Experiment 2- Use Divide and Conquer approach to implement Develop a code for Strassen's Matrix Multiplication and analyze it.

Learning Objective: Students should be able to perform Strassen's matrix multiplication on a given data set using divide and conquer strategy.

Tools: C/C++/Java/Python under Windows or Linux environment.

Theory: Develop a code for Strassen's matrix multiplication using divide and conquer strategy and analyze it. (Menu driven program)

Following is a simple Divide and Conquer method to multiply two square matrices.

1. Divide matrices A and B in 4 sub-matrices of size N/2 x N/2 as shown in the below diagram.
2. Calculate following values recursively. ae + bg, af + bh, ce + dg and cf + dh.

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

Strassen suggested a divide and conquer strategy-based matrix multiplication technique that requires fewer multiplications than the traditional method. The multiplication operation is defined as follows using Strassen's method:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

C11 = P1 + P4 – P5 + P7
C12 = P3 + P5
C21 = P2 +P4
C22 = P1 + P3 – P2 + P6

Where,
P1 = (A11 + A22) * (B11 + B22)
P2 = (A21 + A22) * B11
P3 = A11 * (B12 – B22)
P4 = A22 * (B21 – B11)
P5 = (A11 + A12) * B22
P6 = (A21 – A11) * (B11 + B12)P7 = (A12 – A22) * (B21 + B22)
Let us check if it is the same as the conventional approach.
C12 = P3 + P5
= A11 * (B12 – B22) + (A11 + A12) * B22
= A11 * B12 – A11 * B22 + A11 * B22 + A12 * B22
= A11 * B12 + A12 * B22
This is the same as C12 derived using the conventional approach. Similarly we can derive all Cij for strassen's matrix multiplication.

ALGORITHM:

```
Algorithm STRESSEN_MAT_MUL (int *A, int *B, int *C, int n)
// A and B are input matrices
// C is the output matrix
// All matrices are of size n x n


if n == 1 then
   *C = *C + (*A) * (*B)
else
   STRESSEN_MAT_MUL (A, B, C, n/4)
   STRESSEN_MAT_MUL (A, B + (n/4), C + (n/4), n/4)
   STRESSEN_MAT_MUL (A + 2 * (n/4), B, C + 2 * (n/4), n/4)
   STRESSEN_MAT_MUL (A + 2 * (n/4), B + (n/4), C + 3 * (n/4), n/4)
   STRESSEN_MAT_MUL (A + (n/4), B + 2 * (n/4), C, n/4)
   STRESSEN_MAT_MUL (A + (n/4), B + 3 * (n/4), C + (n/4), n/4)
   STRESSEN_MAT_MUL (A + 3 * (n/4), B + 2 * (n/4), C + 2 * (n/4), n/4)
   STRESSEN_MAT_MUL (A + 3 * (n/4), B + 3 * (n/4), C + 3 * (n/4), n/4)
end
```

ADVANTAGE of Divide & Conquer Algorithm:

1. The difficult problem can be solved easily.

2. It divides the entire problem into subproblems thus it can be solved parallelly ensuring multiprocessing
3. Efficiently uses cache memory without occupying much space
4. Reduces time complexity of the problem

DISADVANTAGES of Divide & Conquer Algorithm:

1. It involves recursion which is sometimes slow
2. Efficiency depends on the implementation of logic
3. It may crash the system if the recursion is performed rigorously

ANALYSIS:

Time Complexity:
Addition and Subtraction of two matrices takes O(N2) time. So time complexity can be written as
T(N) = 7T(N/2) + O(N2)
From Master's Theorem, time complexity of above method is
O(NLog7) which is approximately O(N2.8074)

Space Complexity:
A new matrix is used to store the result of the multiplication. So, the space complexity is O(N^2).

APPLICATION:

Generally, Strassen’s Method is not preferred for practical applications for the following reasons.

- 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 noninteger values, larger errors accumulate in Strassen’s algorithm than in Naive Method

Learning Outcome: The student should have the ability to
LO1: understand and implement divide and conquer strategy.
LO2: perform matrix multiplication of Strassen's matrix using divide and conquer.

Course Outcomes: Upon completion of the course students will be able to apply divide and conquer strategy to given data.

CODE:

```
def strassen_2x2_recursive(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        a11, a12, a21, a22 = split_matrix(A)
        b11, b12, b21, b22 = split_matrix(B)

        # Calculate the following values recursively
        p1 = strassen_2x2_recursive(add_matrices(a11, a22), add_matrices(b11, b22))
        p2 = strassen_2x2_recursive(add_matrices(a21, a22), b11)
        p3 = strassen_2x2_recursive(a11, subtract_matrices(b12, b22))
        p4 = strassen_2x2_recursive(a22, subtract_matrices(b21, b11))
        p5 = strassen_2x2_recursive(add_matrices(a11, a12), b22)
        p6 = strassen_2x2_recursive(subtract_matrices(a21, a11), add_matrices(b11, b12))
        p7 = strassen_2x2_recursive(subtract_matrices(a12, a22), add_matrices(b21, b22))

        # Combine the results into a 2x2 matrix
        c11 = subtract_matrices(add_matrices(add_matrices(p1, p4), p7), p5)
        c12 = add_matrices(p3, p5)
        c21 = add_matrices(p2, p4)
        c22 = subtract_matrices(add_matrices(add_matrices(p1, p3), p6), p2)

        C = [[0, 0], [0, 0]]
        for i in range(n // 2):
            for j in range(n // 2):
                C[i][j] = c11[i][j]
                C[i][j + n // 2] = c12[i][j]
                C[i + n // 2][j] = c21[i][j]
                C[i + n // 2][j + n // 2] = c22[i][j]

        return C

def split_matrix(A):
    n = len(A)
    m = n // 2
```

```
    a11 = [[0] * m for i in range(m)]
    a12 = [[0] * m for i in range(m)]
    a21 = [[0] * m for idef strassen_2x2_recursive(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        a11, a12, a21, a22 = split_matrix(A)
        b11, b12, b21, b22 = split_matrix(B)

        # Calculate the following values recursively
        p1 = strassen_2x2_recursive(add_matrices(a11, a22), add_matrices(b11, b22))
        p2 = strassen_2x2_recursive(add_matrices(a21, a22), b11)
        p3 = strassen_2x2_recursive(a11, subtract_matrices(b12, b22))
        p4 = strassen_2x2_recursive(a22, subtract_matrices(b21, b11))
        p5 = strassen_2x2_recursive(add_matrices(a11, a12), b22)
        p6 = strassen_2x2_recursive(subtract_matrices(a21, a11), add_matrices(b11, b12))
        p7 = strassen_2x2_recursive(subtract_matrices(a12, a22), add_matrices(b21, b22))

        # Combine the results into a 2x2 matrix
        c11 = subtract_matrices(add_matrices(add_matrices(p1, p4), p7), p5)
        c12 = add_matrices(p3, p5)
        c21 = add_matrices(p2, p4)
        c22 = subtract_matrices(add_matrices(add_matrices(p1, p3), p6), p2)

        C = [[0, 0], [0, 0]]
        for i in range(n // 2):
            for j in range(n // 2):
                C[i][j] = c11[i][j]
                C[i][j + n // 2] = c12[i][j]
                C[i + n // 2][j] = c21[i][j]
                C[i + n // 2][j + n // 2] = c22[i][j]

        return C

def split_matrix(A):
    n = len(A)
    m = n // 2
    a11 = [[0] * m for i in range(m)]
```

```
    a12 = [[0] * m for i in range(m)]
    a21 = [[0] * m for i in range(m)]
    a22 = [[0] * m for i in range(m)]
    for i in range(m):
        for j in range(m):
            a11[i][j] = A[i][j]
            a12[i][j] = A[i][j + m]
            a21[i][j] = A[i + m][j]
            a22[i][j] = A[i + m][j + m]
    return a11, a12, a21, a22

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

def subtract_matrices(A, B):
    n = len(A)
    C = [[0] * n for i in range(n)]
    for i in range(n):
        for j in range(n):
            C[i][j] = A[i][j] - B[i][j]
    return C
a= [[12,1],[23,80]]
b= [[56,14],[90,18]]
print(strassen_2x2_recursive(a,b))
```

OUTPUT:

[[762, 186], [8488, 1762]]

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				