Detailed analysis and design of the Strassen's Matrix Algorithm

DAA Assignment 4 - Group 21

Divyesh Rana
IIT2019063

Akash Deep
IIT2019064

Gopal Pedada
IIT2019065

***Abstract*—This paper contains the design and the detailed analysis of the algorithm used to solve the following problem: Implement Strassen's matrix algorithm using divide and conquer.**

I. Introduction

Strassen's Algorithm is used to compute multiplication of two matrices. There is a condition for the matrix multiplication : If $A_{m\times n}$ and $B_{p\times q}$ then $C = A \times B$ is only possible when $p = q$ and the resultant matrix C will be the matrix with size of $m \times q$. Strassen's algorithm is used for multiplication of square matrix. This algorithm is based on divide and conquer technique. Strassen's algorithm is basically an improvement in standard divide and conquer approach to compute multiplication of two matrices.

This report further contains -
II. Algorithmic Design
III. Algorithm Analysis
IV. Experimental Study
V. Conclusion

II. Algorithmic Design

A. Naive approach to multiply matrix

Multiplying two matrices A and B both of them having dimensions $n \times n$, and matrix $C = A \times B$ will be also of $n \times n$ dimensions.

$$C = A \times B$$

Run a loop for each row in matrix A with index i and inside the loop, we will run another loop for each column in matrix B with index j, inside that loop, run a loop with index k to traverse through the elements of each columns of matrix A in ith row, and to traverse through the elements of each rows of matrix B in jth column, this traversal can be done with a single loop $(A[i][k], B[k][j])$.Multiply each column elements of matrix A and each row elements of matrix B and store the summation of multiplication in matrix C. i.e $C[i][j] = A[i][k] \times B[k][j]$.

In other words,

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

for $i = 0$ to $n - 1$ and $j = 0$ to $n - 1$.

B. Strassen's Algorithm

Note that, there are requirements for strassen's algorithm to work:
1. Both input matrices should be size of $n \times n$.
2. n should be power of 2.

1) Matrix multiplication using divide and conquer

Divide matrices A and B in 4 submatrices of size $n/2\times n/2$.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

then,

$$C = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Use recursion to calculate values of $A_{11}B_{11} + A_{12}B_{21}$, $A_{11}B_{12} + A_{12}B_{22}$, $A_{21}B_{11} + A_{22}B_{21}$ and $A_{21}B_{12} + A_{22}B_{22}$.

It can be seen that we do 8 multiplications for matrices of size $n/2\times n/2$ and 4 additions. Addition of two matrices takes $O(n^2)$ time. So the time complexity is $T(n) = 8T(n/2) + O(n^2)$, also from this recurrence relation the time complexity of this algorithm will be $O(n^3)$ which is same as navie approach.

2) Strassen's Algortihm

Strassen's algorithm makes use of the same divide and conquer approach as above, but instead uses only 7 recursive calls rather than 8.This is enough to reduce the time complexity to sub-cubic time.This is algorithm is improvement of divide and conquer algorithm for matrix multiplication.

After getting submatrices, Compute seven matrices shown below recursively.

$P_1 = A_{11} \times (B_{12} - B_{22})$
$P_2 = (A_{11} + A_{12}) \times B_{22}$
$P_3 = (A_{21} + A_{22}) \times B_{11}$
$P_4 = A_{22} \times (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$
$P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$
$P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})$

If C is $A \times B$ and $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$ then

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

Run simple loops to combine submatrices in one matrix.

Note that Addition and Subtraction in Matrix works differently than numbers, run a loop with index i for elements of each row and run a loop with index j for elements of each column and compute addition and subtraction indices wise, i.e $C[i][j] = A[i][j] + B[i][j]$ and $C = A + B$ similarly for subtraction.

Algorithm 1: Naive approach

```
  Input: Integer $n$ and two matrices
  Output: multiplication of two matrices
  Require: $n \geq 0$
1 Function Multiplication($A$,$B$,$n$):
2     for $i \leftarrow 0$ to $n-1$ do
3         for $j \leftarrow 0$ to $n-1$ do
4             $C[i][j] \leftarrow 0$
5             for $k \leftarrow 0$ to $n-1$ do
6                 $C[i][j] \leftarrow C[i][j] + (A[i][k] \times B[k][j])$
7     return C
```

Algorithm 2: Subtraction of Matrices

```
  Input: Integer $n$ and two matrices
  Output: Subtraction of two matrices
  Require: $n \geq 0$
1 Function Sub($A$,$B$,$n$):
2     for $i \leftarrow 0$ to $n-1$ do
3         for $j \leftarrow 0$ to $n-1$ do
4             $A[i][j] \leftarrow A[i][j] - B[i][j]$
5     return A
```

Algorithm 3: Addition of Matrices

```
  Input: Integer $n$ and two matrices
  Output: Addition of two matrices
  Require: $n \geq 0$
1 Function Add($A$,$B$,$n$):
2     for $i \leftarrow 0$ to $n-1$ do
3         for $j \leftarrow 0$ to $n-1$ do
4             $A[i][j] \leftarrow A[i][j] + B[i][j]$
5     return A
```

Algorithm 4: Divide and conquer approach

```
   Input: Integer $n$ and two matrices
   Output: multiplication of two matrices
   Require: $n \geq 0$ and $n$ is power of 2
1  Function Multi($A$,$B$,$n$):
2      if $n = 1$ then
3          $C[0][0] \leftarrow A[0][0] \times B[0][0]$
4      else
5          for $i \leftarrow 0$ to $(n/2) - 1$ do
6              for $j \leftarrow 0$ to $(n/2) - 1$ do
                   /* Dividing A into
                      submartrices                */
7                  $A_{11}[i][j] \leftarrow A[i][j]$
8                  $A_{12}[i][j] \leftarrow A[i][j + (n/2)]$
9                  $A_{21}[i][j] \leftarrow A[i + (n/2)][j]$
10                 $A_{22}[i][j] \leftarrow A[i + (n/2)][j + (n/2)]$
                   /* Dividing B into
                      submartrices                */
11                 $B_{11}[i][j] \leftarrow B[i][j]$
12                 $B_{12}[i][j] \leftarrow B[i][j + (n/2)]$
13                 $B_{21}[i][j] \leftarrow B[i + (n/2)][j]$
14                 $B_{22}[i][j] \leftarrow B[i + (n/2)][j + (n/2)]$
15         $k \leftarrow (n/2)$
16         $P_1 \leftarrow Multi(A_{11}, Sub(B_{12}, B_{22}, k), k)$
17         $P_2 \leftarrow Multi(Add(A_{11}, A_{12}, k), B_{22}, k)$
18         $P_3 \leftarrow Multi(Add(A_{21}, A_{22}, k), B_{11}, k)$
19         $P_4 \leftarrow Multi(A_{22}, Sub(B_{21}, B_{11}, k), k)$
20         $P_5 \leftarrow$
             $Multi(Add(A_{11}, A_{22}, k), Add(B_{11}, B_{22}, k), k)$
21         $P_6 \leftarrow$
             $Multi(Sub(A_{12}, A_{22}, k), Add(B_{21}, B_{22}, k), k)$
22         $P_7 \leftarrow$
             $Multi(Sub(A_{11}, A_{21}, k), Add(B_{11}, B_{12}, k), k)$
           /* Getting C                               */
23         $C_{11} \leftarrow Sub(Add(Add(P_5, P_4, k), P_6, k), P_2, k)$
24         $C_{12} \leftarrow Add(P_1, P_2, k)$
25         $C_{21} \leftarrow Add(P_3, P_4, k)$
26         $C_{22} \leftarrow Sub(Add(P_5, P_1, k), Add(P_3, P_7, k), k)$
           /* Combining submatrices in C  */
27         for $i \leftarrow 0$ to $(n/2) - 1$ do
28             for $j \leftarrow 0$ to $(n/2) - 1$ do
29                 $C[i][j] \leftarrow C_{11}[i][j]$
30                 $C[i][j + (n/2)] \leftarrow C_{12}[i][j]$
31                 $C[i + (n/2)][j] \leftarrow C_{21}[i][j]$
32                 $C[i + (n/2)][j + (n/2)] \leftarrow C_{22}[i][j]$
33     return C
```

III. Algorithm Analysis

A. Time Complexity:

Approach 1: Here, we assume that integer operations take O(1) time. Three for loops are used in this code snippet ,nested one into other.Therefore, time complexity of this algorithm is $O(n^3)$.

Approach 2: Here, addition and subtraction of two matrices takes $O(n^2)$ time. In this algorithm, we do 7 multiplications $(P_{1...7})$ for matrices and some addition and subtraction so the recurrence relation would be :

$T(n) = 7T(n/2) + O(n^2)$

Using Master Theorem, Time complexity for this recurrence relation will be $O(n^{log_2 7})$, which is sub-cubic complexity $(O(n^{2.8}))$.

$T_{avg} = O(n^{log_2 7})$

$T_{best} = O(1)$

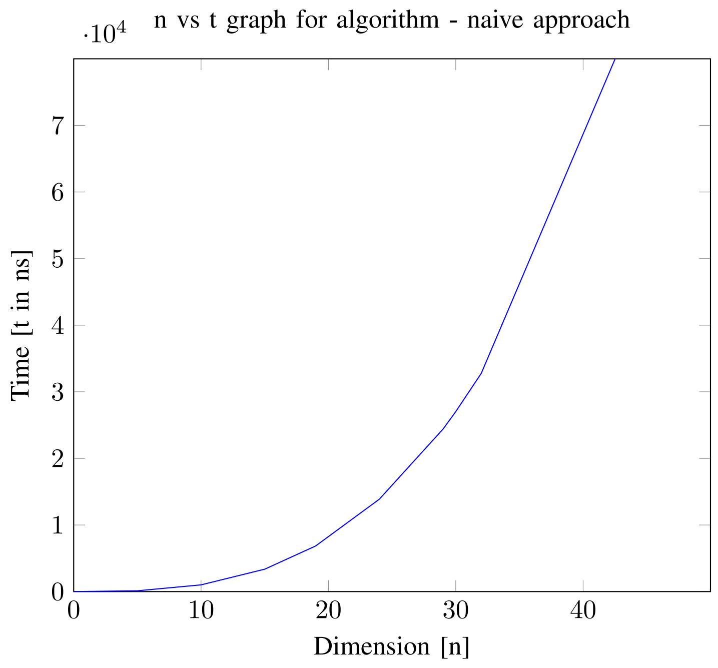
B. Space Complexity

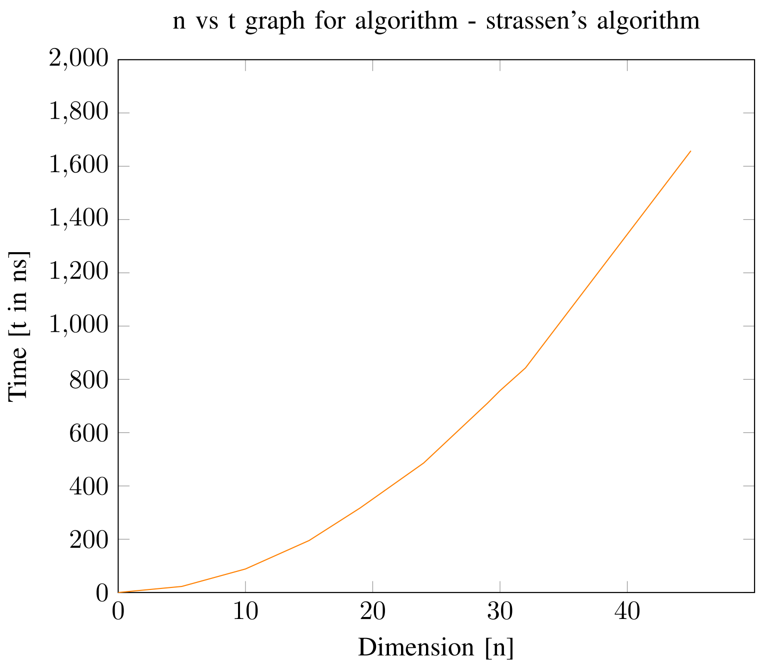
The space complexity for both algorithms is $O(n^2)$.

IV. Experimental Analysis

Graph - 1 : It can be clearly seen that first graph has $O(n^3)$ behaviour.

Graph - 2 : The second graph has something like sub-cubic behaviour.





V. Conclusion

For matrix with higher dimension, Strassen's method to compute multiplication of matrices is significantly faster than naive method.

VI. Reference

1) https://www.cs.cmu.edu/ 15451-f19/LectureNotes/lec01-strassen.pdf
2) https://en.wikipedia.org/wiki/Strassen_algorithm