A paper with text and numbers

Description automatically generated

A screenshot of a paper

Description automatically generated

A paper with text and a few words

Description automatically generated with medium confidence

Square MatrixMultiplyRecursive code:

public class SquareMatrixMultiplyRecursive {

    public static void main(String[] args) {

        int[][] A = {{1, 2}, {3, 4}};

        int[][] B = {{5, 6}, {7, 8}};

        int[][] C = squareMatrixMultiplyRecursive(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[][] squareMatrixMultiplyRecursive(int[][] A, int[][] B) {

        int n = A.length;

        int[][] C = new int[n][n];

        if (n == 1) {

            C[0][0] = A[0][0] \* B[0][0];

        } else {

            // Partition 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 the seven products recursively

            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);

            // Calculate the three resulting matrices recursively

            int[][] P1 = squareMatrixMultiplyRecursive(A11, S1);

            int[][] P2 = squareMatrixMultiplyRecursive(S2, B22);

            int[][] P3 = squareMatrixMultiplyRecursive(S3, B11);

            int[][] P4 = squareMatrixMultiplyRecursive(A22, S4);

            int[][] P5 = squareMatrixMultiplyRecursive(S5, S6);

            int[][] P6 = squareMatrixMultiplyRecursive(S7, S8);

            int[][] P7 = squareMatrixMultiplyRecursive(S9, S10);

            // Calculate the resulting submatrices

            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 the resulting submatrices into the final result matrix

            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;

    }

    public static int[][] partition(int[][] matrix, int row, int col) {

        int n = matrix.length / 2;

        int[][] submatrix = new int[n][n];

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                submatrix[i][j] = matrix[i + row][j + col];

            }

        }

        return submatrix;

    }

    public static void combine(int[][] submatrix, int[][] matrix, int row, int col) {

        int n = submatrix.length;

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                matrix[i + row][j + col] = submatrix[i][j];

            }

        }

    }

    public static int[][] add(int[][] A, int[][] B) {

        int n = A.length;

        int[][] C = new int[n][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;

    }

    public static int[][] subtract(int[][] A, int[][] B) {

        int n = A.length;

        int[][] C = new int[n][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;

    }

}

Output:

A screen shot of a computer program

Description automatically generated

Square MatrixMultiplyRecursive optimization:

public class SquareMatrixMultiplyRecursiveOptimization{

    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++) {

            for (int j = 0; j < n; j++) {

                for (int k = 0; k < n; k++) {

                    C[i][j] += A[i][k] \* B[k][j];

                }

            }

        }

        return C;

    }

    // Helper methods for matrix operations

    public static int[][] partition(int[][] matrix, int row, int col) {

        int n = matrix.length / 2;

        int[][] submatrix = new int[n][n];

        for (int i = 0; i < n; i++) {

            System.arraycopy(matrix[i + row], col, submatrix[i], 0, n);

        }

        return submatrix;

    }

    public static void combine(int[][] submatrix, int[][] matrix, int row, int col) {

        int n = submatrix.length;

        for (int i = 0; i < n; i++) {

            System.arraycopy(submatrix[i], 0, matrix[i + row], col, n);

        }

    }

    public static int[][] add(int[][] A, int[][] B) {

        int n = A.length;

        int[][] C = new int[n][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;

    }

    public static int[][] subtract(int[][] A, int[][] B) {

        int n = A.length;

        int[][] C = new int[n][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;

    }

}

Output:

A screenshot of a computer program

Description automatically generated

All code works properly!!