstRow; i++) {
        for (int j = 0; j < secondCol; j++) {
            resultMatrix[i][j] = 0;
        }
    }

    // Perform the matrix multiplication
    for (int i = 0; i < firstRow; i++) {
        for (int j = 0; j < secondCol; j++) {
            for (int k = 0; k < firstCol; k++) {
                resultMatrix[i][j] += firstMatrix[i][k] * secondMatrix[k][j];
            }
        }
    }
}

// Function to display the matrix
void displayMatrix(int matrix[][MAX], int row, int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    clock_t start_time, end_time;
    double execution_time;

    int firstMatrix[MAX][MAX], secondMatrix[MAX][MAX], resultMatrix[MAX][MAX];
    int firstRow, firstCol, secondRow, secondCol;

    // Start the timer
    start_time = clock();
```

```c
    // Input dimensions of the first matrix
    printf("Enter the number of rows and columns for the first matrix: ");
    scanf("%d %d", &firstRow, &firstCol);

    // Input elements of the first matrix
    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < firstRow; i++) {
        for (int j = 0; j < firstCol; j++) {
            printf("Enter element a%d%d: ", i + 1, j + 1);
            scanf("%d", &firstMatrix[i][j]);
        }
    }

    // Input dimensions of the second matrix
    printf("Enter the number of rows and columns for the second matrix: ");
    scanf("%d %d", &secondRow, &secondCol);

    // Input elements of the second matrix
    printf("Enter elements of the second matrix:\n");
    for (int i = 0; i < secondRow; i++) {
        for (int j = 0; j < secondCol; j++) {
            printf("Enter element b%d%d: ", i + 1, j + 1);
            scanf("%d", &secondMatrix[i][j]);
        }
    }

    // Perform matrix multiplication
    multiplyMatrices(firstMatrix, secondMatrix, resultMatrix, firstRow, firstCol, secondRow,

    // Display the result
    printf("Result of Matrix Multiplication:\n");
    displayMatrix(resultMatrix, firstRow, secondCol);

    // End the timer and calculate the time taken
    end_time = clock();
    execution_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

    // Print the time taken
    printf("Execution Time: %.6f seconds\n", execution_time);

    return 0;
}
```

-**Compilation Command**:

Description of step by step how to compile c program using GCC

1.Open WSL terminal.

2.Make sure gcc is install/ if not install it.

3. Create a c file by navigating to the directory where you want to save your c program. Inside create a new c file by using nano or any other editor.
4. Copy and paste your c code inside the nano editor.
5. save and exit nano editor.
6. In wsl terminal run gcc compiler to compile your c code using this command " gcc matrix.c -o matrix" .

-Execution Command

I runned the compiled c program using this command " ./matrix " inside wsl after that I entered the input values and got the results.

Python Language Implementation

-Source Code

```
import time
start_time = time.time()
def get_matrix_input(rows, cols, name): #Function to get matrix input from the user.
    print(f"Enter the elements of {name} matrix row by row:")
    matrix = []
    for i in range(rows):
        row = list(map(int, input(f"Row {i + 1}: ").split()))
        while len(row) != cols:
            print(f"Invalid input! Please enter exactly {cols} numbers.")
            row = list(map(int, input(f"Row {i + 1}: ").split()))
        matrix.append(row)
    return matrix

def multiply_matrices(A, B):  #Function to multiply two matrices.
    ROWS_A = len(A)
    COLS_A = len(A[0])
    ROWS_B = len(B[0])
    COLS_B = len(B[0])

    # Check if matrix multiplication is possible
    if COLS_A != ROWS_B:
        print("Error: Number of columns in Matrix A must be equal to number of rows in Matri
        return None

    # Initialize result matrix with zeros
    result = [[0] * COLS_B for _ in range(ROWS_A)]
```

```
    # Perform matrix multiplication
    for i in range(ROWS_A):
        for j in range(COLS_B):
            for k in range(COLS_A):
                result[i][j] += A[i][k] * B[k][j]

    return result

def display_matrix(matrix): #Function to display a matrix.
   if matrix is None:
        return
    for row in matrix:
        print(" ".join(map(str, row)))

# Get user input for matrix dimensions
ROWS_A = int(input("Enter number of rows for Matrix A: "))
COLS_A = int(input("Enter number of columns for Matrix A: "))

ROWS_B = int(input("Enter number of rows for Matrix B: "))
COLS_B = int(input("Enter number of columns for Matrix B: "))

# Ensure matrix multiplication is possible
if COLS_A != ROWS_B:
    print("Error: Number of columns in Matrix A must be equal to number of rows in Matrix B!
else:
    # Get matrix inputs
    A = get_matrix_input(ROWS_A, COLS_A, "Matrix A")
    B = get_matrix_input(ROWS_B, COLS_B, "Matrix B")

    # Display input matrices
    print("\nMatrix A:")
    display_matrix(A)

    print("\nMatrix B:")
    display_matrix(B)

    # Perform matrix multiplication
    result = multiply_matrices(A, B)

    # Display the resultant matrix
    if result:
        print("\nResultant Matrix:")
        display_matrix(result)
end_time = time.time()
execution_time = end_time - start_time
```

```
print(f"Execution Time: {execution_time:.6f} seconds")
```

-**Execution command**

step by step of how to run python script. 1. Ensure that python is installed in WSL

2. Navigate to the script directory
3. Run python Script using " python3 matrix.py " command.

Algorithm Verification

-**Correctness** : The methodology used to verify the correctness of the matrix multiplication algorithm are:

Input Validation

1.The algorithm first checks whether the matrices A and B have compatible dimensions for multiplication. Specifically, the number of columns of matrix A (`COLS_A`) must equal the number of rows of matrix B (`ROWS_B`), as per the matrix multiplication rule. If this condition is not met, an error message is displayed, and no further operations are carried out.

Manual and Automated test

To verify the correctness of the algorithm, I used both **manual test cases** and **automated validation**.

Manual Test Cases: I tested the algorithm with simple 2x2 matrices and known result. By comparing the output from the program with the expected output, I verified that the algorithm computes the correct result.

Automated Validation: I also used an online matrix multiplication calculator to cross-check results for various test cases, ensuring that the output produced by the algorithm matched the expected results.

Output verification

Finally, the algorithm prints the matrices and the result, allowing me to visually confirm that the calculations are correct by comparing the expected result with the displayed output. So, by passing through this entire process and testing the algorithm with various inputs and edge cases, I am pretty sure that the matrix multiplication results are correct.

Performance Analysis

-**Execution Times**:

The table below illustrate the comparison between C and python language execution time for matrix multiplication program.

Matrix size	C execution time (s)	Python execution time (s)
2 x 2	0.001670	141.591794
3 x 3	0.002418	121.148879
4 x 4	0.004694	181.454444

Analysis:

The results show that C significantly outperforms Python in matrix multiplication, with execution times in milliseconds compared to Python's excessively high times (over 100 seconds). This is due to C being a **Compiled language** with *direct memory access* and *minimal overhead*, whereas Python is **Interpreted language**, *dynamically manages memory*, and has *higher computational overhead*. The unexpectedly slow Python performance suggests an inefficient implementation, likely using raw loops instead of optimized libraries. To improve Python's speed, **NumPy** should be used, as it leverages C-based optimizations for efficient numerical computations.

Conclusion

This project enhanced my understanding of *WSL, Markdown, and Programming Efficiency*. Using the command line, I learned basic *Linux operations* like compiling and running programs. Writing the report in *Markdown* improved my ability to create structured documentation. The performance comparison showed that *C is much faster than Python* due to its compiled nature and direct memory access, while Python's interpreted execution caused significant delays. I also realized the importance of *Optimized libraries like NumPy* for better performance. Overall, this project enhanced my understanding of programming, execution models, and technical documentation, providing a strong foundation for future development and optimization tasks.

References

[1]https://learn.microsoft.com/en-us/windows/wsl/basic-commands

[2]https://phoenixnap.com/kb/linux-commands-cheat-sheet

[3]https://www.nano-editor.org/docs.php

[4]https://youtu.be/g2PU–TctAM?si=tMhQW66OVgwN9KS3

[5]https://www.geeksforgeeks.org/time-function-in-c/

[6]https://www.programiz.com/cprogramming/examples/matrix-multiplication

[7]https://gcc.gnu.org/onlinedocs/gcc-14.2.0/gcc/Standards.html#C-Language

[8]https://matrix.reshish.com/multiplication.php

[9]https://www.tutorialspoint.com/why-does-c-code-run-faster-than-python-s

[10]https://www.w3schools.com/python/numpy/numpy_intro.asp