Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з комп’ютерного практикуму №2 з дисципліни
«Технології паралельних обчислень»

«Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності»

Виконав(ла) *ІП-11 Прищепа В. С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Дифучин А. Ю.* _____
(прізвище, ім'я, по батькові)

Київ 2024

Завдання

1. Реалізуйте стрічковий алгоритм множення матриць. Результат множення записуйте в об’єкт класу Result. 30 балів.

2. Реалізуйте алгоритм Фокса множення матриць. 30 балів.

3. Виконайте експерименти, варіюючи розмірність матриць, які перемножуються, для обох алгоритмів, та реєструючи час виконання алгоритму. Порівняйте результати дослідження ефективності обох алгоритмів. 20 балів.

4. Виконайте експерименти, варіюючи кількість потоків, що використовується для паралельного множення матриць, та реєструючи час виконання. Порівняйте результати дослідження ефективності обох алгоритмів. 20 балів.

Виконання

Лістинг

FoxMatrixMultiplier.java

```
package org.example.task1;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

class FoxMatrixMultiplier {
    // Вкладений клас для зберігання результату множення матриці
    private static class Result {
        private final int[][] matrix; // Результуюча матриця
        private final int rowOffset; // Зсув за рядками
        private final int colOffset; // Зсув за стовпцями

        public Result(int[][] matrix, int rowOffset, int colOffset) {
            this.matrix = matrix;
            this.rowOffset = rowOffset;
            this.colOffset = colOffset;
        }

        public int[][] getMatrix() {
            return matrix;
        }
```

```
        public int getRowOffset() {
            return rowOffset;
        }

        public int getColOffset() {
            return colOffset;
        }
    }

    // Завдання для виконання множення матриці на блок
    private static class MatrixMultiplicationTask implements Callable<Result> {
        private final int[][] matrixA; // Перша матриця
        private final int[][] matrixB; // Друга матриця
        private final int rowOffset; // Зсув за рядками
        private final int colOffset; // Зсув за стовпцями
        private final int k; // Індекс блоку
        private final int blockSize; // Розмір блоку

        public MatrixMultiplicationTask(int[][] matrixA, int[][] matrixB, int rowOffset,
int colOffset, int k, int blockSize) {
            this.matrixA = matrixA;
            this.matrixB = matrixB;
            this.rowOffset = rowOffset;
            this.colOffset = colOffset;
            this.k = k;
            this.blockSize = blockSize;
        }

        @Override
        public Result call() {
            int[][] result = new int[blockSize][blockSize]; // Результуюча матриця
            for (int i = 0; i < blockSize; i++) {
                for (int j = 0; j < blockSize; j++) {
                    for (int x = 0; x < blockSize; x++) {
                        result[i][j] += matrixA[rowOffset + i][k + x] * matrixB[k + x]
[colOffset + j]; // Множення блоків матриць
                    }
                }
            }
            return new Result(result, rowOffset, colOffset); // Повернення результату
разом зі зсувами
        }
    }
```

```
    // Метод для множення матриць з використанням алгоритму Фокса та вказаною кількістю потоків
    public static int[][] multiply(int[][] matrixA, int[][] matrixB, int blockSize, int numThreads) {
        ExecutorService executor = Executors.newFixedThreadPool(numThreads); // Створення пула потоків
        int[][] result = new int[matrixA.length][matrixB[0].length]; // Результуюча матриця
        try {
            for (int i = 0; i < matrixA.length; i += blockSize) {
                for (int j = 0; j < matrixB[0].length; j += blockSize) {
                    List<Future<Result>> futures = new ArrayList<>();
                    for (int k = 0; k < matrixA[0].length; k += blockSize) {
                        futures.add(executor.submit(new MatrixMultiplicationTask(matrixA, matrixB, i, j, k, blockSize))); // Відправлення завдань для множення блоків
                    }
                    for (Future<Result> future : futures) {
                        Result res = future.get(); // Отримання результату
                        for (int x = 0; x < blockSize; x++) {
                            for (int y = 0; y < blockSize; y++) {
                                result[res.getRowOffset() + x][res.getColOffset() + y] += res.getMatrix()[x][y]; // Збереження результату
                            }
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            executor.shutdown(); // Завершення роботи пула потоків
        }
        return result; // Повернення результату
    }

    public static void main(String[] args) {
        int SIZE = 3;
        int[][] testMatrixA = MatrixUtils.generateMatrix(SIZE, SIZE);
        System.out.println("Test Matrix A:");
        MatrixUtils.printMatrix(testMatrixA);
        int[][] testMatrixB = MatrixUtils.generateMatrix(SIZE, SIZE);
        System.out.println("Test Matrix B:");
        MatrixUtils.printMatrix(testMatrixB);
        int[][] testResult = multiply(testMatrixA, testMatrixB, 1, 8);
        System.out.println("Result Matrix:");
```

```
        MatrixUtils.printMatrix(testResult);

        int[] dimensions = {500, 1000, 1500, 2000, 2500, 3000};
        for (int size : dimensions) {
            int[][] matrixA = MatrixUtils.generateMatrix(size, size);
            int[][] matrixB = MatrixUtils.generateMatrix(size, size);
            long startTime = System.currentTimeMillis();
            int[][] result = multiply(matrixA, matrixB, 50, 8);
            long endTime = System.currentTimeMillis();
            System.out.println("Matrix size " + size + ": " + (endTime - startTime) + "
ms");
        }
    }
}
```

MatrixMultiplier.java

```
package org.example.task1;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

class MatrixMultiplier {
    // Вкладений клас для зберігання результату множення матриці на визначений
рядок
    private static class Result {
        private final int[] row; // Результуючий рядок
        private final int rowIndex; // Індекс рядка у вихідній матриці

        public Result(int[] row, int rowIndex) {
            this.row = row;
            this.rowIndex = rowIndex;
        }

        public int[] getRow() {
            return row;
        }

        public int getRowIndex() {
            return rowIndex;
        }
    }

    // Завдання для виконання множення матриці на визначений рядок
    private static class MatrixMultiplicationTask implements Callable<Result> {
```

```
        private final int[][] matrixA; // Перша матриця
        private final int[][] matrixB; // Друга матриця
        private final int rowIndex; // Індекс рядка для обчислення

        public MatrixMultiplicationTask(int[][] matrixA, int[][] matrixB, int rowIndex)
{
            this.matrixA = matrixA;
            this.matrixB = matrixB;
            this.rowIndex = rowIndex;
        }

        @Override
        public Result call() {
            // Виконання множення для визначеного рядка
            int[] rowResult = new int[matrixB[0].length]; // Результуючий рядок
            for (int j = 0; j < matrixB[0].length; j++) {
                int sum = 0;
                for (int k = 0; k < matrixA[0].length; k++) {
                    sum += matrixA[rowIndex][k] * matrixB[k][j];
                }
                rowResult[j] = sum; // Збереження результату у рядку
            }
            return new Result(rowResult, rowIndex); // Повернення обчисленого рядка
разом з його індексом
        }
    }

    // Метод для множення двох матриць з вказаною кількістю потоків
    public static int[][] multiply(int[][] matrixA, int[][] matrixB, int numThreads) {
        ExecutorService executor = Executors.newFixedThreadPool(numThreads); //
Створення пула потоків
        int[][] result = new int[matrixA.length][matrixB[0].length]; // Результуюча
матриця
        List<Future<Result>> futures = new ArrayList<>(); // Список для зберігання
результатів кожного рядка множення
        try {
            // Відправлення завдань множення для кожного рядка
            for (int i = 0; i < matrixA.length; i++) {
                futures.add(executor.submit(new MatrixMultiplicationTask(matrixA,
matrixB, i)));
            }
            // Отримання результатів і заповнення результатуючої матриці
            for (Future<Result> future : futures) {
                Result res = future.get(); // Отримання обчисленого рядка
                result[res.getRowIndex()] = res.getRow(); // Присвоєння рядка
```

```
відповідному індексу у результуючій матриці
            }
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
        executor.shutdown(); // Завершення роботи сервісу виконавців
        return result; // Повернення результуючої матриці
    }

    public static void main(String[] args) {
        int SIZE = 3;
        int[][] testMatrixA = MatrixUtils.generateMatrix(SIZE, SIZE);
        System.out.println("Test Matrix A:");
        MatrixUtils.printMatrix(testMatrixA);
        int[][] testMatrixB = MatrixUtils.generateMatrix(SIZE, SIZE);
        System.out.println("Test Matrix B:");
        MatrixUtils.printMatrix(testMatrixB);
        int[][] testResult = multiply(testMatrixA, testMatrixB, 8);
        System.out.println("Result Matrix:");
        MatrixUtils.printMatrix(testResult);

        int[] dimensions = {500, 1000, 1500, 2000, 2500, 3000};
        for (int size : dimensions) {
            int[][] matrixA = MatrixUtils.generateMatrix(size, size);
            int[][] matrixB = MatrixUtils.generateMatrix(size, size);
            long startTime = System.currentTimeMillis();
            int[][] result = multiply(matrixA, matrixB, 8);
            long endTime = System.currentTimeMillis();
            System.out.println("Matrix size " + size + ": " + (endTime - startTime) + "
ms");
        }
    }
}
```

MatrixUtils.java

```
package org.example.task1;

import java.util.Random;

public class MatrixUtils {
    // Метод для генерації випадкової матриці з вказаною кількістю рядків і
стовпців
    public static int[][] generateMatrix(int rows, int cols) {
        Random random = new Random();
        int[][] matrix = new int[rows][cols];
```

```
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int randomInt = random.nextInt(10) + 1;
                matrix[i][j] = randomInt;
            }
        }
        return matrix;
    }

    // Метод для виведення матриці на екран
    public static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

Результат

```
Test Matrix A:
9   3   3
10  4   1
8   10  7

Test Matrix B:
2   2   3
6   3   1
1   10  5

Result Matrix:
39  57  45
45  42  39
83  116 69

Matrix size 500: 84 ms
Matrix size 1000: 749 ms
Matrix size 1500: 8946 ms
Matrix size 2000: 25843 ms
Matrix size 2500: 60441 ms
Matrix size 3000: 118780 ms
```

Виконання множення матриць стрічковим алгоритмом.

Виконання множення матриць алгоритмом Фокса.

Тестування

Проведемо тестування для обох алгоритмів множення матриць, змінюючи кількість потоків та розмірність матриць. Результат записано у мс.

Розмір матриці	2 потоки		4 потоки		8 потоків	
	Стрічковий	Фокса	Стрічковий	Фокса	Стрічковий	Фокса
500	84	84	64	69	60	93
1000	749	516	521	334	414	273
1500	8946	1960	4323	1096	2812	766
2000	25843	4479	14684	2428	8174	1647
2500	60441	9735	30799	4902	17499	2957
3000	118780	15384	58137	8230	40631	5070

З результатів видно, що алгоритм Фокса в рази (до 10 раз) швидший за стрічковий при великій розмірності матриці (не залежно від кількості потоків). При невеликій розмірності матриці (500 на 500) Фокс програє стрічковому, але не значно. Загалом, алгоритм Фокса ефективніший за стрічковий по всім параметрам в рази.

Висновок: Виконуючи лабораторну роботу, я реалізував стрічковий алгоритм та алгоритм Фокса для множення матриць. Виконав експерименти, варіюючи розмірність матриць, які перемножуються, та кількість потоків, що використовуються для паралельного множення матриць, реєструючи час

виконання. Алгоритм Фокса виявився значно ефективнішим за часом, ніж стрічковий.