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»

Виконала студентка ІП-13 Замковий Дмитро Володимирович
(шифр, прізвище, ім'я, по батькові)

Перевірив ст.викл. Дифучин Антон Юрійович
(прізвище, ім'я, по батькові)

Оцінка _____

Дата оцінювання _____

Київ 2024

Лабораторна робота 2

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.**

Лістинги програми

Файл Main.java

```
public class Main {
    public static void main(String[] args) {
        Result test = new Result(500);

        test.Fox(10);
        test.outTime("Fox");

        test.Striping(10);
        test.outTime("Striping");

        test.Basic();
        test.outTime("Basic");
    }
}
```

Файл Fox.java

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.ArrayList;

public class Fox {
    private final Matrix A;
    private final Matrix B;
    private final Matrix Result;
    private int nTreads;

    public Fox(Matrix A, Matrix B, int nTreads) {
        this.A = A;
        this.B = B;
        this.Result = new Matrix(A.getRows(), B.getCols(), 0);
        this.nTreads = nTreads;
    }

    private static int findDividers(int s, int p) {
        int i = s;
        while (i > 1 && p % i != 0) {
```

```
            if (i >= s)
                i++;
            else
                i--;

            if (i > Math.sqrt(p))
                i = Math.min(s, p / s) - 1;
        }

        return i >= s ? i : (i != 0 ? p / i : p);
    }

    private Matrix copyBlock(Matrix matrix, int i, int j, int size) {
        Matrix result = new Matrix(size, size);

        for (int k = 0; k < size; k++) {
            for (int n = 0; n < size; n++) {
                result.set(k, n, matrix.get(k + i, j + n));
            }
        }

        return result;
    }

    public Matrix multiply() {
        this.nTreads = Fox.findDividers(Math.min(this.nTreads,
this.A.getRows()), this.A.getRows());
        int step = this.A.getRows() / this.nTreads;

        ExecutorService executor = Executors.newFixedThreadPool(this.nTreads);
        ArrayList<Future> threads = new ArrayList<>();

        int[][] matrixOfSizesI = new int[this.nTreads][this.nTreads];
        int[][] matrixOfSizesJ = new int[this.nTreads][this.nTreads];

        int stepI = 0;
        for (int i = 0; i < this.nTreads; i++) {
            int stepJ = 0;
            for (int j = 0; j < this.nTreads; j++) {
                matrixOfSizesI[i][j] = stepI;
                matrixOfSizesJ[i][j] = stepJ;
                stepJ += step;
            }
            stepI += step;
        }

        for (int i = 0; i < this.nTreads; i++) {
            for (int j = 0; j < this.nTreads; j++) {
                for (int k = 0; k < this.nTreads; k++) {
                    int stepI0 = matrixOfSizesI[i][k];
                    int stepJ0 = matrixOfSizesJ[j][k];

                    int stepI1 = matrixOfSizesI[i][(j + i) % this.nTreads];
                    int stepJ1 = matrixOfSizesJ[j][(j + i) % this.nTreads];

                    int stepI2 = matrixOfSizesI[(j + i) % this.nTreads][k];
                    int stepJ2 = matrixOfSizesJ[(j + i) % this.nTreads][k];

                    threads.add(
                            executor.submit(
                                    new FoxThread(
                                            this.copyBlock(this.A, stepI1,
stepJ1, step),
                                            this.copyBlock(this.B, stepI2,
```

```
stepJ2, step),
                                                this.Result,
                                                stepI0,
                                                stepJ0
                                            )
                                        )
                                    );
                                }
                            }
                        }

                        for (Future thread : threads) {
                            try {
                                thread.get();
                            } catch (Exception ignored) {}
                        }

                        executor.shutdown();
                        return this.Result;
                    }
                }
```

Файл FoxThread.java

```
public class FoxThread extends Thread {
    private final Matrix A;
    private final Matrix B;
    private final Matrix Result;
    private final int i;
    private final int j;

    public FoxThread(Matrix A, Matrix B, Matrix Result, int i, int j) {
        this.A = A;
        this.B = B;
        this.Result = Result;

        this.i = i;
        this.j = j;
    }

    @Override
    public void run() {
        Matrix block = multiplyBlock();

        for (int i = 0; i < block.getRows(); i++) {
            for (int j = 0; j < block.getCols(); j++) {
                this.Result.set(i + this.i, j + this.j, this.Result.get(i +
this.i, j + this.j) + block.get(i, j));
            }
        }
    }

    private Matrix multiplyBlock() {
        Matrix result = new Matrix(this.A.getRows(), this.B.getCols());

        for (int i = 0; i < this.A.getRows(); i++) {
            for (int j = 0; j < this.B.getCols(); j++) {
                for (int k = 0; k < this.A.getCols(); k++) {
                    result.set(i, j, result.get(i, j) + this.A.get(i, k) *
this.B.get(k, j));
                }
            }
        }
```

```
        return result;
    }
}
```

Файл Basic.java

```
public class Basic {
    private final Matrix A;
    private final Matrix B;
    private final Matrix Result;
    private final int n;
    private final int m;

    public Basic(Matrix A, Matrix B) {
        this.A = A;
        this.B = B;
        this.n = A.getRows();
        this.m = B.getCols();
        this.Result = new Matrix(this.n, this.m, 0);
    }

    public Matrix multiply() {
        for (int i = 0; i < this.n; i++) {
            for (int j = 0; j < this.m; j++) {
                for (int k = 0; k < this.A.getCols(); k++) {
                    this.Result.set(i, j, this.Result.get(i, j) + this.A.get(i,
k) * this.B.get(k, j));
                }
            }
        }
        return this.Result;
    }
}
```

Файл Matrix.java

```
public class Matrix {
    private final int[][] matrix;
    private final int rows;
    private final int cols;

    public Matrix(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        matrix = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = (int) (Math.random() * 10);
            }
        }
    }

    public Matrix(int rows, int cols, int value) {
        this.rows = rows;
        this.cols = cols;
        matrix = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = value;
```

```
            }
        }
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

    public int get(int i, int j) {
        return matrix[i][j];
    }

    public void set(int i, int j, int value) {
        matrix[i][j] = value;
    }

    public void set(Matrix matrix) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                this.matrix[i][j] = matrix.get(i, j);
            }
        }
    }

    public int[] get(int n) {
        return matrix[n];
    }

    public void set(int i, int[] row) {
        matrix[i] = row;
    }
}
```

Файл Result.java

```
public class Result {
    private final Matrix A;
    private final Matrix B;
    private final Matrix Result;
    private long startTime;
    private long endTime;

    public Result(int n) {
        this.A = new Matrix(n, n);
        this.B = new Matrix(n, n);
        this.Result = new Matrix(n, n, 0);
        this.startTime = 0;
        this.endTime = 0;
    }

    public void Basic() {
        this.startTime = System.nanoTime();

        Basic stripingLinear = new Basic(A, B);
        this.Result.set(stripingLinear.multiply());

        this.endTime = System.nanoTime();
    }
```

```
    public void Fox(int nTreads) {
        this.startTime = System.nanoTime();

        Fox stripingLinear = new Fox(A, B, nTreads);
        this.Result.set(stripingLinear.multiply());

        this.endTime = System.nanoTime();
    }

    public void Striping(int nTreads) {
        this.startTime = System.nanoTime();

        Striping stripingLinear = new Striping(A, B, nTreads);
        this.Result.set(stripingLinear.multiply());

        this.endTime = System.nanoTime();
    }

    public long getTime() {
        return (this.endTime - this.startTime) / 1000000;
    }

    public void outTime(String name) {
        System.out.println("Time: " + this.getTime() + " ms");
    }
}
```

Файл Striping.java

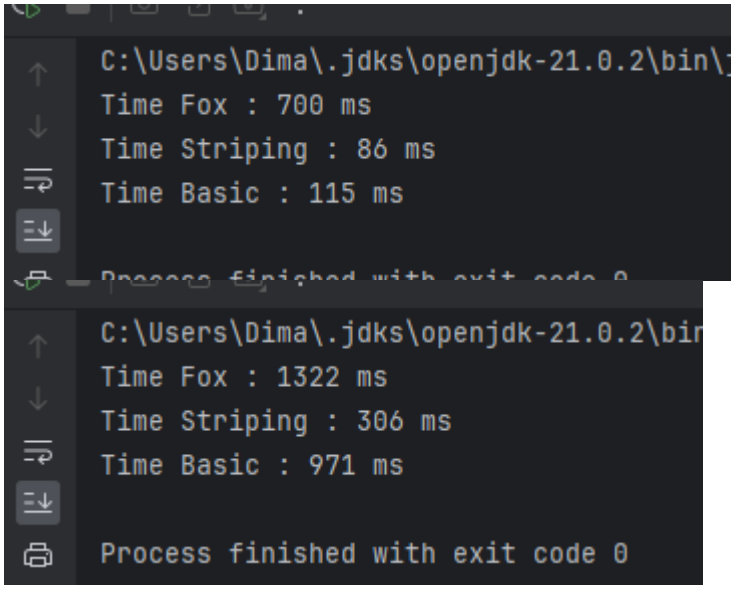
```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Striping {
    protected Matrix A;
    protected Matrix B;
    protected Matrix Result;
    protected int nThreads;

    public Striping(Matrix A, Matrix B, int nThreads) {
        this.A = A;
        this.B = B;
        this.nThreads = nThreads;
        this.Result = new Matrix(A.getRows(), B.getCols(), 0);
    }

    public Matrix multiply() {
        ExecutorService executor = Executors.newFixedThreadPool(this.nThreads);

        List<Future<int[]>> list = new ArrayList<>();

        for (int i = 0; i < this.A.getRows(); i++) {
            Callable<int[]> worker = new StripingCallable(this.A.get(i),
this.B);
            Future<int[]> submit = executor.submit(worker);
            list.add(submit);
        }

        for (int i = 0; i < list.size(); i++) {
            try {
```

```
                this.Result.set(i, list.get(i).get());
            } catch (Exception ignored) {}
        }

        executor.shutdown();

        return this.Result;
    }
}
```

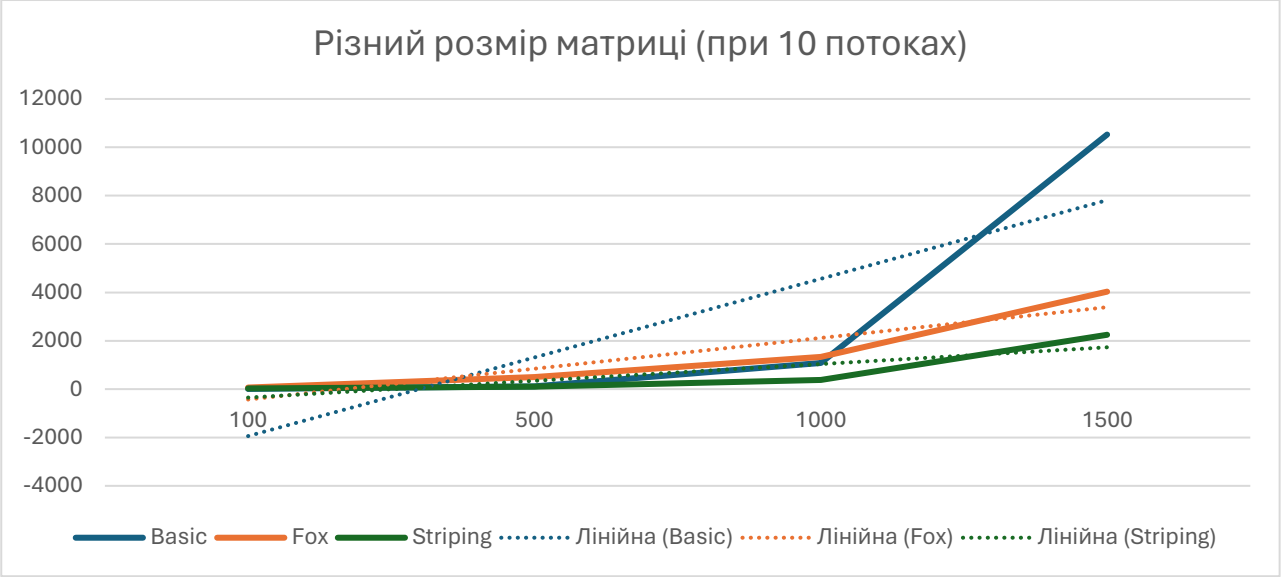
Файл StripingCallable.java

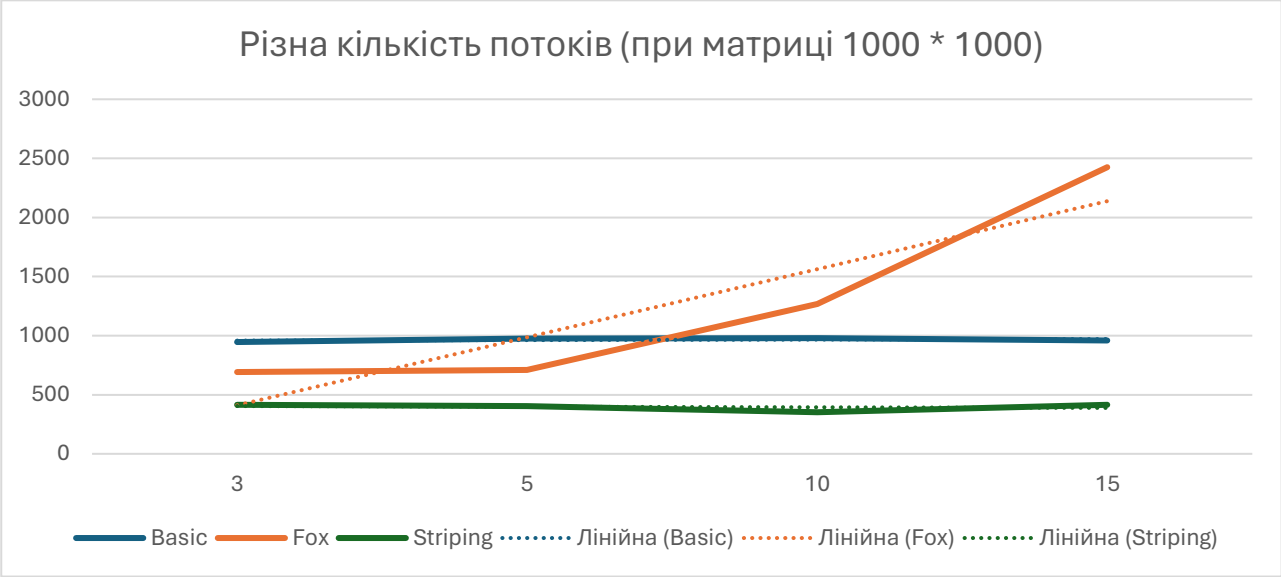
```
import java.util.concurrent.Callable;

public class StripingCallable implements Callable<int[]> {
    private final int[] row;
    private final Matrix B;
    private final int[] result;

    public StripingCallable(int[] row, Matrix B) {
        this.row = row;
        this.B = B;
        this.result = new int[B.getCols()];
    }

    @Override
    public int[] call() {
        for (int i = 0; i < B.getCols(); i++) {
            for (int j = 0; j < this.result.length; j++) {
                this.result[i] += row[j] * B.get(j, i);
            }
        }
        return this.result;
    }
}
```

Скріншоти



Інші результати





Висновки

В ході даної лабораторної роботи було розроблено та досліджено різні алгоритму множення матриць. З результатів слід зазначити, що при базовому алгоритмі значно збільшується час виконання, проте цей алгоритм не є паралельним, а тобто не залежить від кількості потоків. Час виконання стрічкового алгоритму не залежить від кількості потоків, проте при збільшенні розмірності матриці показує хорошу ефективність обислення. Алгоритм Фокса показує свою ефективність при великих розмірностях матриці, проте завелика кількість потоків заважає його роботі. Можна підбити зазначивши, що алгоритм Фокса добре працює тільки при великих матриця, базовий – при малих, а Стрічковий – при середній розмірності матриці.