Lab Manual

Course Title: Computer Algorithm

Department of CSE, AMUST

Lab Topic: Matrix Multiplication Using Naive Approach

Objective

To implement matrix multiplication using the naive (brute-force) approach and analyze its time complexity.

Theory

Matrix multiplication is a binary operation that produces a matrix from two matrices. If A is a matrix of size m x n and B is a matrix of size n x p, then their product C = A × B is a matrix of size m x p.

Naive Matrix Multiplication Algorithm:
This approach uses three nested loops to compute the dot product between rows of A and columns of B.
The formula for each element C[i][j] is:

$C[i][j] = \Sigma (A[i][k] * B[k][j])$ for k from 0 to n-1

Time Complexity:
- Time Complexity: O(m × n × p)
- For square matrices of size n × n, this becomes $O(n^3)$.

Algorithm Steps

1. Start
2. Input matrix A of size m x n
3. Input matrix B of size n x p
4. Initialize result matrix C of size m x p with all zeros
5. For i from 0 to m-1:
6. For j from 0 to p-1:
7. For k from 0 to n-1:

8. C[i][j] += A[i][k] * B[k][j]
9. Output matrix C
10. End

C++ Implementation

```
#include <iostream>
using namespace std;

int main() {
  int m, n, p;
  cout << "Enter rows and columns of matrix A: ";
  cin >> m >> n;
  cout << "Enter columns of matrix B: ";
  cin >> p;

  int A[m][n], B[n][p], C[m][p] = {0};

  cout << "Enter elements of matrix A:\n";
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      cin >> A[i][j];

  cout << "Enter elements of matrix B:\n";
  for (int i = 0; i < n; i++)
    for (int j = 0; j < p; j++)
      cin >> B[i][j];

  // Naive matrix multiplication
  for (int i = 0; i < m; i++)
    for (int j = 0; j < p; j++)
      for (int k = 0; k < n; k++)
        C[i][j] += A[i][k] * B[k][j];

  cout << "Resultant Matrix C:\n";
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < p; j++)
      cout << C[i][j] << " ";
    cout << endl;
  }

  return 0;
```

}

Sample Input/Output

Input:
Matrix A (2x3):
1 2 3
4 5 6

Matrix B (3x2):
7 8
9 10
11 12

Output:
Resultant Matrix C (2x2):
58 64
139 154

Tasks for Students

- Modify the program to count the number of multiplication and addition operations.

Viva Questions

- What is the time complexity of the naive matrix multiplication?
- Can we multiply a 2x3 and a 3x2 matrix? Why or why not?
- What are the limitations of the naive approach?
- Name some optimized algorithms for matrix multiplication.

Lab Topic: Matrix Multiplication Using Divide and Conquer Approach

Objective

To implement matrix multiplication using the divide and conquer approach (excluding Strassen's algorithm) and analyze its time complexity.

Theory

The divide and conquer method involves breaking down a large problem into smaller subproblems, solving them recursively, and combining their results.
For matrix multiplication, this approach recursively splits square matrices into four submatrices (quadrants) and combines the results from recursive multiplications of these submatrices.

Let A and B be n×n matrices, where n is a power of 2. A and B can be divided as follows:

A = | A11 A12 | B = | B11 B12 |
    | A21 A22 | | B21 B22 |

Then the product C = A × B is computed using:

C11 = A11×B11 + A12×B21
C12 = A11×B12 + A12×B22
C21 = A21×B11 + A22×B21
C22 = A21×B12 + A22×B22

Time Complexity:
- $T(n) = 8T(n/2) + O(n^2)$, which solves to $O(n^3)$

This is the same as the naive approach in complexity but enables more structured parallelization.

Algorithm Steps

11. Start
12. Input square matrices A and B of size n x n (n must be a power of 2)
13. If n == 1, multiply the two elements directly
14. Else, divide A and B into four (n/2) x (n/2) submatrices
15. Recursively compute:
16. C11 = A11×B11 + A12×B21
17. C12 = A11×B12 + A12×B22
18. C21 = A21×B11 + A22×B21
19. C22 = A21×B12 + A22×B22
20. Combine submatrices C11, C12, C21, C22 into result matrix C
21. Output matrix C

22. End

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<int>> Matrix;

Matrix add(const Matrix &A, const Matrix &B) {
   int n = A.size();
   Matrix C(n, vector<int>(n));
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         C[i][j] = A[i][j] + B[i][j];
   return C;
}

Matrix subtract(const Matrix &A, const Matrix &B) {
   int n = A.size();
   Matrix C(n, vector<int>(n));
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         C[i][j] = A[i][j] - B[i][j];
   return C;
}

Matrix multiply(const Matrix &A, const Matrix &B) {
   int n = A.size();
   Matrix C(n, vector<int>(n, 0));
   if (n == 1) {
      C[0][0] = A[0][0] * B[0][0];
   } else {
      int k = n / 2;
      Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k)), A21(k, vector<int>(k)), A22(k,
vector<int>(k));
      Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k)), B21(k, vector<int>(k)), B22(k,
vector<int>(k));

      for (int i = 0; i < k; i++)
         for (int j = 0; j < k; j++) {
```

```
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + k];
            A21[i][j] = A[i + k][j];
            A22[i][j] = A[i + k][j + k];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + k];
            B21[i][j] = B[i + k][j];
            B22[i][j] = B[i + k][j + k];
          }

      Matrix C11 = add(multiply(A11, B11), multiply(A12, B21));
      Matrix C12 = add(multiply(A11, B12), multiply(A12, B22));
      Matrix C21 = add(multiply(A21, B11), multiply(A22, B21));
      Matrix C22 = add(multiply(A21, B12), multiply(A22, B22));

      for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
          C[i][j] = C11[i][j];
          C[i][j + k] = C12[i][j];
          C[i + k][j] = C21[i][j];
          C[i + k][j + k] = C22[i][j];
        }
    }
    return C;
}
```

Sample Input/Output

Input:
Matrix A:
1 2
3 4

Matrix B:
5 6
7 8

Output:
Resultant Matrix C:
19 22
43 50

Tasks for Students

- Implement the divide and conquer matrix multiplication algorithm in C++.
- Test with matrices of size 2x2, 4x4, etc. (ensure sizes are powers of 2).
- Compare performance with the naive method for various sizes.
- Extend the program to pad non-power-of-2 matrices with zeros.

Viva Questions

- Explain how divide and conquer is applied in matrix multiplication.
- What is the time complexity of this approach?
- Name some optimized algorithms for matrix multiplication.

Lab Topic: Matrix Multiplication Using Strassen's Algorithm

Objective

To implement matrix multiplication using Strassen's algorithm and analyze its time complexity.

Theory

Strassen's algorithm is a divide-and-conquer algorithm that improves the time complexity of matrix multiplication from O(n^3) to approximately O(n^2.81). It reduces the number of recursive multiplications required to compute the product of two square matrices.
Given two n x n matrices A and B (where n is a power of 2), we split them into 4 (n/2) x (n/2) submatrices:

A = | A11 A12 | B = | B11 B12 |
| A21 A22 | | B21 B22 |

We compute 7 products (instead of 8):
M1 = (A11 + A22) × (B11 + B22)
M2 = (A21 + A22) × B11
M3 = A11 × (B12 - B22)
M4 = A22 × (B21 - B11)
M5 = (A11 + A12) × B22
M6 = (A21 - A11) × (B11 + B12)
M7 = (A12 - A22) × (B21 + B22)

Then the result matrix C is:
C11 = M1 + M4 - M5 + M7
C12 = M3 + M5
C21 = M2 + M4
C22 = M1 - M2 + M3 + M6

This reduces the number of recursive multiplications from 8 to 7, giving a time complexity of approximately O(n^2.81).

Algorithm Steps

23. Start
24. Input square matrices A and B of size n x n (n must be a power of 2)
25. If n == 1, multiply the elements directly
26. Else, divide A and B into four (n/2) x (n/2) submatrices
27. Compute the seven intermediate matrices M1 to M7 using Strassen's formulas

28. Compute the resulting submatrices C11, C12, C21, C22
29. Combine them to get the final result matrix C
30. Output matrix C
31. End

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<int>> Matrix;

Matrix add(const Matrix &A, const Matrix &B) {
  int n = A.size();
  Matrix C(n, vector<int>(n));
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      C[i][j] = A[i][j] + B[i][j];
  return C;
}

Matrix subtract(const Matrix &A, const Matrix &B) {
  int n = A.size();
  Matrix C(n, vector<int>(n));
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      C[i][j] = A[i][j] - B[i][j];
  return C;
}

Matrix strassen(const Matrix &A, const Matrix &B) {
  int n = A.size();
  Matrix C(n, vector<int>(n));
  if (n == 1) {
    C[0][0] = A[0][0] * B[0][0];
  } else {
    int k = n / 2;
    Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k)), A21(k, vector<int>(k)), A22(k,
vector<int>(k));
    Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k)), B21(k, vector<int>(k)), B22(k,
vector<int>(k));
```

```
    for (int i = 0; i < k; ++i)
      for (int j = 0; j < k; ++j) {
        A11[i][j] = A[i][j];
        A12[i][j] = A[i][j + k];
        A21[i][j] = A[i + k][j];
        A22[i][j] = A[i + k][j + k];
        B11[i][j] = B[i][j];
        B12[i][j] = B[i][j + k];
        B21[i][j] = B[i + k][j];
        B22[i][j] = B[i + k][j + k];
      }

    Matrix M1 = strassen(add(A11, A22), add(B11, B22));
    Matrix M2 = strassen(add(A21, A22), B11);
    Matrix M3 = strassen(A11, subtract(B12, B22));
    Matrix M4 = strassen(A22, subtract(B21, B11));
    Matrix M5 = strassen(add(A11, A12), B22);
    Matrix M6 = strassen(subtract(A21, A11), add(B11, B12));
    Matrix M7 = strassen(subtract(A12, A22), add(B21, B22));

    Matrix C11 = add(subtract(add(M1, M4), M5), M7);
    Matrix C12 = add(M3, M5);
    Matrix C21 = add(M2, M4);
    Matrix C22 = add(subtract(add(M1, M3), M2), M6);

    for (int i = 0; i < k; ++i)
      for (int j = 0; j < k; ++j) {
        C[i][j] = C11[i][j];
        C[i][j + k] = C12[i][j];
        C[i + k][j] = C21[i][j];
        C[i + k][j + k] = C22[i][j];
      }
  }
  return C;
}
```

Sample Input/Output

Input:
Matrix A:
1 2
3 4

Matrix B:
5 6
7 8

Output:
Resultant Matrix C:
19 22
43 50

Tasks for Students

- Implement Strassen's algorithm in C++.
- Verify correctness with different test cases of power-of-2 sizes.
- Compare with naive and divide-and-conquer methods.
- Handle non-power-of-2 sizes by padding matrices.

Viva Questions

- How does Strassen's algorithm improve over the standard method?
- Why are only 7 multiplications used?
- What is the time complexity of Strassen's algorithm?