



ET3272: Design and Analysis of Algorithms

Name of the student: Aditya Jannawar

Roll No.: 02

Div.: A Batch: 1

Date of performance: 13/02/2023

Experiment No. 5

Title: Matrix Multiplication using Divide and Conquer

Theory/Description of the Problem Statement:

Matrix multiplication is a fundamental operation in linear algebra and has many applications in various fields. The standard algorithm for multiplying two matrices is the naive algorithm, which has a time complexity of $O(n^3)$, where n is the dimension of the matrix. In this answer, we will describe an alternative approach, known as the Divide and Conquer algorithm, which has a time complexity of $O(n^3 \log_2(7))$.

The Divide and Conquer algorithm for matrix multiplication is based on the idea of dividing the matrices into smaller sub-matrices and recursively multiplying them.

Algorithm:

Algorithm : Matrix Multiplication using Divide and Conquer

1.Divide each matrix into four sub-matrices, as follows:

- **A11, A12, A21, A22 from matrix A**
 - **B11, B12, B21, B22 from matrix B**
-

2.Compute seven products recursively:

- **$P1 = A11*(B12-B22)$**
- **$P2 = (A11+A12)*B22$**
- **$P3 = (A21+A22)*B11$**
- **$P4 = A22*(B21-B11)$**
- **$P5 = (A11+A22)*(B11+B22)$**



Department of Electronics & Telecommunication Engineering

- $P6 = (A_{12}-A_{22})*(B_{21}+B_{22})$
- $P7 = (A_{11}-A_{21})*(B_{11}+B_{12})$

3. Compute the resulting sub-matrices

- $C_{11} = P5+P4-P2+P6$
- $C_{12} = P1+P2$
- $C_{21} = P3+P4$
- $C_{22} = P5+P1-P3-P7$

4. Concatenate the sub-matrices to form the resulting matrix C.

The algorithm takes advantage of the fact that the product of two $n/2$ by $n/2$ matrices can be computed using only seven products of $n/2$ by $n/2$ matrices, as shown in steps 2 and 3. This results in a time complexity of $O(n^{\log_2(7)})$, which is faster than the naive algorithm for $n > 2$.

Algorithm MATRIX_MULTIPLICATION(A, B, C)

// A and B are input matrices of size $n \times n$

// C is the output matrix of size $n \times n$

for $i \leftarrow 1$ to n do

 for $j \leftarrow 1$ to n do

$C[i][j] \leftarrow 0$

 for $k \leftarrow 1$ to n do

$C[i][j] \leftarrow C[i][j] + A[i][k]*B[k][j]$

 end

 end

end



Analysis of the Algorithm

Time Complexity

Addition and Subtraction of two matrices takes $O(N^2)$ time. So time complexity can be written as

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

$O(N^{\log_2 7})$ which is approximately $O(N^{2.8074})$

Generally Strassen's Method is not preferred for practical applications because of the constants used in Strassen's method are high and for a typical application Naive method works better. For Sparse matrices, there are better methods especially designed for them. The submatrices in recursion take extra space. Because of the limited precision of computer arithmetic on non integer values, larger errors accumulate in Strassen's algorithm than in Naive Method

Experiment and result:

Code:

```
#include <bits/stdc++.h>
using namespace std;

#define ROW_1 4
#define COL_1 4

#define ROW_2 4
#define COL_2 4

void print(string display, vector<vector<int> > matrix,
           int start_row, int start_column, int end_row,
           int end_column)
```

Department of Electronics & Telecommunication Engineering

```
{
    cout << endl << display << " =>" << endl;
    for (int i = start_row; i <= end_row; i++) {
        for (int j = start_column; j <= end_column; j++) {
            cout << setw(10);
            cout << matrix[i][j];
        }
        cout << endl;
    }
    cout << endl;
    return;
}

vector<vector<int> >
add_matrix(vector<vector<int> > matrix_A,
           vector<vector<int> > matrix_B, int split_index,
           int multiplier = 1)
{
    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++)
            matrix_A[i][j]
                = matrix_A[i][j]
                + (multiplier * matrix_B[i][j]);
    return matrix_A;
}

vector<vector<int> >
multiply_matrix(vector<vector<int> > matrix_A,
               vector<vector<int> > matrix_B)
{
    int col_1 = matrix_A[0].size();
    int row_1 = matrix_A.size();
    int col_2 = matrix_B[0].size();
    int row_2 = matrix_B.size();

    if (col_1 != row_2) {
        cout << "\nError: The number of columns in Matrix "
              << "A must be equal to the number of rows in "
              << "Matrix B\n";
        return {};
    }

    vector<int> result_matrix_row(col_2, 0);
    vector<vector<int> > result_matrix(row_1,
```

Department of Electronics & Telecommunication Engineering

```
result_matrix_row);

if (col_1 == 1)
    result_matrix[0][0]
        = matrix_A[0][0] * matrix_B[0][0];
else {
    int split_index = col_1 / 2;

    vector<int> row_vector(split_index, 0);

    vector<vector<int>> > a00(split_index, row_vector);
    vector<vector<int>> > a01(split_index, row_vector);
    vector<vector<int>> > a10(split_index, row_vector);
    vector<vector<int>> > a11(split_index, row_vector);
    vector<vector<int>> > b00(split_index, row_vector);
    vector<vector<int>> > b01(split_index, row_vector);
    vector<vector<int>> > b10(split_index, row_vector);
    vector<vector<int>> > b11(split_index, row_vector);

    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++) {
            a00[i][j] = matrix_A[i][j];
            a01[i][j] = matrix_A[i][j + split_index];
            a10[i][j] = matrix_A[split_index + i][j];
            a11[i][j] = matrix_A[i + split_index]
                [j + split_index];
            b00[i][j] = matrix_B[i][j];
            b01[i][j] = matrix_B[i][j + split_index];
            b10[i][j] = matrix_B[split_index + i][j];
            b11[i][j] = matrix_B[i + split_index]
                [j + split_index];
        }

    vector<vector<int>> > p(multiply_matrix(
        a00, add_matrix(b01, b11, split_index, -1)));
    vector<vector<int>> > q(multiply_matrix(
        add_matrix(a00, a01, split_index), b11));
    vector<vector<int>> > r(multiply_matrix(
        add_matrix(a10, a11, split_index), b00));
    vector<vector<int>> > s(multiply_matrix(
        a11, add_matrix(b10, b00, split_index, -1)));
    vector<vector<int>> > t(multiply_matrix(
        add_matrix(a00, a11, split_index),
        add_matrix(b00, b11, split_index)));
}
```

Department of Electronics & Telecommunication Engineering

```
vector<vector<int> > u(multiply_matrix(
    add_matrix(a01, a11, split_index, -1),
    add_matrix(b10, b11, split_index)));
vector<vector<int> > v(multiply_matrix(
    add_matrix(a00, a10, split_index, -1),
    add_matrix(b00, b01, split_index)));

vector<vector<int> > result_matrix_00(add_matrix(
    add_matrix(add_matrix(t, s, split_index), u,
        split_index),
    q, split_index, -1));
vector<vector<int> > result_matrix_01(
    add_matrix(p, q, split_index));
vector<vector<int> > result_matrix_10(
    add_matrix(r, s, split_index));
vector<vector<int> > result_matrix_11(add_matrix(
    add_matrix(add_matrix(t, p, split_index), r,
        split_index, -1),
    v, split_index, -1));

for (auto i = 0; i < split_index; i++)
    for (auto j = 0; j < split_index; j++) {
        result_matrix[i][j]
            = result_matrix_00[i][j];
        result_matrix[i][j + split_index]
            = result_matrix_01[i][j];
        result_matrix[split_index + i][j]
            = result_matrix_10[i][j];
        result_matrix[i + split_index]
            [j + split_index]
            = result_matrix_11[i][j];
    }

a00.clear();
a01.clear();
a10.clear();
a11.clear();
b00.clear();
b01.clear();
b10.clear();
b11.clear();
p.clear();
q.clear();
r.clear();
```



Department of Electronics & Telecommunication Engineering

```
s.clear();
t.clear();
u.clear();
v.clear();
result_matrix_00.clear();
result_matrix_01.clear();
result_matrix_10.clear();
result_matrix_11.clear();
}
return result_matrix;
}

int main()
{
    vector<vector<int> > matrix_A = { { 1, 1, 1, 1 },
                                       { 2, 2, 2, 2 },
                                       { 3, 3, 3, 3 },
                                       { 2, 2, 2, 2 } };

    print("Array A", matrix_A, 0, 0, ROW_1 - 1, COL_1 - 1);

    vector<vector<int> > matrix_B = { { 1, 1, 1, 1 },
                                       { 2, 2, 2, 2 },
                                       { 3, 3, 3, 3 },
                                       { 2, 2, 2, 2 } };

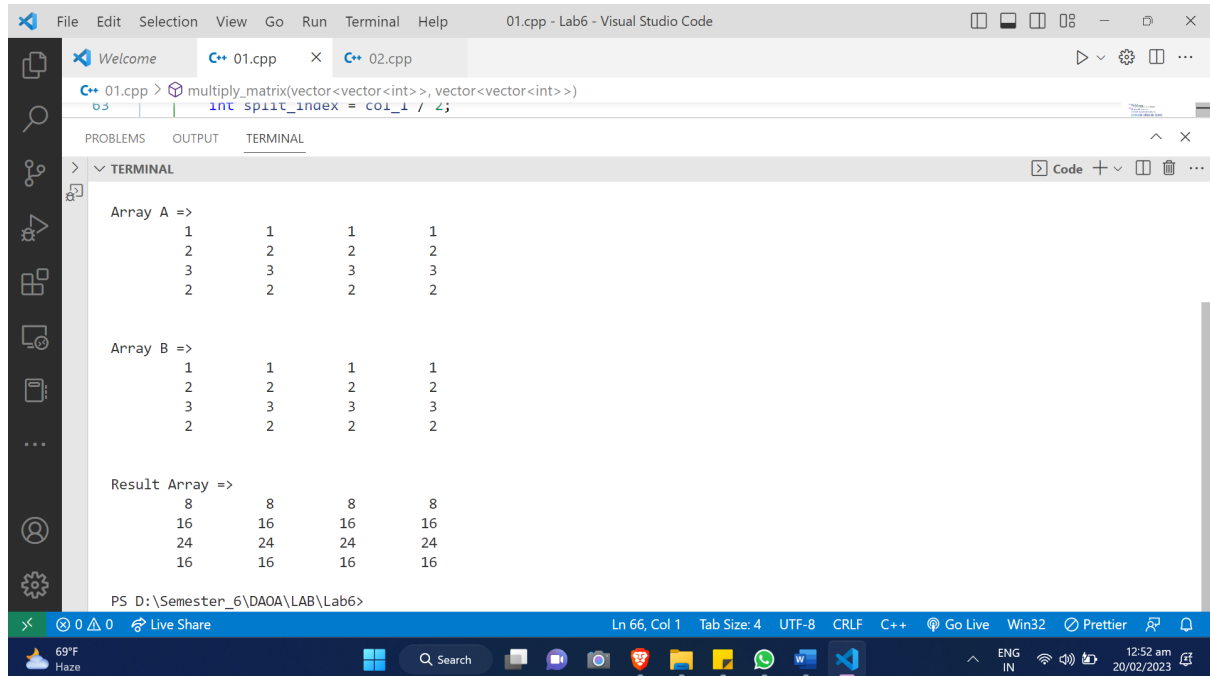
    print("Array B", matrix_B, 0, 0, ROW_2 - 1, COL_2 - 1);

    vector<vector<int> > result_matrix(
        multiply_matrix(matrix_A, matrix_B));

    print("Result Array", result_matrix, 0, 0, ROW_1 - 1,
        COL_2 - 1);
}
```

Output:

Department of Electronics & Telecommunication Engineering



```
File Edit Selection View Go Run Terminal Help 01.cpp - Lab6 - Visual Studio Code
Welcome 01.cpp x 02.cpp
01.cpp > multiply_matrix(vector<vector<int>>, vector<vector<int>>)>
int split_index = col_1 / 2;

PROBLEMS OUTPUT TERMINAL
TERMINAL
Array A =>
1 1 1 1
2 2 2 2
3 3 3 3
2 2 2 2

Array B =>
1 1 1 1
2 2 2 2
3 3 3 3
2 2 2 2

Result Array =>
8 8 8 8
16 16 16 16
24 24 24 24
16 16 16 16

PS D:\Semester_6\DAOA\LAB\Lab6>
```

Conclusions:

The `matrix_multiply_divide_conquer` function takes two matrices A and B as input, and recursively divides each matrix into four sub-matrices until they become 1x1 matrices. Then, it multiplies these matrices and combines the results using matrix addition and subtraction to obtain the final result.