Name, SID, Date tim allec ....

In Class Assignment 18: Square Matrix Multiply Recursive

Benjamin Sanders, MS November 25, 2020

1 Introduction

You may work in groups of up to two or three students. Write all student names at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

....
....
....
....

2 Assignment Description

2.1 Big Picture

Square Matrix Multiply Recursive is a common function in Linear Algebra, and is used extensively in realtime image processing systems, such as video games.

2.2 Algorithm Implementation

Implement the following algorithm in Java, using the Vector data structure for any 1-D array, 2-D array (a.k.a. matrix), or linear algebra purposes.

To keep things simple, when we use a divide-and-conquer algorithm to compute the matrix product $C = A \cdot B$, we assume that n is an exact power of 2 in each of the $n \times n$ matrices. We make this assumption because in each divide step, we will divide $n \times n$ matrices into four $n/2 \times n/2$ matrices, and by assuming that n is an exact power of 2, we are guaranteed that as long as $n \geq 2$, the dimension $n/2$ is an integer.

Suppose that we partition each of A, B, and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \tag{4.9}$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. \tag{4.10}$$

Equation (4.10) corresponds to the four equations

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \tag{4.11}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \tag{4.12}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \tag{4.13}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \tag{4.14}$$

Each of these four equations specifies two multiplications of $n/2 \times n/2$ matrices and the addition of their $n/2 \times n/2$ products. We can use these equations to create a straightforward, recursive, divide-and-conquer algorithm:

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  if n == 1
4       c11 = a11 · b11
5  else partition A, B, and C as in equations (4.9)
6       C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
7       C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
8       C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
9       C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10 return C
```

This pseudocode glosses over one subtle but important implementation detail. How do we partition the matrices in line 5? If we were to create 12 new $n/2 \times n/2$ matrices, we would spend $\Theta(n^2)$ time copying entries. In fact, we can partition the matrices without copying entries. The trick is to use index calculations. We identify a submatrix by a range of row indices and a range of column indices of the original matrix. We end up representing a submatrix a little differently from how we represent the original matrix, which is the subtlety we are glossing over. The advantage is that, since we can specify submatrices by index calculations, executing line 5 takes only $\Theta(1)$ time (although we shall see that it makes no difference asymptotically to the overall running time whether we copy or partition in place).

2.3 Time Complexity Analysis

Where n is the number of data points in A and B together, analyze the time complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n^2)$ in the space below.

On^3

2.4 Space Complexity Analysis

Where n is the number of data points in A and B together, analyze the space complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n \cdot log(n))$ in the space below.

O(n^2)

Name, SID, Date ....

2.5 Optimize the Algorithm for a Purpose

Choose an optimization, either in time or in space, for the given algorithm. Write your intended big-O notation, i.e. $O(1)$, $O(n)$, etc., in the space below, and write N/A in the other space.

- Time Complexity: ....O(n^2.81)..................
- Space Complexity: ....O(n^2)..................

What application would benefit from the purpose of the above optimization? Why? Write two sentences to answer these questions in the space below.

Many machine learning algorithms involve matrix operations, such as matrix factorization, neural network training, and clustering algorithms.

2.6 New Algorithm Design

In the space below, design an algorithm that achieves the same purpose of the given algorithm, but includes the optimization you have specified above. Use pseudocode written in a style similar to the given algorithm. You may use as many additional pages as necessary for this purpose.

```
import java.util.Arrays;

public class StrassenMatrixMultiply {
    public static void main(String[] args) {
        int[][] A = {{1, 2}, {3, 4}};
        int[][] B = {{5, 6}, {7, 8}};
        int[][] C = strassenMatrixMultiply(A, B);
        for (int i = 0; i < C.length; i++) {
            for (int j = 0; j < C[0].length; j++) {
                System.out.print(C[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static int[][] strassenMatrixMultiply(int[][] A, int[][] B) {
        int n = A.length;

        // Base case: When matrix size is small, switch to naive matrix multiplication
        if (n <= 64) {
            return naiveMatrixMultiply(A, B);
        }

        // Divide matrices into submatrices
        int[][] A11 = partition(A, 0, 0);
        int[][] A12 = partition(A, 0, n / 2);
        int[][] A21 = partition(A, n / 2, 0);
        int[][] A22 = partition(A, n / 2, n / 2);
        int[][] B11 = partition(B, 0, 0);
        int[][] B12 = partition(B, 0, n / 2);
        int[][] B21 = partition(B, n / 2, 0);
        int[][] B22 = partition(B, n / 2, n / 2);

        // Calculate Strassen submatrices
        int[][] S1 = subtract(B12, B22);
        int[][] S2 = add(A11, A12);
        int[][] S3 = add(A21, A22);
        int[][] S4 = subtract(B21, B11);
        int[][] S5 = add(A11, A22);
        int[][] S6 = add(B11, B22);
        int[][] S7 = subtract(A12, A22);
        int[][] S8 = add(B21, B22);
        int[][] S9 = subtract(A11, A21);
        int[][] S10 = add(B11, B12);

        // Recursive calls for Strassen submatrices
        int[][] P1 = strassenMatrixMultiply(A11, S1);
        int[][] P2 = strassenMatrixMultiply(S2, B22);
        int[][] P3 = strassenMatrixMultiply(S3, B11);
        int[][] P4 = strassenMatrixMultiply(A22, S4);
        int[][] P5 = strassenMatrixMultiply(S5, S6);
        int[][] P6 = strassenMatrixMultiply(S7, S8);
        int[][] P7 = strassenMatrixMultiply(S9, S10);

        // Combine results
        int[][] C11 = add(subtract(add(P5, P4), P2), P6);
        int[][] C12 = add(P1, P2);
        int[][] C21 = add(P3, P4);
        int[][] C22 = subtract(subtract(add(P5, P1), P3), P7);

        // Combine submatrices into result matrix
        int[][] C = new int[n][n];
        combine(C11, C, 0, 0);
        combine(C12, C, 0, n / 2);
        combine(C21, C, n / 2, 0);
        combine(C22, C, n / 2, n / 2);

        return C;
    }

    // Naive matrix multiplication algorithm
    public static int[][] naiveMatrixMultiply(int[][] A, int[][] B) {
        int n = A.length;
        int[][] C = new int[n][n];
        for (int i = 0; i < n; i++) {
```

3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.