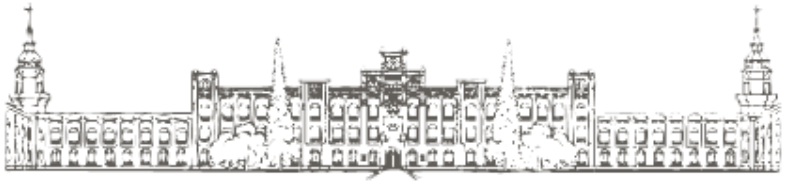

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Звіт

до лабораторного практикуму 4

«Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework»

з дисципліни

«Технології паралельних обчислень»

Виконав:
студент групи ІП-01
Пашковський Євгеній

Київ – 2023

Завдання

1. Побудуйте алгоритм статистичного аналізу тексту та визначте характеристики випадкової величини «довжина слова в символах» з використанням ForkJoinFramework.
2. Дослідіть побудований алгоритм аналізу текстових документів на ефективність експериментально.
3. Реалізуйте один з алгоритмів комп’ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework.
4. Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework.
5. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework.

Хід виконання

Main.java:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

public class Main {
    public static void main(String[] args) throws IOException {
        Main.runTask1();
    }

    public static void runTask1() throws IOException {
        CustomFileReader customFileReader = new CustomFileReader();
        String filePath = "E:\\Projects\\parallel-
computing\\lab4\\src\\files\\sherlock.txt";

        String text = customFileReader.readAll(filePath);

        TextAnalyzer textAnalyzer = new TextAnalyzer();

        final long singleThreadStartTime =
System.currentTimeMillis();
        OccasionalCharacteristics
occasionalCharacteristicsSingleThread =
textAnalyzer.getWordLengthCharacteristicsSingleThread(text);
        final long singleThreadEndTime =
System.currentTimeMillis();
        System.out.println("Single thread result:");
        System.out.println("Elapsed: " + (singleThreadEndTime -
singleThreadStartTime) + "ms");
        System.out.println(occasionalCharacteristicsSingleThread);

        final long forkJoinStartTime = System.currentTimeMillis();
        OccasionalCharacteristics occasionalCharacteristicsForkJoin
= textAnalyzer.getWordLengthCharacteristics(text);
        final long forkJoinEndTime = System.currentTimeMillis();
        System.out.println("Fork join result:");
        System.out.println("Elapsed: " + (forkJoinEndTime -
forkJoinStartTime) + "ms");
        System.out.println(occasionalCharacteristicsForkJoin);
    }

    public static void runTask2() {
        // in lab2
    }

    public static void runTask3() throws IOException {
//        String text1 = "Hello Hello Hello  1";
//        String text2 = "Hello no 1";

        CustomFileReader customFileReader = new CustomFileReader();
```

```
        String text1 =
customFileReader.readAll("E:\\Projects\\parallel-
computing\\lab4\\src\\files\\sherlock.txt");
        String text2 =
customFileReader.readAll("E:\\Projects\\parallel-
computing\\lab4\\src\\files\\romeo_and_juliet.txt");

        TextAnalyzer textAnalyzer = new TextAnalyzer();

        Set<String> commonWords =
textAnalyzer.findCommonWords(text1, text2);

        System.out.println(commonWords);
    }

    public static void runTask4() {
        TextAnalyzer textAnalyzer = new TextAnalyzer();

        List<String> keyWords = new ArrayList<>(List.of(new
String[]{"love", "crime"}));

        List<String> files =
textAnalyzer.findInDirectoryByKeyWords("E:\\Projects\\parallel-
computing\\lab4\\src\\files", keyWords);

        System.out.println(files);
    }
}
```

OccasionalCharacteristics.java:

```
public class OccasionalCharacteristics {
    private final double M;
    private final double Mx2;
    private final double D;
    private final double deviation;

    public OccasionalCharacteristics(double Mx, double Mx2) {
        this.M = Mx;
        this.Mx2 = Mx2;
        this.D = Mx2 - Math.pow(Mx, 2);
        this.deviation = Math.sqrt(D);
    }

    public double getMathematicalExpectation() {
        return M;
    }

    public double getMx2() {
        return Mx2;
    }

    public double getVariance() {
        return D;
    }

    public double getDeviation() {
```

```
        return deviation;
    }

    @Override
    public String toString() {
        return "M = " + M + "\n" + "D = " + D + "\n" + "V = " +
deviation + "\n";
    }
}
```

FindByKeyWordsTask.java:

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;

public class FindByKeyWordsTask extends RecursiveTask<List<String>>
{
    private final String path;
    private final List<String> keyWords;
    private final CustomFileReader customFileReader = new
CustomFileReader();

    public FindByKeyWordsTask(String path, List<String> keyWords) {
        this.path = path;
        this.keyWords = keyWords;
    }

    @Override
    protected List<String> compute() {
        File pathFile = new File(path);

        if (!pathFile.exists()) return new ArrayList<>();

        if (pathFile.isDirectory()) {
            File[] files = pathFile.listFiles();

            if (files == null) return new ArrayList<>();

            List<FindByKeyWordsTask> tasks =
Arrays.stream(files).map((file) -> new
FindByKeyWordsTask(file.getPath(), keyWords)).toList();

            return
invokeAll(tasks).stream().map(ForkJoinTask::join).reduce((acc, el)
-> {
                acc.addAll(el);
                return acc;
            }).orElseGet(ArrayList::new);
        }

        try {
            String text = customFileReader.readAll(path);
```

```
            for (String keyWord : keyWords) {
                if (text.matches(".*" + keyWord + ".*")) {
                    return new ArrayList<>(List.of(new
String[]{path}));
                }
            }

        } catch (IOException ignored) {
            System.out.println();
        }

        return new ArrayList<>();
    }
}
```

CustomFileReader.java:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class CustomFileReader {
    public String readAll(String path) throws IOException {
        BufferedReader reader = new BufferedReader(new
FileReader(path));

        StringBuilder textBuilder = new StringBuilder();

        while (true) {
            String line = reader.readLine();
            if (line == null) break;

            textBuilder.append(line).append("\n");
        }

        return textBuilder.toString();
    }
}
```

CommonWordsTask.java:

```
import java.util.*;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;

public class CommonWordsTask extends RecursiveTask<Set<String>> {
    private static final int THRESHOLD = 10;

    private final String[] words1;
    private final String[] words2;
    private final int start;
    private final int end;

    public CommonWordsTask(String[] words1, String[] words2, int
start, int end) {
        this.words1 = words1;
        this.words2 = words2;
```

```
        this.start = start;
        this.end = end;
    }

    @Override
    protected Set<String> compute() {
        if (end - start > THRESHOLD) {
            List<CommonWordsTask> dividedTasks = new ArrayList<>();
            int middleIndex = (start + end) / 2;
            dividedTasks.add(new CommonWordsTask(words1, words2,
start, middleIndex));
            dividedTasks.add(new CommonWordsTask(words1, words2,
middleIndex, end));

            return
invokeAll(dividedTasks).stream().map(ForkJoinTask::join).reduce((ac
c, el) -> {
                acc.addAll(el);
                return acc;
            }).get();
        }

        Set<String> commonWords = new HashSet<>();

        for (int i = start; i < end; i++) {
            String word1 = words1[i];
            for (String word2 : words2) {
                if (Objects.equals(word1, word2))
commonWords.add(word1);
            }
        }

        return commonWords;
    }
}
```

WordLengthCharacteristics.java:

```
import java.util.*;
import java.util.concurrent.RecursiveTask;

public class WordLengthCharacteristicsTask extends
RecursiveTask<OccasionalCharacteristics> {
    private static final int THRESHOLD = 1000;

    private final String[] words;
    private final int start;
    private final int end;


    public WordLengthCharacteristicsTask(String[] words, int start,
int end) {
        this.words = words;
        this.start = start;
        this.end = end;
    }
```

```
    @Override
    protected OccasionalCharacteristics compute() {
        if (end - start > THRESHOLD) {
            List<WordLengthCharacteristicsTask> dividedTasks = new
ArrayList<>();
            int middleIndex = (start + end) / 2;
            dividedTasks.add(new
WordLengthCharacteristicsTask(words, start, middleIndex));
            dividedTasks.add(new
WordLengthCharacteristicsTask(words, middleIndex, end));

            double Mx = 0;
            double Mx2 = 0;

            for (WordLengthCharacteristicsTask task :
invokeAll(dividedTasks)) {
                OccasionalCharacteristics result = task.join();

                Mx += result.getMathematicalExpectation();
                Mx2 += result.getMx2();
            }

            return new OccasionalCharacteristics(Mx, Mx2);
        }

        Map<Integer, Integer> countMap = new HashMap<>();

        for (int i = start; i < end; i++) {
            String word = words[i];

            int wordLength = word.length();

            if (!countMap.containsKey(wordLength)) {
                countMap.put(wordLength, 1);
                continue;
            }

            int count = countMap.get(wordLength);
            countMap.put(wordLength, count + 1);
        }

        double Mx = 0;
        double Mx2 = 0;

        for (Map.Entry<Integer, Integer> entry :
countMap.entrySet()) {
            double possibility = (double) entry.getValue() /
(double) words.length;
            Mx += entry.getKey() * possibility;
            Mx2 += Math.pow(entry.getKey(), 2) * possibility;
        }

        return new OccasionalCharacteristics(Mx, Mx2);
    }
}
```

TextAnalyzer.java:

```
import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.Future;

public class TextAnalyzer {
    private final String wordSplitRegexp = "\\W+";

    public OccasionalCharacteristics
getWordLengthCharacteristics(String text) {
        String[] words = text.split(wordSplitRegexp);

        ForkJoinPool forkJoinPool = ForkJoinPool.commonPool();

        Future<OccasionalCharacteristics>
occationalCharacteristicsFuture = forkJoinPool.submit(new
WordLengthCharacteristicsTask(words, 0, words.length));

        forkJoinPool.shutdown();

        try {
            return occationalCharacteristicsFuture.get();
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
    }

    public OccasionalCharacteristics
getWordLengthCharacteristicsSingleThread(String text) {
        String[] words = text.split(wordSplitRegexp);

        Map<Integer, Integer> countMap = new HashMap<>();

        for (String word : words) {
            int wordLength = word.length();

            if (!countMap.containsKey(wordLength)) {
                countMap.put(wordLength, 1);
                continue;
            }

            int count = countMap.get(wordLength);
            countMap.put(wordLength, count + 1);
        }

        double Mx = 0;
        double Mx2 = 0;

        for (Map.Entry<Integer, Integer> entry :
countMap.entrySet()) {
            double possibility = (double) entry.getValue() /
(double) words.length;
            Mx += entry.getKey() * possibility;
            Mx2 += Math.pow(entry.getKey(), 2) * possibility;
```

```
        }

        return new OccasionalCharacteristics(Mx, Mx2);
    }

    public Set<String> findCommonWords(String text1, String text2)
{
        String[] words1 = text1.split(wordSplitRegexp);
        String[] words2 = text2.split(wordSplitRegexp);

        ForkJoinPool forkJoinPool = ForkJoinPool.commonPool();

        Future<Set<String>> commonWordsFuture =
forkJoinPool.submit(new CommonWordsTask(words1, words2, 0,
words1.length));

        forkJoinPool.shutdown();

        try {
            return commonWordsFuture.get();
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
    }

    public Set<String> findCommonWordsSingleThread(String text1,
String text2) {
        Set<String> commonWords = new HashSet<>();

        String[] words1 = text1.split(wordSplitRegexp);
        String[] words2 = text2.split(wordSplitRegexp);

        for (String word1 : words1) {
            for (String word2 : words2) {
                if (Objects.equals(word1, word2))
commonWords.add(word1);
            }
        }

        return commonWords;
    }

    public List<String> findInDirectoryByKeyWords(String path,
List<String> keyWords) {
        ForkJoinPool forkJoinPool = ForkJoinPool.commonPool();
        Future<List<String>> foundFilesFuture =
forkJoinPool.submit(new FindByKeyWordsTask(path, keyWords));

        forkJoinPool.shutdown();

        try {
            return foundFilesFuture.get();
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
```

```
    }
}
```

Метод матриці для виконання множення матриць за допомогою ForkJoinPool:

```
public Result multiplyForkJoin(IntegerMatrix integerMatrix) {
    final long startTime = System.currentTimeMillis();

    if (!this.validateMatrixForProduct(integerMatrix)) throw new
RuntimeException("Matrix has bad size");

    ForkJoinPool forkJoinPool = ForkJoinPool.commonPool();

    Future<IntegerMatrix> integerMatrixFuture =
forkJoinPool.submit(new MultiplyTask(this.array,
integerMatrix.getTransponedMatrix().array, 0, this.array.length));

    forkJoinPool.shutdown();

    try {
        IntegerMatrix matrix = integerMatrixFuture.get();
        final long endTime = System.currentTimeMillis();
        return new Result(matrix, endTime - startTime);
    } catch (InterruptedException | ExecutionException e) {
        throw new RuntimeException(e);
    }
}
```

MultiplyTask:

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;

public class MultiplyTask extends RecursiveTask<IntegerMatrix> {
    private static final int THRESHOLD = 20;
    private final int[][] rows;
    private final int[][] columns;
    private final int start;
    private final int end;


    public MultiplyTask(int[][] rows, int[][] columns, int start,
int end) {
        this.rows = rows;
        this.columns = columns;
        this.start = start;
        this.end = end;
    }

    @Override
    protected IntegerMatrix compute() {
        if (end - start > THRESHOLD) {
```

```
            List<MultiplyTask> dividedTasks = new ArrayList<>();
            int middleIndex = (start + end) / 2;
            dividedTasks.add(new MultiplyTask(rows, columns, start,
middleIndex));
            dividedTasks.add(new MultiplyTask(rows, columns,
middleIndex, end));

            return
invokeAll(dividedTasks).stream().map(ForkJoinTask::join).reduce((ac
c, el) -> {
                int[][] accArray = acc.getArray();
                int[][] elArray = el.getArray();

                int[][] newArray = new int[accArray.length +
elArray.length][columns.length];

                System.arraycopy(accArray, 0, newArray, 0,
accArray.length);
                System.arraycopy(elArray, 0, newArray,
accArray.length, elArray.length);

                return new IntegerMatrix(newArray);
            }).orElseThrow();
        }

        int[][] multipliedRows = new int[end -
start][columns.length];

        for (int j = start; j < end; j++) {
            for (int i = 0; i < columns.length; i++) {
                for (int k = 0; k < columns.length; k++) {
                    multipliedRows[j - start][i] += rows[j][k] *
columns[i][k];
                }
            }
        }

        return new IntegerMatrix(multipliedRows);
    }
}
```

Тестування алгоритмів

Тестувалося на пристрої з такими характеристиками:

- Процесор Intel Core i7-8550U. Базова частота 1.8-2.0ГГц, але фактично може досягати 4ГГц у пікових навантаженнях. Містить 4 фізичних ядра та 8 логічних процесорів.
- 16Гб оперативної пам’яті з частотою 2400МГц
- ОС Windows 10.

Алгоритм статичного аналізу тексту (визначення характеристик випадкової величини «довжина слова»):

```
Single thread result:
Elapsed average: 21ms
M = 4.074754901960785
D = 4.342550284389059
V = 2.0838786635476305

Fork join result:
Elapsed average: 14ms
M = 4.074754901960785
D = 4.342550284389059
V = 2.0838786635476305
```

Алгоритм перемноження матриць за домогою ForkJoinPool:

Matrix Size	Serial algorithm, s	ForkJoinPool Algorithm	
		Time, s	Speed Up
500	0,145	0,084	1,726
1000	0,943	0,137	6,883
1500	9,379	0,295	31,793
2000	40,803	0,662	61,636

Висновки

Виконуючи роботу комп’ютерного практикуму, було реалізовано алгоритми аналізу тексту та алгоритм множення матриць за допомогою ForkJoinPool та проведено дослідження ефективності деяких паралельних алгоритмів з їх послідовними аналогами.