Design Analysis and Algorithem, Lab Assignment 4

Bhuvana Kanakam - SE21UCSE035

September 29, 2023

Question

Assignment Implementing Strassen's Matrix Multiplication Algorithm

Approach

1. Implement the standard matrix multiplication and Strassen's matrix multiplication
2. Test your implementations by generating random matrices and comparing the results of standard matrix multiply and strassen matrix multiply to ensure they produce the same results.
3. Measure the execution time of both matrix multiplication methods for increasingly larger matrix sizes (e.g., n = 2, 4, 8, 16, ...), and create a graph that shows the time complexity of each algorithm as a function of the matrix size.

Answer

Below is the Python code :

```
import random
import time
import matplotlib.pyplot as plt

def standard_matrix_multiply(A, B):
    if len(A[0]) != len(B):
        raise ValueError("Matrix dimensions are not compatible for multiplication")
    C = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
    return C

def divide_matrix(matrix):
    n = len(matrix)
    half = n // 2
    a11 = [matrix[i][:half] for i in range(half)]
    a12 = [matrix[i][half:] for i in range(half)]
    a21 = [matrix[i][:half] for i in range(half, n)]
    a22 = [matrix[i][half:] for i in range(half, n)]
    return a11, a12, a21, a22

def add_matrices(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def subtract_matrices(A, B):
    return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in range(len(A))]
```

```
def combine_matrices(c11, c12, c21, c22):
    n = len(c11)
    result = [[0 for _ in range(2 * n)] for _ in range(2 * n)]
    for i in range(n):
        for j in range(n):
            result[i][j] = c11[i][j]
            result[i][j + n] = c12[i][j]
            result[i + n][j] = c21[i][j]
            result[i + n][j + n] = c22[i][j]
    return result

def strassen_matrix_multiply(A, B):
    if len(A) == 1 and len(B) == 1:
        return [[A[0][0] * B[0][0]]]
    a11, a12, a21, a22 = divide_matrix(A)
    b11, b12, b21, b22 = divide_matrix(B)
    p1 = strassen_matrix_multiply(a11, subtract_matrices(b12, b22))
    p2 = strassen_matrix_multiply(add_matrices(a11, a12), b22)
    p3 = strassen_matrix_multiply(add_matrices(a21, a22), b11)
    p4 = strassen_matrix_multiply(a22, subtract_matrices(b21, b11))
    p5 = strassen_matrix_multiply(add_matrices(a11, a22), add_matrices(b11, b22))
    p6 = strassen_matrix_multiply(subtract_matrices(a12, a22), add_matrices(b21, b22))
    p7 = strassen_matrix_multiply(subtract_matrices(a11, a21), add_matrices(b11, b12))
    c11 = subtract_matrices(add_matrices(add_matrices(p5, p4), p6), p2)
    c12 = add_matrices(p1, p2)
    c21 = add_matrices(p3, p4)
    c22 = subtract_matrices(subtract_matrices(add_matrices(p5, p1), p3), p7)
    C = combine_matrices(c11, c12, c21, c22)
    return C

matrix_sizes = [2 ** i for i in range(1, 10)]

for n in matrix_sizes:
    A = [[random.randint(1, 10) for _ in range(n)] for _ in range(n)]
    B = [[random.randint(1, 10) for _ in range(n)] for _ in range(n)]
    standard_result = standard_matrix_multiply(A, B)
    strassen_result = strassen_matrix_multiply(A, B)

    if standard_result == strassen_result:
        print(f"Matrix size {n}x{n}: Results are the same.")
    else:
        print(f"Matrix size {n}x{n}: Results are different.")

standard_times = []
strassen_times = []

for n in matrix_sizes:
    A = [[random.randint(1, 10) for _ in range(n)] for _ in range(n)]
    B = [[random.randint(1, 10) for _ in range(n)] for _ in range(n)]

    start_time = time.time()
    standard_matrix_multiply(A, B)
    standard_times.append(time.time() - start_time)

    start_time = time.time()
    strassen_matrix_multiply(A, B)
    strassen_times.append(time.time() - start_time)
```

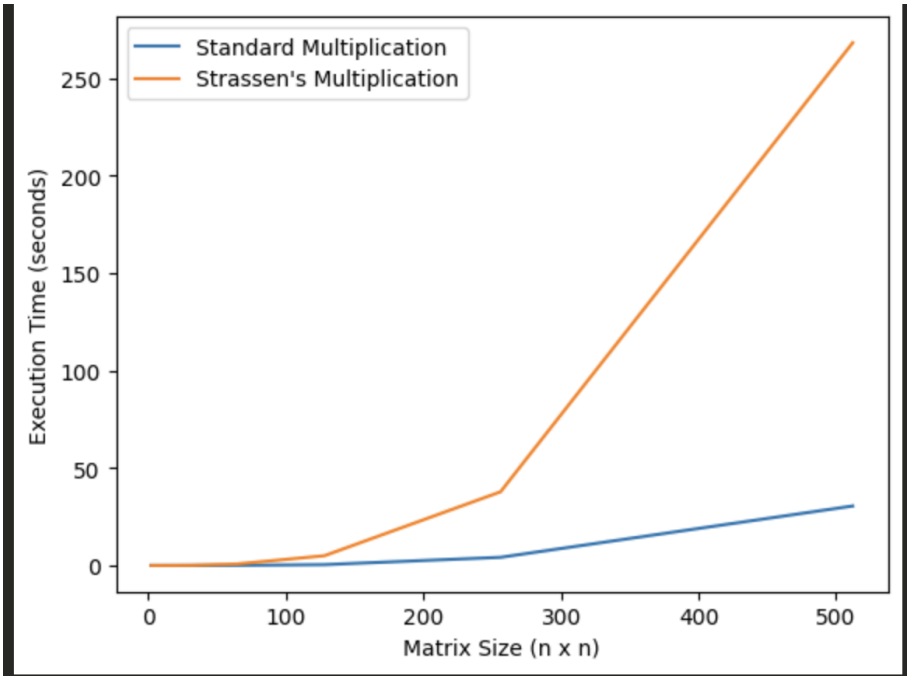

Figure 1: Enter Caption

```
plt.plot(matrix_sizes, standard_times, label='Standard Multiplication')
plt.plot(matrix_sizes, strassen_times, label="Strassen's Multiplication")
plt.xlabel('Matrix Size (n x n)')
plt.ylabel('Execution Time (seconds)')
plt.legend()
plt.show()
```

Explanation

Strassen's Matrix Multiplication

Since matrix multiplications are more expensive than matrix additions O(n3) versus O(n2) Based on this logic Volker Strassen defined the approach for matrix multiplication faster than O(n3) in 1969

- Recursively divides the matrices into smaller submatrices and uses a set of recursive formulas to compute the result

- Input: Two square matrix of size nXn

- Output: Multiplication of two matrix