Strassen Matrix Multiplication

Overview

Strassen's algorithm, proposed by Volker Strassen in 1969, is an algorithm for matrix multiplication that breaks down the problem into a set of seven smaller multiplication tasks, rather than the traditional eight. This approach results in a lower time complexity than the standard matrix multiplication algorithm.

Introduction

Matrix multiplication is an essential operation in several fields such as computer graphics, physics, and linear algebra. A common approach for multiplying two square matrices has a time complexity of $O(n^3)$, where n is the matrix dimension. Strassen's algorithm improves this to approximately $O(n^{log\ 7})$, demonstrating a significant speedup for large matrices.

Background

Traditionally, we use the Divide and Conquer algorithm to perform matrix multiplication. Divide and Conquer is a significant algorithm design technique used to solve complex problems. The strategy involves breaking down a problem into smaller sub-problems of the same type (divide), solving each sub-problem independently (conquer), and combining the solutions to solve the original problem.

The conventional divide-and-conquer algorithm for multiplying two matrices, A and B, involves the following steps:

1. Divide the input matrices A and B, each of size n × n, into four n/2 × n/2 sub-matrices.

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

2. Compute the product of A and B by recursively computing the products of the sub-matrices, and adding them appropriately.

$$C = A * B$$

$$= \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} * \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$$= \begin{vmatrix} A_{11}*B_{11} + A_{12}*B_{21} & A_{11}*B_{12} + A_{12}*B_{22} \\ A_{21}*B_{11} + A_{22}*B_{21} & A_{21}*B_{12} + A_{22}*B_{22} \end{vmatrix}$$

3. Each product on the right-hand side is the product of matrices of size n/2 × n/2, which can be computed recursively.

This algorithm reduces a problem of size n to eight smaller problems of size n/2, along with O(n^2) operations for the additions and subtractions, leading to time complexity of O(n^3).

Strassen's Algorithm

Volker Strassen proposed a unique method of matrix multiplication using the divide-and-conquer technique that reduces the number of recursive multiplications from eight (as in standard divide-and-conquer matrix multiplication) to seven. This reduction is crucial as it improves the time complexity to approximately O($n^{log\ 7}$), which is faster than the traditional method for sufficiently large matrices.

The steps of Strassen's algorithm are as follows:

1. Divide:

Similar to the standard divide-and-conquer algorithm, the input matrices A and B, each of size n × n, are divided into four n/2 × n/2 sub-matrices.

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

2. Formulate seven products:

In typical matrix multiplication, there are 8 multiplicative operations involved:

$$C = A * B$$
$$= \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} * \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$
$$= \begin{vmatrix} A_{11}*B_{11} + A_{12}*B_{21} & A_{11}*B_{12} + A_{12}*B_{22} \\ A_{21}*B_{11} + A_{22}*B_{21} & A_{21}*B_{12} + A_{22}*B_{22} \end{vmatrix}$$

Strassen found that with a clever grouping and rearrangement of these operations, the number could be reduced to 7 multiplicative operations.

Instead of directly multiplying corresponding sub-matrices, Strassen proposed that certain linear combinations of sub-matrices be multiplied instead. These products are formed by adding or subtracting the sub-matrices of A and B. Each product involves a multiplication operation and hence a recursive call, and hence the number of recursive calls is reduced from 8 to 7.

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

3. Formulate Four Quadrants:

The four quadrants of the output matrix C are then calculated using the seven products:

$C_{11} = P_5 + P_4 - P_2 + P_6$
$C_{12} = P_1 + P_2$
$C_{21} = P_3 + P_4$
$C_{22} = P_5 + P_1 - P_3 - P_7$

4. Recurse and Conquer: Steps 2 and 3 are recursively applied to smaller and smaller sub-matrices until the base case is reached, where a single element is multiplied by another.

The algorithm's correctness stems from the fact that these formulas for P_1, ..., P_7, and C_{11}, ..., C_{22} are equivalent to the equations you would get if you manually performed the multiplication step by step. This approach essentially computes the same result as traditional matrix multiplication but does so in a more efficient manner for large matrices.

Time Complexity

The Master Theorem is a key tool in algorithm analysis for describing the running time of divide-and-conquer algorithms. It provides an easy way to determine the time complexity of recurrence relations of the form:

$$T(n) = aT(n/b) + O(n^d)$$

With:

- $T(n)$ is the running time of the problem of size 'n'.
- a represents the number of subproblems in the recursion.
- Each subproblem has a size that is $1/b$ fraction of the original problem size.
- $O(n^d)$is the time we need to solve the non-recursive part of the algorithm (usually combining the results from recursive calls).

The Master Theorem can handle recurrences of three different types:

$$If\ a > b^d,\ then\ T(n) = O(n^{log_b(a)})$$
$$If\ a = b^d,\ then\ T(n) = O(n^d log\ n)$$
$$If\ a < b^d,\ then\ T(n) = 0(n^d)$$

The Master Theorem simplifies the process of solving recurrence relations that come up when analyzing the time complexity of recursive algorithms, especially divide-and-conquer algorithms.

In the case of Strassen's algorithm, we have

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

With:

- $a = 7$ (since there are seven recursive calls)
- $b = 2$ (since the problem size is halved at each step)
- $d = 2$ (for the $O(n^2)$ cost of adding and subtracting matrices)

To apply the Master Theorem, we need to compare a and b^d. In this case, $b^d = 2^2 = 4$ and $a = 7$. Since $a > b^d$, we're in Case 3 of the Master Theorem. Therefore, the time complexity is:

$$T(n) = O(n^{log_b(a)}) = O(n^{log_2 7})$$

Space Complexity

Strassen's algorithm uses auxiliary space to store the intermediate matrices and for the recursive calls. Therefore, the space complexity can be reduced to $O(n^2)$ (the size of the input matrices) by using an iterative process and reusing memory.

Proof of Correctness

To prove the correctness of Strassen's Algorithm, it's necessary to demonstrate that the computation of the seven products (P_1 through P_7) and the resulting sub-matrices (C_{11}, C_{12}, C_{21}, C_{22}) are equivalent to standard matrix multiplication.

The original matrices A and B are split as follows:

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

Strassen's algorithm computes seven products (P_1 through P_7) as follows:

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

Then, the algorithm combines these products to form the quadrants of the resulting matrix C:

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

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

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

To show the correctness of these formulas, we can explicitly expand these formulas and see that they do match the original formulas for matrix multiplication:

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

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

$= A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22} - A_{11}B_{22} - A_{12}B_{22} + A_{12}B_{21} - A_{22}B_{21}$

$= A_{11}B_{11} + A_{12}B_{21}$ (the standard formula for C_{11} in matrix multiplication)

Similarly, we can show that C_{12}, C_{21}, and C_{22} also match their respective formulas in standard matrix multiplication. Hence, Strassen's algorithm is correct.

Implementation

Below is the implementation of the Strassen Algorithm in C++ including test cases:

```cpp
//-------------------------strassen.cpp----------------------------------
// Author: Noah Nguyen
// Created: May 27, 2023
//-----------------------------------------------------------------------
// Implementing the Strassen's Algorithm
//-----------------------------------------------------------------------

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

/** 2 n x n matrices are presented by
 *      vector<vector<int>> A
 *      vector<vector<int>> B
 *      A.size() == B.size() == n
 */

//-------------------------subtractMatrix()-----------------------------
// perform matrix subtraction
// C = A - B
//-----------------------------------------------------------------------
vector<vector<int>> subtractMatrix(vector<vector<int>> A,
                                   vector<vector<int>> B) {
```

```
  int n = A.size();
  vector<vector<int>> 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;
} // end of subtractMatrix

//-------------------------subtractMatrix()-----------------------------
// perform matrix addition
// C = A + B
//----------------------------------------------------------------------
vector<vector<int>> addMatrix(vector<vector<int>> A, vector<vector<int>>
B) {
  int n = A.size();
  vector<vector<int>> 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;
} // end of addMatrix

//-------------------------strassen()-----------------------------------
// perform multiplication using strassen algorithm
//----------------------------------------------------------------------
vector<vector<int>> strassen(vector<vector<int>> A, vector<vector<int>> B)
{
  int n = A.size();
  // make sure both are n size
  // return null vector
  if (n != B.size()) {
    cerr << "Invalid matrix" << endl;
    return vector<vector<int>>();
  }
  // matrix size 1 and 2 -- base case (2x2 matrix)
  // compute sing standard method because the overhead of further
recursion or
  // the Strassen's method wouldn't be beneficial for such small-sized
matrices.
  if (n <= 2) {
```

```
    vector<vector<int>> C(n, vector<int>(n));
    C[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0];
    C[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1];
    C[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0];
    C[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1];
    return C;
  } else {
    // splitting matrix
    int new_size = n / 2;
    vector<vector<int>> a11(new_size, vector<int>(new_size)),
        a12(new_size, vector<int>(new_size)),
        a21(new_size, vector<int>(new_size)),
        a22(new_size, vector<int>(new_size)),
        b11(new_size, vector<int>(new_size)),
        b12(new_size, vector<int>(new_size)),
        b21(new_size, vector<int>(new_size)),
        b22(new_size, vector<int>(new_size));

    // dividing matrices into 4 sub-matrices
    for (int i = 0; i < new_size; i++) {
      for (int j = 0; j < new_size; j++) {
        a11[i][j] = A[i][j];
        a12[i][j] = A[i][j + new_size];
        a21[i][j] = A[i + new_size][j];
        a22[i][j] = A[i + new_size][j + new_size];

        b11[i][j] = B[i][j];
        b12[i][j] = B[i][j + new_size];
        b21[i][j] = B[i + new_size][j];
        b22[i][j] = B[i + new_size][j + new_size];
      }
    }

    // compute 7 products
    vector<vector<int>> p1 = strassen(a11, subtractMatrix(b12, b22)),
                        p2 = strassen(addMatrix(a11, a12), b22),
                        p3 = strassen(addMatrix(a21, a22), b11),
                        p4 = strassen(a22, subtractMatrix(b21, b11)),
                        p5 = strassen(addMatrix(a11, a22), addMatrix(b11,
b22)),
```

```
                        p6 = strassen(subtractMatrix(a12, a22),
                                      addMatrix(b21, b22)),
                        p7 = strassen(subtractMatrix(a11, a21),
                                      addMatrix(b11, b12));

    // calculate 4 quadrants
    vector<vector<int>> c11 = addMatrix(subtractMatrix(addMatrix(p5, p4),
p2),
                                        p6),
                        c12 = addMatrix(p1, p2), c21 = addMatrix(p3, p4),
                        c22 = subtractMatrix(
                            subtractMatrix(addMatrix(p5, p1), p3), p7);

    // calculate the final result
    vector<vector<int>> C(n, vector<int>(n));
    for (int i = 0; i < new_size; i++) {
      for (int j = 0; j < new_size; j++) {
        C[i][j] = c11[i][j];
        C[i][j + new_size] = c12[i][j];
        C[i + new_size][j] = c21[i][j];
        C[i + new_size][j + new_size] = c22[i][j];
      }
    }
    return C;
  }

} // end of strassen

int main() {

//-----------------------test1-----------------------------------------------
  vector<vector<int>> A1 = {{1, 2}, {3, 4}};
  vector<vector<int>> B1 = {{5, 6}, {7, 8}};

  cout << "Matrix A1: \n";
  for (int i = 0; i < A1.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < A1[i].size(); j++) {
      cout << A1[i][j] << " ";
    }
```

```
    cout << "]\n";
  }

  cout << "\nMatrix B1: \n";
  for (int i = 0; i < B1.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < B1[i].size(); j++) {
      cout << B1[i][j] << " ";
    }
    cout << "]\n";
  }

  vector<vector<int>> C1 = strassen(A1, B1);

  cout << "\nMatrix C1 = A1 * B1: \n";
  for (int i = 0; i < C1.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < C1[i].size(); j++) {
      cout << C1[i][j] << " ";
    }
    cout << "]\n";
  }

//------------------------------------------------------------------------

//----------------------test2-------------------------------------------------
  vector<vector<int>> A2 = {
      {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}};
  vector<vector<int>> B2 = {
      {17, 18, 19, 20}, {21, 22, 23, 24}, {25, 26, 27, 28}, {29, 30, 31,
32}};

  cout << "Matrix A2: \n";
  for (int i = 0; i < A2.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < A2[i].size(); j++) {
      cout << A2[i][j] << " ";
    }
    cout << "]\n";
  }
```

```
  cout << "\nMatrix B2: \n";
  for (int i = 0; i < B2.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < B2[i].size(); j++) {
      cout << B2[i][j] << " ";
    }
    cout << "]\n";
  }

  vector<vector<int>> C2 = strassen(A2, B2);

  cout << "\nMatrix C2 = A2 * B2: \n";
  for (int i = 0; i < C2.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < C2[i].size(); j++) {
      cout << C2[i][j] << " ";
    }
    cout << "]\n";
  }

//------------------------------------------------------------------------

//-----------------------------test 3-------------------------------------
  vector<vector<int>> A3(10, vector<int>(10));
  vector<vector<int>> B3(10, vector<int>(10));
  int count = 1;
  for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
      A3[i][j] = count;
      B3[i][j] = count + 100; // just to create different matrices
      count++;
    }
  }
  vector<vector<int>> C3 = strassen(A3, B3);

  cout << "\nMatrix C3 = A3 * B3: \n";
  for (int i = 0; i < C3.size(); i++) {
    cout << "[ ";
    for (int j = 0; j < C3[i].size(); j++) {
```

```
      cout << C3[i][j] << " ";
    }
    cout << "]\n";
  }

//------------------------------------------------------------------------
  return 0;
}
```

Output:
Matrix A1:
[1 2]
[3 4]

Matrix B1:
[5 6]
[7 8]

Matrix C1 = A1 * B1:
[19 22]
[43 50]
Matrix A2:
[1 2 3 4]
[5 6 7 8]
[9 10 11 12]
[13 14 15 16]

Matrix B2:
[17 18 19 20]
[21 22 23 24]
[25 26 27 28]
[29 30 31 32]

Matrix C2 = A2 * B2:
[250 260 270 280]
[618 644 670 696]
[986 1028 1070 1112]
[1354 1412 1470 1528]

Matrix C3 = A3 * B3:

[6240 6280 6320 6360 0 6440 6480 6520 6560 0]
[17520 17640 17760 17880 0 18120 18240 18360 18480 0]
[28800 29000 29200 29400 0 29800 30000 30200 30400 0]
[40080 40360 40640 40920 0 41480 41760 42040 42320 0]
[0 0 0 0 0 0 0 0 0 0]
[62640 63080 63520 63960 0 64840 65280 65720 66160 0]
[73920 74440 74960 75480 0 76520 77040 77560 78080 0]
[85200 85800 86400 87000 0 88200 88800 89400 90000 0]
[96480 97160 97840 98520 0 99880 100560 101240 101920 0]
[0 0 0 0 0 0 0 0 0 0]

References

https://medium.com/swlh/strassens-matrix-multiplication-algorithm-936f42c2b344#:~:text=Strassen%20algorithm%20is%20a%20recursive,four%202%20x%202%20matrices.
https://en.wikipedia.org/wiki/Strassen_algorithm
https://www.geeksforgeeks.org/strassens-matrix-multiplication/
https://math.mit.edu/~djk/18.310/18.310F04/Matrix_%20Multiplication.html
https://www.interviewbit.com/blog/strassens-matrix-multiplication/
https://www.geeksforgeeks.org/how-to-analyse-complexity-of-recurrence-relation/
https://www.codingninjas.com/codestudio/library/strassens-matrix-multiplication