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 №8 з дисципліни
«Технології паралельних обчислень»

«Розробка алгоритмів для розподілених систем клієнт-серверної архітектури»

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. Розробити веб-застосування клієнт-серверної архітектури, що реалізує алгоритм множення матриць або інший, який був Вами реалізований в рамках курсу «Технології паралельних обчислень», на стороні сервера з використанням паралельних обчислень. Розгляньте два варіанти реалізації 1) дані для обчислень знаходяться на сервері та 2) дані для обчислень знаходяться на клієнтській частині застосування. 60 балів.
2. Дослідити швидкість виконання запиту користувача при різних обсягах даних. 30 балів.
3. Порівняти реалізацію алгоритму в клієнт-серверній системі та в розподіленій системі з рівноправними процесорами. 10 балів.

Виконння

Лістинг

Server.java

```
import java.io.*;
import java.net.*;

public class Server {
    private static final boolean OUTPUT = true;
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(5000)) {
            System.out.println("The server is waiting for a connection...");
            Socket socket = serverSocket.accept();
            System.out.println("Client is connected.");

            ObjectOutputStream output = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream input = new ObjectInputStream(socket.getInputStream());

            while (true){
                boolean ClientGenerate = input.readBoolean();
                if (ClientGenerate) {
                    try {
                        Matrix matrix1 = (Matrix) input.readObject();
                        Matrix matrix2 = (Matrix) input.readObject();
                        StripeAlgorithmForkJoin stripeAlgorithmFJ = new
StripeAlgorithmForkJoin(matrix1, matrix2, 20);
                        stripeAlgorithmFJ.multiply();
```

```
                    output.writeObject(stripeAlgorithmFJ.getResult().getResultMatrix());
                    output.flush();
                } catch (ClassNotFoundException cnfe) {
                    System.out.println("Class not found for read object: " +
cnfe.getMessage());
                    cnfe.printStackTrace();
                }
            }
            else {
                int mSize = input.readInt();
                Matrix matrix1 = new Matrix(mSize, mSize, true);
                Matrix matrix2 = new Matrix(mSize, mSize, true);
                if (OUTPUT){
                    System.out.println("First Matrix:");
                    matrix1.OutputMatrix();
                    System.out.println("Second Matrix:");
                    matrix2.OutputMatrix();
                }
                StripeAlgorithmForkJoin stripeAlgorithmFJ = new
StripeAlgorithmForkJoin(matrix1, matrix2, 20);
                stripeAlgorithmFJ.multiply();
                output.writeObject(stripeAlgorithmFJ.getResult().getResultMatrix());
                output.flush();
            }
        }
    } catch (IOException e) {
        System.out.println("I/O error: " + e.getMessage());
    }
  }
}
```

StripeAlgorithmForkJoin.java

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.TimeUnit;
import java.util.List;
import java.util.ArrayList;

public class StripeAlgorithmForkJoin {
  private Matrix rightMatrix, leftMatrix;
  private int numOfThreads;
  private Result result;

  public StripeAlgorithmForkJoin (Matrix leftMatrix, Matrix rightMatrix, int
numOfThreads)
  {
```

```
        this.leftMatrix = leftMatrix;
        this.rightMatrix = rightMatrix;
        this.numOfThreads = Math.min(numOfThreads, leftMatrix.GetNumOfRows());
        this.result = new Result(this.rightMatrix, this.leftMatrix);
    }

    Result getResult() { return result; }

    public void multiply() {
        ForkJoinPool pool = new ForkJoinPool(numOfThreads);
        long startTime = System.nanoTime();

        int chunkSize = (int) Math.ceil((double) leftMatrix.GetNumOfRows() /
numOfThreads);
        List<StripeTask> tasks = new ArrayList<>();
        for (int i = 0; i < leftMatrix.GetNumOfRows(); i += chunkSize) {
            int endRow = Math.min(i + chunkSize, leftMatrix.GetNumOfRows());
            tasks.add(new StripeTask(leftMatrix, rightMatrix, result, i, endRow));
        }
        for (StripeTask task : tasks){
            pool.execute(task);
        }

        pool.shutdown();
        try {
            pool.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        long endTime = System.nanoTime();
        result.setTime((endTime - startTime) / 1e6f);
    }

}
```

StripeTask.java

```
import java.util.concurrent.RecursiveAction;


public class StripeTask extends RecursiveAction {
    private Matrix leftMatrix;
    private Matrix rightMatrix;
    private Result result;
    private int startRow;
```

```
    private int endRow;

    public StripeTask(Matrix leftMatrix, Matrix rightMatrix, Result result, int
startRow, int endRow) {
        this.leftMatrix = leftMatrix;
        this.rightMatrix = rightMatrix;
        this.result = result;
        this.startRow = startRow;
        this.endRow = endRow;
    }

    @Override
    protected void compute() {
        for (int i = startRow; i < endRow; i++) {
            int[] rowResult = new int[rightMatrix.GetNumOfColumns()];
            for (int j = 0; j < rightMatrix.GetNumOfColumns(); j++) {
                int sum = 0;
                for (int k = 0; k < leftMatrix.GetNumOfColumns(); k++) {
                    sum += leftMatrix.GetMatrix()[i][k] * rightMatrix.GetMatrix()[k][j];
                }
                rowResult[j] = sum;
            }
            synchronized (result) {
                result.getResultMatrix().setRow(i, rowResult);
            }
        }
    }
}
```

Matrix.java

```
import java.io.Serializable;
import java.util.Random;

public class Matrix implements Serializable{
    private final int numOfRows;
    private final int numOfColumns;
    private int[][] matrix;

    public Matrix(int numOfRows, int numOfColumns, boolean Generate){
        this.numOfRows = numOfRows;
        this.numOfColumns = numOfColumns;
        matrix = new int[this.numOfRows][this.numOfColumns];
        Random random = new Random();
        if (Generate){
            for (int i = 0; i < this.numOfRows; i++) {
```

```
                for (int j = 0; j < this.numOfColumns; j++) {
                    matrix[i][j] = random.nextInt(10);
                }
            }
        }

    }

    public void setMatrix(int[][] matrix) { this.matrix = matrix; }
    public void setElement(int i, int j, int element){
        matrix[i][j] = element;
    }
    public int GetNumOfRows() { return numOfRows; }
    public int GetNumOfColumns() { return numOfColumns; }
    public int GetElement(int i, int j) { return matrix[i][j]; }
    public int [][] GetMatrix() { return matrix; }
    public int[] GetRow(int i) {
        int[] row = matrix[i];
        int length = row.length;
        int[] reversedRow = new int[length];
        for (int j = 0; j < length; j++) {
            reversedRow[j] = row[length - j - 1];
        }
        return reversedRow;
    }
    public void setRow(int i, int[] row) { System.arraycopy(row, 0, matrix[i], 0,
numOfColumns); }
    public int[] GetColumn(int j) {
        int[] column = new int[numOfRows];
        for (int i = 0; i < numOfRows; i++) {
            column[i] = matrix[i][numOfRows - j - 1];
        }
        return column;
    }

    public Matrix GetBlock(int startRow, int endRow, int startColumn, int
endColumn) {
        if (startRow < 0 || startRow >= numOfRows || endRow < 0 || endRow >
numOfRows ||
                startColumn < 0 || startColumn >= numOfColumns || endColumn < 0 ||
endColumn > numOfColumns) {
            throw new IllegalArgumentException("Invalid block boundaries");
        }

        int blockRows = endRow - startRow;
```

```
        int blockColumns = endColumn - startColumn;
        int[][] block = new int[blockRows][blockColumns];

        for (int i = startRow, blockRow = 0; i < endRow; i++, blockRow++) {
            for (int j = startColumn, blockColumn = 0; j < endColumn; j++,
blockColumn++) {
                block[blockRow][blockColumn] = matrix[i][j];
            }
        }

        Matrix Mblock = new Matrix(block.length, block[0].length, false);
        Mblock.setMatrix(block);

        return Mblock;
    }

    public void transformMatrix(){
        int[] firstColumn = this.GetColumn(0);

        for (int i = 0; i < numOfColumns - 1; i++) {
            for (int j = 0; j < numOfRows; j++) {
                matrix[j][i] = matrix[j][i + 1];
            }
        }

        for (int j = 0; j < numOfRows; j++) {
            matrix[j][numOfColumns - 1] = firstColumn[j];
        }
    }

    public void OutputMatrix(){
        for(int i = 0; i < numOfRows; i++){
            for(int j = 0; j < numOfColumns; j++){
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

Result.java

```
public class Result {
    private Matrix rightMatrix, leftMatrix, resultMatrix;
    private float time;
```

```
    public Result(Matrix rightMatrix, Matrix leftMatrix){
        this.rightMatrix = rightMatrix;
        this.leftMatrix = leftMatrix;
        resultMatrix = new Matrix(leftMatrix.GetNumOfColumns(),
rightMatrix.GetNumOfRows(), false);
    }

    public void setTime(float time) { this.time = time; }
    public float getTime() { return time; }

    public void setResultMatrix(int[][] matrix) { resultMatrix.setMatrix(matrix); }
    public Matrix getResultMatrix() { return resultMatrix; }
}
```

Client.java

```
import java.io.*;
import java.net.*;

public class Client {
    private static final boolean ClientGenereate = false;
    private static final boolean OUTPUT = true;
    public static void main(String[] args) {

        try (Socket socket = new Socket("localhost", 5000)) {
            ObjectOutputStream output = new
ObjectOutputStream(socket.getOutputStream());
            ObjectInputStream input = new ObjectInputStream(socket.getInputStream());

            BufferedReader consoleInput = new BufferedReader(new
InputStreamReader(System.in));
            String userInput;
            double start_time = System.currentTimeMillis();

            while (true) {
                System.out.println("compute/exit:");
                userInput = consoleInput.readLine();

                if ("exit".equalsIgnoreCase(userInput)) {
                    break;
                }
                else {
                    System.out.println("Enter size of matrices:");
                    int mSize = Integer.parseInt(consoleInput.readLine());
```

```
                start_time = System.currentTimeMillis();
                output.writeBoolean(ClientGenereate);
                if (ClientGenereate){
                    Matrix matrix1 = new Matrix(mSize, mSize, true);
                    Matrix matrix2 = new Matrix(mSize, mSize, true);
                    if (OUTPUT) {
                        System.out.println("First Matrix:");
                        matrix1.OutputMatrix();
                        System.out.println("Second Matrix:");
                        matrix2.OutputMatrix();
                    }
                    output.writeObject(matrix1);
                    output.writeObject(matrix2);
                }
                else {
                    output.writeInt(mSize);
                }
                output.flush();
            }

            try {
                Matrix response = (Matrix) input.readObject();
                double end_time = System.currentTimeMillis();
                if (OUTPUT){
                    System.out.println("Received matrix from server:");
                    response.OutputMatrix();
                }
                System.out.println("Time: " + (end_time - start_time) / 1000 + " s");
            } catch (ClassNotFoundException e) {
                System.out.println("Class not found: " + e.getMessage());
            }
        }

    } catch (NumberFormatException e) {
        System.out.println("Invalid input. Please enter a valid integer.");
    } catch (UnknownHostException e) {
        System.out.println("Server not found: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("I/O error: " + e.getMessage());
    }
  }
}
```

Результат

При генерації даних на стороні клієнта:

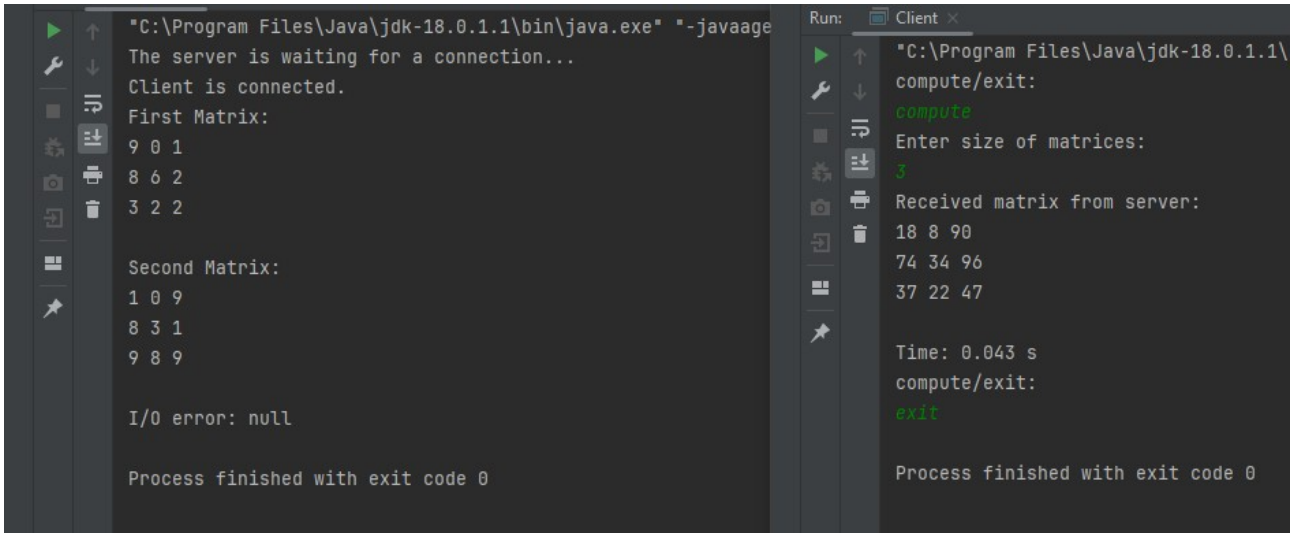
```
compute/exit:
compute
Enter size of matrices:
3
First Matrix:
9 3 2
5 8 7
1 8 1

Second Matrix:
2 6 3
2 6 4
6 9 7

Received matrix from server:
36 90 53
68 141 96
24 63 42

Time: 0.044 s
```

При генерації даних на стороні сервера:



Дослідимо швидкість виконання запиту клієнта при різних розмірах матриць:

Розмірність	Час виконання (секунд)	
	Дані на клієнтській стороні	Дані на серверній стороні
100x100	0.093	0.071

500x500	0.140	0.110
1000x1000	0.719	0.682
1500x1500	3.573	3.412
2000x2000	10.562	9.987
2500x2500	19.846	19.727
3000x3000	42.468	39.068

Висновок: Під час виконання лабораторної роботи мною було набуто навички розробки алгоритмів для розподілених систем клієнт-серверної архітектури. Було розроблено веб-застосування клієнт-серверної архітектури, що реалізує стрічковий алгоритм множення матриць з використанням ForkJoinFramework на стороні сервера. Було досліджено та порівняно швидкості виконання запитів клієнта при генерації даних як на клієнтській частині, так і на серверній частині. Виконання запитів при генерації даних на серверній частині виконується швидше, адже не витрачається час на передачу вхідних даних до сервера.