**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

кафедра ЕОМ



***Лабораторна робота №3***

**з дисципліни “ Паралельні та розподілені обчислення “**

**на тему:**

**“ МОЖЛИВОСТІ ВИКОРИСТАННЯ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ.”**

Виконав: ст.гр. КІ-33

Кулик В.Я.

Прийняв:

Козак Н.Б.

Львів – 2020

**Мета:** Дослідити можливості розв’язання різноманітних задач за допомогою паралельних алгоритмів. Навчитися виділяти незалежні гілки обчислень та виконувати їх паралельно.

**Варіант №8**

**Завдання:** В полі 8\*8 кліток зображено кілька прямокутників, кожен з яких складається з кліток, різні прямокутники не перетинаються і не доторкаються один до одного. Задана квадратна матриця порядку 8, в якій елемент рівний нулю, якщо відповідна клітина належить прямокутнику і відмінний від нуля, в іншому випадку. Визначити кількість прямокутників. Початковими даними вважати матрицю елементів, яка повинна вводитися під час виконання програми. Графічно відобразити вхідні дані.

**Хід роботи:**

У даному завданні подано матрицю 8х8 тому доцільно використовувати двохвимірну матрицю, кожна позиція у якій буде заповнена (1) або не заповнена (0).

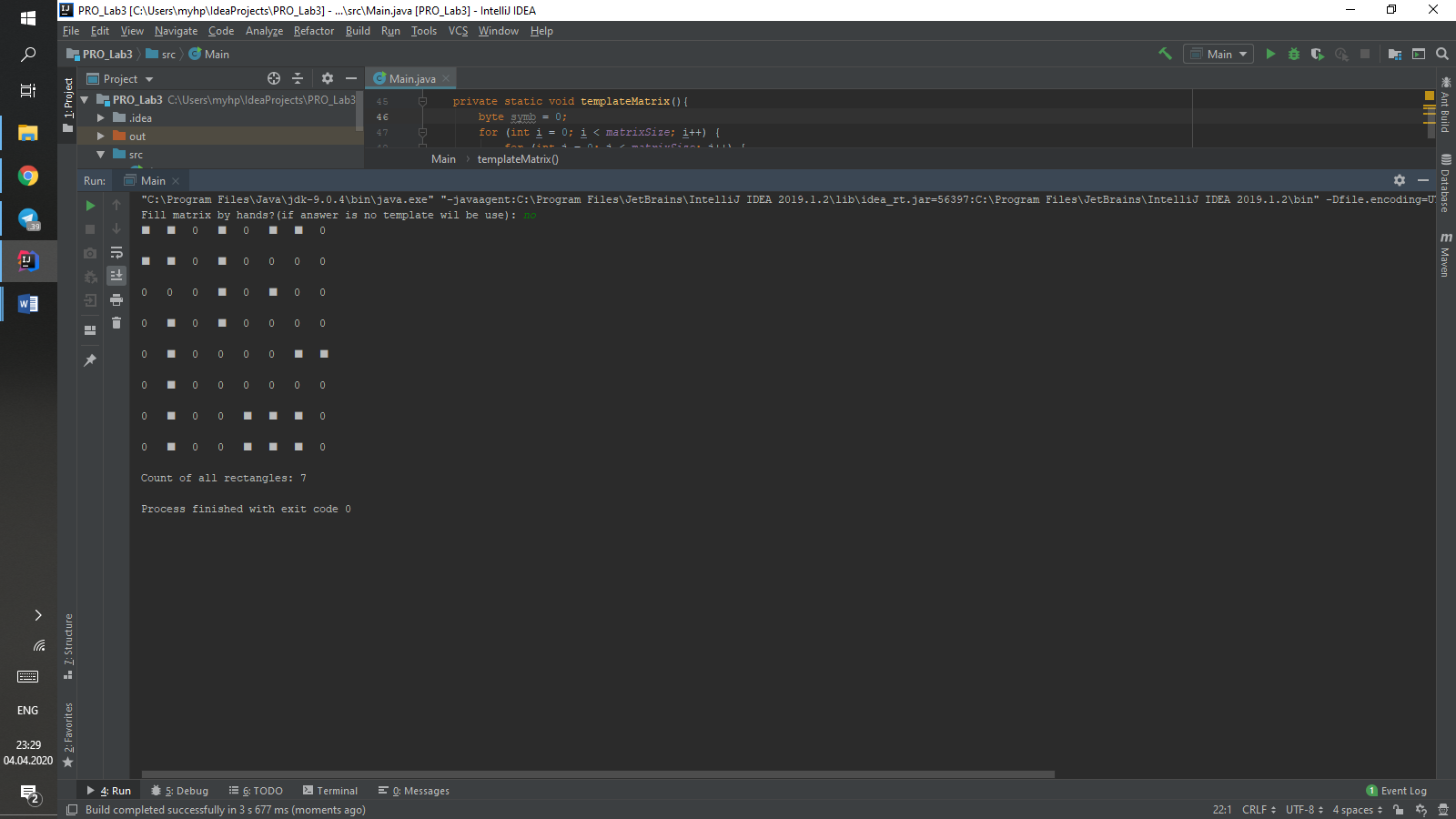
Оскільки прямокутники не перетинаються і не доторкаються то між ними відповідно завжди буде проміжок, який можна не враховувати при вичесленнях. Для цього буде доцільно використовувати список, у який помістити значення всіх можливих точок, що містять значення відмінне від 0. Коли прямокутник знайдений і знайдені всі його межі, зі списку необхідно видаляти розміщення всіх точок, що належать самому прямокутнику та всі значення що знаходяться навколо прямокутника, бо вони обов'язково не будуть містити значення відмінні від 0. Це однозначно пришвидшує роботу програми, але і збільшує її складність.

Для пошуку хоча б однієї точки необхідно використовувати це й же ж список і перебирати значення, до кінця списку поки не буде знайдена хоча б одна точка, що містить в собі значення відмінне від 0. Коли точка знайдена запам'ятовуються всі необхідні вхідні дані та запускаються 2 потоки що визчають довжину і ширину прямокутника. Коли вони одночасно відпрацювали та повернули результат, ці дані запамятовуються та порівнюються. Настувним кроком є перевірка чи прямокутник цілисний і правильний. Для цього, якщо прямокутник довший аніж ширший, то створюються стільки потоків скільки є стовпців, а якщо прямокутник ширший ніж довший то відповідно створюється така кількість потоків яка дорівнює кількості рядків. Далі кожен потік відпрацьовує в своєму рядку чи стовпці та перевіряє наявність усіх клітинок, що повинні бути заповнені, у інакшому випадку, прямокутник не є цілісним а отже і правильним, повідомлення з помилкою виводиться на екран та програма зупиняє свою роботу, оскільки вхідні дані є невірними. Якщо ж всі потоки відпрацювали та повернули значення TRUE то прямокутник знайдений, лічильник збільшується, а всі клітинки що тепер не можуть містити значення відмінне від 0 та клітинки, що складали прямокутник, видаляються. Ці кроки продовжуються поки список не буде пустим та не будуть знайдені всі прямокутники.

Оскільки потоки можуть працювати одночасно на ОС, то пошук відповідності у кожному рядку чи стовпці проводиться паралельно, що значно пришвидшує роботу програми взагалому. При розширенні матриці ця продуктивність буде швидко зростати в порівнянні з лінійним алгоритмом.

Загальна складність мого алгоритму: O(N^2), оскільки в мене тільки 2 вкладені цикли.

А часові затрати зі збільшенням розмірності будуть наближатись до значення 1/N, в порівнянні з лінійним. При умові достатньо великого N, де N >> часу створення потоку.



*Рис. 1 Результат виконання програми*

Висновок: на цій лабораторній роботі дослідив можливості розв’язання різноманітних задач за допомогою паралельних алгоритмів та навчитися виділяти незалежні гілки обчислень та виконувати їх паралельно.

Додаток (Текст програми):

import java.util.LinkedList;  
import java.util.List;  
import java.util.Scanner;  
  
public class Main {  
  
 private static int *matrixSize* = 8;  
 private static List<Integer> *possCell* = new LinkedList<Integer>();  
 private static char[][] *matrix* = new char[*matrixSize*][*matrixSize*];  
 private static Scanner *sc* = new Scanner(System.*in*);  
 private static final char *symbol* = (char)9632;  
  
 private static void initMatrix(){  
 for (int i = 0; i < *matrixSize*; i++) {  
 for (int j = 0; j < *matrixSize*; j++) {  
 *possCell*.add(i\*10+j);  
 }  
 }  
 }  
  
 private static void fillMatrix(){  
 byte symb = 0;  
 System.*out*.println("Enter matrix(1 or 0): ");  
 for (int i = 0; i < *matrixSize*; i++) {  
 for (int j = 0; j < *matrixSize*; j++) {  
 System.*out*.print(String.*format*("%d%d", i,j) + ": ");  
 symb = *sc*.nextByte();  
 if(symb == 0){  
 *matrix*[i][j] = '0';  
 } else if(symb == 1){  
 *matrix*[i][j] = *symbol*;  
 } else{  
 System.*out*.println("Enter 1 or 0!!!");  
 if(j != 0){  
 j--;  
 } else{  
 i--;  
 j = *matrixSize* - 1;  
 }  
 }  
 }  
 }  
 }  
  
 private static void templateMatrix(){  
 byte symb = 0;  
 System.*out*.println("Enter matrix(1 or 0): ");  
 for (int i = 0; i < *matrixSize*; i++) {  
 for (int j = 0; j < *matrixSize*; j++) {  
 *matrix*[i][j] = '0';  
 }  
 }  
  
 *matrix*[0][0] = *symbol*;  
 *matrix*[0][1] = *symbol*;  
 *matrix*[0][3] = *symbol*;  
 *matrix*[0][5] = *symbol*;  
 *matrix*[0][6] = *symbol*;  
 *matrix*[1][0] = *symbol*;  
 *matrix*[1][1] = *symbol*;  
 *matrix*[1][3] = *symbol*;  
 *matrix*[2][3] = *symbol*;  
 *matrix*[2][5] = *symbol*;  
 *matrix*[3][1] = *symbol*;  
 *matrix*[3][3] = *symbol*;  
 *matrix*[4][1] = *symbol*;  
 *matrix*[4][6] = *symbol*;  
 *matrix*[4][7] = *symbol*;  
 *matrix*[5][1] = *symbol*;  
 *matrix*[6][1] = *symbol*;  
 *matrix*[6][4] = *symbol*;  
 *matrix*[6][5] = *symbol*;  
 *matrix*[6][6] = *symbol*;  
 *matrix*[7][1] = *symbol*;  
 *matrix*[7][4] = *symbol*;  
 *matrix*[7][5] = *symbol*;  
 *matrix*[7][6] = *symbol*;  
 }  
  
 private static void printMatrix(){  
 for (int i = 0; i < *matrixSize*; i++) {  
 for (int j = 0; j < *matrixSize*; j++) {  
 System.*out*.print(String.*format*("%c\t", *matrix*[i][j]));  
 }  
 System.*out*.println("\n");  
 }  
 }  
  
 private static boolean checkLine(int start, int end, int line, boolean isColumn){  
 final boolean[] result = {true};  
  
 Thread thread = new Thread(() -> {  
 for (int k = start; k < end; k++) {  
 if(isColumn){  
 if(*matrix*[k][line] != *symbol*){  
 result[0] = false;  
 break;  
 }  
 } else {  
 if(*matrix*[line][k] != *symbol*){  
 result[0] = false;  
 break;  
 }  
 }  
 }  
 });  
 thread.start();  
  
 return result[0];  
 }  
  
 public static void main(String[] args) throws InterruptedException {  
 *initMatrix*();  
 System.*out*.print("Fill matrix by hands?(if answer is no template wil be use): ");  
 String answer = *sc*.next();  
 if(answer.equals("yes")){  
 *fillMatrix*();  
 } else {  
 *templateMatrix*();  
 }  
  
 *printMatrix*(); //End of initialization  
  
 int recCount = 0;  
  
 while(!*possCell*.isEmpty()){  
 int cell = *possCell*.get(0);  
  
 int startI = cell/10;  
 int startJ = cell - startI\*10;  
  
 if(*matrix*[startI][startJ] == *symbol*){  
 final int[] endI = {startI};  
 final int[] endJ = {startJ};  
  
 int finalEndJ = endJ[0];  
 Thread thread1 = new Thread(() -> {  
 while(endI[0] < *matrixSize*){  
 if(*matrix*[endI[0]][finalEndJ] != *symbol*){  
 break;  
 }  
 endI[0]++;  
 }});  
 thread1.start();  
  
 Thread thread2 = new Thread(() -> {  
 while (endJ[0] < *matrixSize*){  
 if(*matrix*[startI][endJ[0]] != *symbol*){  
 break;  
 }  
 endJ[0]++;  
 }});  
 thread2.start();  
  
 boolean res = true;  
  
 thread1.join();  
 thread2.join();  
 if(endI[0] - startI > endJ[0] - startJ){  
 //More rows than columns  
 for(int n = startI+1; n < endI[0]; n++){  
 if(!*checkLine*(startJ, endJ[0], n, false)){  
 res = false;  
 break;  
 }  
 }  
 }else{  
 //More columns than rows  
 for(int n = startJ+1; n < endJ[0]; n++){  
 if(!*checkLine*(startI, endI[0], n, true)){  
 res = false;  
 break;  
 }  
 }  
 }  
  
 if(res){  
 //Checking is positive  
 recCount++;  
 for (int i = startI - 1; i <= endI[0]; i++) {  
 for (int j = startJ - 1; j <= endJ[0]; j++) {  
 if(i >= 0 && j >= 0){  
 int el = i\*10+j;  
 *possCell*.remove(Integer.*valueOf*(el));  
 }  
 }  
 }  
 }else{  
 System.*out*.println("Input field is incorrect!!!!");  
 break;  
 }  
 }  
 else{  
 *possCell*.remove(*possCell*.get(0));  
 }  
 }  
 System.*out*.println("Count of all rectangles: " + recCount);  
 }  
}