МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

**Звіт**

З лабораторної роботи № 4 з дисципліни

«Технології паралельних обчислень»

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

| **Виконав(ла)** | *ІП-14 Бабіч Денис* |  |  |
| --- | --- | --- | --- |
|  | (шифр, прізвище, ім'я, по батькові) |  |  |

| **Перевірив** | *Дифучина О. Ю.* |  |  |
| --- | --- | --- | --- |
|  | (шифр, прізвище, ім'я, по батькові) |  |  |

Київ 2024

# ОСНОВНА ЧАСТИНА

**Мета роботи**: Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework.

1. Побудуйте алгоритм статистичного аналізу тексту та визначте характеристики випадкової величини «довжина слова в символах» з використанням ForkJoinFramework. Дослідіть побудований алгоритм аналізу текстових документів на ефективність експериментально.

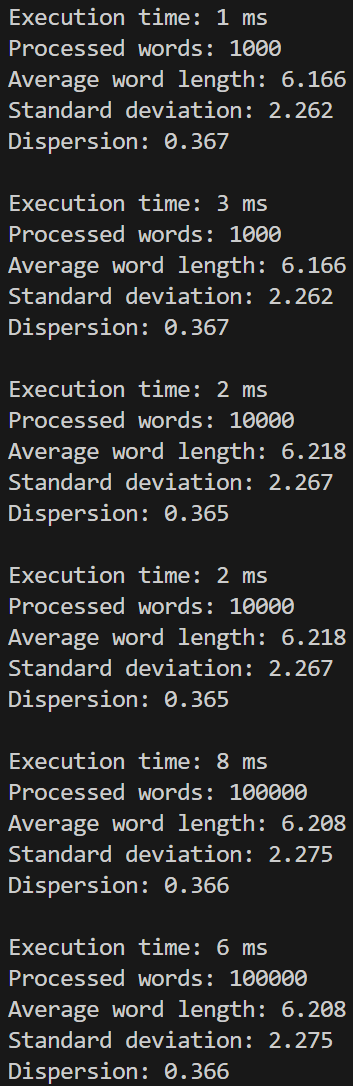


Рисунок 1.1 – Результати роботи алгоритму

**Лістинг методу getWordsStatsDefault**

public static WordsStats getWordsStatsDefault(Document document)

{

if (document == null)

{

throw new IllegalArgumentException("document cannot be null.");

}

long timestepStart = System.currentTimeMillis();

int totalLength = 0;

for (String word : document.getWords())

{

totalLength += word.length();

}

final double AVERAGE\_WORD\_LENGTH = (double)totalLength / document.getWordsCount();

double totalSquares = 0;

for (String word : document.getWords())

{

totalSquares += Math.pow(word.length() - AVERAGE\_WORD\_LENGTH, 2);

}

final double STANDARD\_DEVIATION = Math.sqrt(totalSquares / document.getWordsCount());

final double DISPERSION = STANDARD\_DEVIATION / AVERAGE\_WORD\_LENGTH;

WordsStats stats = new WordsStats(document.getWordsCount(), DISPERSION, STANDARD\_DEVIATION, AVERAGE\_WORD\_LENGTH);

long timestepEnd = System.currentTimeMillis();

long executionTime = timestepEnd - timestepStart;

System.out.println(String.format("Execution time: %d ms", executionTime));

return stats;

}

**Лістинг методу getWordsStatsEnhanced**

public static WordsStats getWordsStatsEnhanced(Document document)

{

if (document == null)

{

throw new IllegalArgumentException("document cannot be null.");

}

long timestepStart = System.currentTimeMillis();

final int TOTAL\_LENGTH = ForkJoinPool.commonPool().invoke(new TotalLengthTask(document, 0, document.getWordsCount()));

final double AVERAGE\_WORD\_LENGTH = (double)TOTAL\_LENGTH / document.getWordsCount();

final double TOTAL\_SQUARES = ForkJoinPool.commonPool().invoke(new TotalSquaresTask(AVERAGE\_WORD\_LENGTH, document, 0, document.getWordsCount()));

final double STANDARD\_DEVIATION = Math.sqrt(TOTAL\_SQUARES / document.getWordsCount());

final double DISPERSION = STANDARD\_DEVIATION / AVERAGE\_WORD\_LENGTH;

long timestepEnd = System.currentTimeMillis();

long executionTime = timestepEnd - timestepStart;

WordsStats stats = new WordsStats(document.getWordsCount(), DISPERSION, STANDARD\_DEVIATION, AVERAGE\_WORD\_LENGTH);

System.out.println(String.format("Execution time: %d ms", executionTime));

return stats;

}

**Лістинг класу TotalLengthTask**

package common;

import java.util.List;

import java.util.concurrent.RecursiveTask;

public final class TotalLengthTask extends RecursiveTask<Integer>

{

private static int Threshold = 100;

private final int INDEX\_END;

private final int INDEX\_START;

private final Document DOCUMENT;

public TotalLengthTask(Document document, int indexStart, int indexEnd)

{

this.DOCUMENT = document;

this.INDEX\_END = indexEnd;

this.INDEX\_START = indexStart;

}

@Override

protected Integer compute()

{

if ((this.INDEX\_END - this.INDEX\_START) <= TotalLengthTask.Threshold)

{

int totalLength = 0;

List<String> wordsSlice = this.DOCUMENT.getWordsSlice(this.INDEX\_START, this.INDEX\_END);

for (String word : wordsSlice)

{

totalLength += word.length();

}

return totalLength;

}

else

{

final int INDEX\_MID = (INDEX\_END + INDEX\_START) / 2;

TotalLengthTask firstTask = new TotalLengthTask(this.DOCUMENT, this.INDEX\_START, INDEX\_MID);

TotalLengthTask secondTask = new TotalLengthTask(this.DOCUMENT, INDEX\_MID, this.INDEX\_END);

firstTask.fork();

secondTask.fork();

return firstTask.join() + secondTask.join();

}

}

public static int getThreshold()

{

return TotalLengthTask.Threshold;

}

public static void setThreshold(int value)

{

TotalLengthTask.Threshold = value;

}

}

**Лістинг класу TotalSquaresTask**

package common;

import java.util.List;

import java.util.concurrent.RecursiveTask;

public class TotalSquaresTask extends RecursiveTask<Double>

{

private static int Threshold = 100;

private final int INDEX\_END;

private final int INDEX\_START;

private final Document DOCUMENT;

private final double AVERAGE\_WORD\_LENGTH;

public TotalSquaresTask(double averageWordLength, Document document, int indexStart, int indexEnd)

{

this.DOCUMENT = document;

this.INDEX\_END = indexEnd;

this.INDEX\_START = indexStart;

this.AVERAGE\_WORD\_LENGTH = averageWordLength;

}

@Override

protected Double compute()

{

if (this.INDEX\_END - this.INDEX\_START <= TotalSquaresTask.Threshold)

{

double totalSquares = 0;

List<String> wordsSlice = this.DOCUMENT.getWordsSlice(this.INDEX\_START, this.INDEX\_END);

for (String word : wordsSlice)

{

totalSquares += Math.pow(word.length() - this.AVERAGE\_WORD\_LENGTH, 2);

}

return totalSquares;

}

else

{

final int INDEX\_MID = (INDEX\_END + INDEX\_START) / 2;

TotalSquaresTask firstTask = new TotalSquaresTask(this.AVERAGE\_WORD\_LENGTH, this.DOCUMENT, this.INDEX\_START, INDEX\_MID);

TotalSquaresTask secondTask = new TotalSquaresTask(this.AVERAGE\_WORD\_LENGTH, this.DOCUMENT, INDEX\_MID, this.INDEX\_END);

firstTask.fork();

secondTask.fork();

return firstTask.join() + secondTask.join();

}

}

public static int getThreshold()

{

return TotalSquaresTask.Threshold;

}

public static void setThreshold(int value)

{

TotalSquaresTask.Threshold = value;

}

}

Таблиця 1.1 – Порівняння прискорення

| Кількість слів у файлі | Час виконання, мс | | Прискорення |
| --- | --- | --- | --- |
| Послідовний підхід | ForkJoinFramework |
| 1000 | 1 | 3 | 0.3 |
| 10000 | 2 | 2 | 1 |
| 100000 | 8 | 6 | 1.3 |

1. Реалізуйте один з алгоритмів комп’ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework.

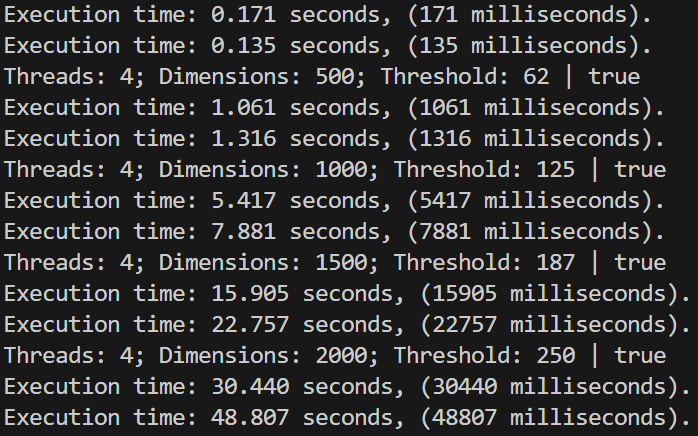


Рисунок 1.2 – Результати виконання алгоритму

Таблиця 1.2 – Отримані значення прискорення

| Matrix Size | Stripe algorithm (enhanced) | Stripe algorithm (default) | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 processors | | 9 processors | | 16 processors | |
| Time | Speed up | Time | Speed up | Time | Speed up |
| 500 | 0.171 | 0.157 | 1.08917 | 0.078 | 2.19231 | 0.092 | 1.8587 |
| 1000 | 1.061 | 1.092 | 0.97161 | 1.122 | 0.94563 | 1.21 | 0.87686 |
| 1500 | 5.417 | 8.032 | 0.67443 | 5.846 | 0.92662 | 5.298 | 1.02246 |
| 2000 | 15.905 | 22.19 | 0.71676 | 15.135 | 1.05088 | 13.941 | 1.14088 |
| 2500 | 30.44 | 45.03 | 0.67599 | 33.438 | 0.91034 | 28.5 | 1.06807 |
| 3000 | 60.544 | 93.29 | 0.64899 | 58.878 | 1.0283 | 54.36 | 1.11376 |

**Лістинг методу multiplyStripedWithActions**

public static Result multiplyStripedWithActions(MatrixInt matrix1, MatrixInt matrix2)

{

if (!MatrixInt.areMultipliable(matrix1, matrix2))

{

throw new IllegalArgumentException("Matrices are not multipliable.");

}

MatrixInt resultingMatrix = new MatrixInt(matrix1.rows, matrix2.columns);

long timestepStart = System.currentTimeMillis();

ForkJoinPool.commonPool().invoke(new StripedMultiplicationTask(matrix1, matrix2, resultingMatrix, 0, matrix1.rows \* matrix2.columns));

long timestepEnd = System.currentTimeMillis();

long executionTime = timestepEnd - timestepStart;

return new Result(resultingMatrix, executionTime);

}

**Лістинг класу StripedMultiplicationTask**

public class StripedMultiplicationTask extends RecursiveAction {

private static int Threshold = 100;

private final MatrixInt MATRIX\_1;

private final MatrixInt MATRIX\_2;

private final MatrixInt MATRIX\_RESULT;

private final int INDEX\_START;

private final int INDEX\_FINISH;

public StripedMultiplicationTask(MatrixInt matrix1, MatrixInt matrix2, MatrixInt result, int indexStart, int indexFinish) {

this.MATRIX\_1 = matrix1;

this.MATRIX\_2 = matrix2;

this.MATRIX\_RESULT = result;

this.INDEX\_START = indexStart;

this.INDEX\_FINISH = indexFinish;

}

@Override

protected void compute() {

if (this.INDEX\_FINISH - this.INDEX\_START <= StripedMultiplicationTask.Threshold) {

for (int i = this.INDEX\_START; i < this.INDEX\_FINISH; ++i)

{

final int INDEX\_ROW = i / this.MATRIX\_1.getColumns();

final int INDEX\_COLUMN = i % this.MATRIX\_2.getColumns();

final int[] ROW = this.MATRIX\_1.getRow(INDEX\_ROW);

final int[] COLUMN = this.MATRIX\_2.getColumn(INDEX\_COLUMN);

int result = 0;

for (int j = 0; j < ROW.length; ++j) {

result += ROW[j] \* COLUMN[j];

}

this.MATRIX\_RESULT.set(INDEX\_ROW, INDEX\_COLUMN, result);

}

}

else {

final int INDEX\_MID = (this.INDEX\_START + this.INDEX\_FINISH) / 2;

StripedMultiplicationTask leftTask = new StripedMultiplicationTask(this.MATRIX\_1, this.MATRIX\_2, this.MATRIX\_RESULT, this.INDEX\_START, INDEX\_MID);

StripedMultiplicationTask rightTask = new StripedMultiplicationTask(this.MATRIX\_1, this.MATRIX\_2, this.MATRIX\_RESULT, INDEX\_MID, this.INDEX\_FINISH);

ForkJoinTask.invokeAll(leftTask, rightTask);

}

}

public static int getThreshold() {

return StripedMultiplicationTask.Threshold;

}

public static void setThreshold(int value) {

StripedMultiplicationTask.Threshold = value;

}

}

1. Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework.

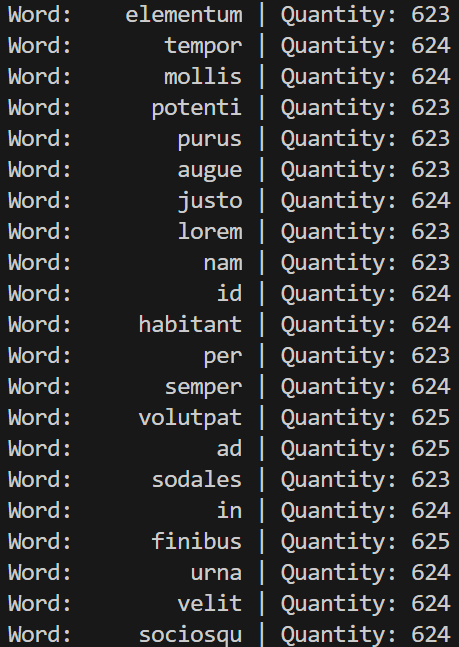


Рисунок 1.3 – Результат роботи алгоритму

**Лістинг методу getCommonWords**

public static HashMap<String, Integer> getCommonWords(Document... documents)

{

HashMap<String, Integer> commonWords = new HashMap<>();

for (Document document : documents)

{

HashMap<String, Integer> documentWords = ForkJoinPool.commonPool().invoke(new CommonWordsTask(document, 0, document.getWordsCount()));

if (commonWords.isEmpty())

{

commonWords.putAll(documentWords);

}

else

{

HashMap<String, Integer> newCommonWords = new HashMap<>();

for (String word : commonWords.keySet())

{

if (documentWords.containsKey(word))

{

newCommonWords.put(word, commonWords.get(word) + documentWords.get(word));

}

}

commonWords = newCommonWords;

}

}

return commonWords;

}

**Лістинг класу CommonWordsTask**

final class CommonWordsTask extends RecursiveTask<HashMap<String, Integer>>{

private static int Threshold = 100;

private final Document DOCUMENT;

private final int INDEX\_START;

private final int INDEX\_FINISH;

public CommonWordsTask(Document document, int indexStart, int indexFinsih) {

this.DOCUMENT = document;

this.INDEX\_START = indexStart;

this.INDEX\_FINISH = indexFinsih;

}

@Override

protected HashMap<String, Integer> compute() {

HashMap<String, Integer> documentWords = new HashMap<>();

if ((this.INDEX\_FINISH - this.INDEX\_START) < CommonWordsTask.Threshold) {

List<String> words = this.DOCUMENT.getWordsSlice(this.INDEX\_START, this.INDEX\_FINISH);

for (String word : words) {

documentWords.compute(word, (key, value) -> (value == null) ? 1 : value + 1);

}

}

else {

final int INDEX\_MID = this.INDEX\_START + (this.INDEX\_FINISH - this.INDEX\_START) / 2;

CommonWordsTask firstTask = new CommonWordsTask(this.DOCUMENT, this.INDEX\_START, INDEX\_MID);

CommonWordsTask secondTask = new CommonWordsTask(this.DOCUMENT, INDEX\_MID, this.INDEX\_FINISH);

ForkJoinTask.invokeAll(firstTask, secondTask);

HashMap<String, Integer> firstResult = firstTask.join();

HashMap<String, Integer> secondResult = secondTask.join();

for (Map.Entry<String, Integer> entry : firstResult.entrySet()) {

documentWords.put(entry.getKey(), entry.getValue());

}

for (Map.Entry<String, Integer> entry : secondResult.entrySet()) {

documentWords.merge(entry.getKey(), entry.getValue(), Integer::sum);

}

}

return documentWords;

}

public static int getThreshold() {

return CommonWordsTask.Threshold;

}

public static void setThreshold(int value) {

CommonWordsTask.Threshold = value;

}

}

1. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework.



Рисунок 1.4 – Результат роботи алгоритма

**Лістинг методу getDocumentsWithKeyWords**

public static List<Document> getDocumentsWithKeyWords(List<String> keywords, Document... documents) {

List<Document> result = new ArrayList<>();

for (Document document : documents) {

if (ForkJoinPool.commonPool().invoke(new KeywordsSearchTask(document, keywords, 0, document.getWordsCount()))) {

result.add(document);

}

}

return result;

}

**Лістинг класу KeywordsSearchTask**

package common;

import java.util.List;

import java.util.concurrent.ForkJoinTask;

import java.util.concurrent.RecursiveTask;

public class KeywordsSearchTask extends RecursiveTask<Boolean>

{

private static int Threshold = 100;

private static int MinRequiredKeywordsCount = 5;

private final Document DOCUMENT;

private final List<String> KEYWORDS;

private final int INDEX\_START;

private final int INDEX\_FINISH;

public KeywordsSearchTask(Document document, List<String> keywords, int indexStart, int indexFinish)

{

this.DOCUMENT = document;

this.KEYWORDS = keywords;

this.INDEX\_START = indexStart;

this.INDEX\_FINISH = indexFinish;

}

@Override

protected Boolean compute()

{

if (this.INDEX\_FINISH - this.INDEX\_START < Threshold)

{

int count = 0;

List<String> words = this.DOCUMENT.getWordsSlice(this.INDEX\_START, this.INDEX\_FINISH);

for (String word : words)

{

for (String keyword : KEYWORDS)

{

if (word.toLowerCase().contains(keyword.toLowerCase()))

{

++count;

if (count >= KeywordsSearchTask.MinRequiredKeywordsCount)

{

return true;

}

}

}

}

return false;

}

else

{

final int INDEX\_MID = this.INDEX\_START + (this.INDEX\_FINISH - this.INDEX\_START) / 2;

KeywordsSearchTask firstTask = new KeywordsSearchTask(this.DOCUMENT, this.KEYWORDS, this.INDEX\_START, INDEX\_MID);

KeywordsSearchTask secondTask = new KeywordsSearchTask(this.DOCUMENT, this.KEYWORDS, INDEX\_MID, this.INDEX\_FINISH);

ForkJoinTask.invokeAll(firstTask, secondTask);

return firstTask.join() || secondTask.join();

}

}

public static int getThreshold()

{

return KeywordsSearchTask.Threshold;

}

public static void setThreshold(int value)

{

KeywordsSearchTask.Threshold = value;

}

public static int getMinRequiredKeywordsCount()

{

return KeywordsSearchTask.MinRequiredKeywordsCount;

}

public static void setMinRequiredKeywordsCount(int value) {

KeywordsSearchTask.MinRequiredKeywordsCount = value;

}

}

# ВИСНОВКИ

В ході виконання цієї лабораторної роботи було розроблено та реалізовано кілька алгоритмів з використанням ForkJoinFramework. Таким чином, був створений алгоритм статистичного аналізу тексту, який дозволив визначити характеристики випадкової величини «довжина слова в символах», де експериментальне дослідження показало ефективність побудованого алгоритму аналізу текстових документів з великою кількістю слів. Також було реалізовано алгоритм стрічкового множення матриць та було визначено прискорення, яке отримане за рахунок використання ForkJoinFramework. Додатково, було розроблено та реалізовано алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework і алгоритм пошуку текстових документів, які відповідають заданим ключовим словам.

Таким чином, використання ForkJoinFramework дозволило оптимізувати процес алгоритми, покращити продуктивність та забезпечити коректність роботи програм. Завдяки паралельному виконанню завдань було можливо значно прискорити обробку великих обсягів даних.