Національний Технічиний Університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №1

Роботу виконала:

Желєзнова В.С.

Група: ДА­-81

Перевірив: Яременко В. С.

Київ – 2021

ЗМІСТ

[Мета роботи 3](#_Toc64843214)

[Завдання 3](#_Toc64843215)

[1. Лістинг програми мовою Java 4](#_Toc64843216)

[2. Порівняння реалізацій 7](#_Toc64843217)

[3. Порівняння часу виконання 11](#_Toc64843218)

[Висновки 12](#_Toc64843219)

Мета роботи

Розробка та реалізація паралельного алгоритму для задач із паралелізмом даних.

Завдання

1. Розробити послідовну та багатопоточну програми, які реалізують варіант індивідуального завдання (мова програмування обирається студентом).
2. Порівняти правильність виконання, порівнявши послідовний та паралельний розв’язки.
3. Виміряти час розрахунку для послідовного та паралельного розв’язків при різних значеннях SIZE та NUMBER\_THREADS для власного варіанту, знайти значення, при яких кожен із розв’язків (послідовний чи паралельний) буде виконуватися швидше за інший, зробити таблицю та графічно представити результат.

**Варіант 10.**

Створити двовимірний масив розмірності n×m. Знайти максимальне та мінімальне значення масиву. (n, m ≥ 100).

1. Лістинг програми мовою Java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package com.company;  
  
import java.util.Arrays;  
  
public class ThreadWork {  
 public static final int *ROWS* = 100;  
 public static final int *COLS* = 100;  
 public static final int *NUMBER\_THREADS* = 4;  
 public static final int *RAND\_MIN* = 1;  
 public static final int *RAND\_MAX* = 250000;  
 public static final int *TESTS* = 10;  
 public static final int *TABLE\_HEADER* = 2;  
 public static final int *TABLE\_FOOTER* = 2;  
  
 public static void main(String[] args) throws InterruptedException {  
  
 final Object[][] resultsTable = new String[*TABLE\_HEADER* + *TESTS* + *TABLE\_FOOTER*][];  
 resultsTable[0] = new String[]{"", "Serial", "", "", "Parallel", ""};  
 resultsTable[1] = new String[]{"min", "max", "time (us)", "min", "max", "time (us)"};  
 long[] serialTimes = new long[*TESTS*];  
 long[] parallelTimes = new long[*TESTS*];  
  
 for (int test\_i = 0; test\_i < *TESTS*; test\_i++) {  
 int[][] array = new int[*ROWS*][*COLS*];  
 for (int i = 0; i < *ROWS*; i++) {  
 for (int j = 0; j < *COLS*; j++) {  
 array[i][j] = *RAND\_MIN* + (int) (Math.*random*() \* *RAND\_MAX*);  
 }  
 }  
 long start = System.*nanoTime*();  
  
 int serialMax = *RAND\_MIN* - 1;  
 int serialMin = *RAND\_MAX* + 1;  
 int localMax;  
 int localMin;  
 for (int i = 0; i < *ROWS*; i++) {  
 localMax = Arrays.*stream*(array[i]).max().orElseThrow();  
 localMin = Arrays.*stream*(array[i]).min().orElseThrow();  
 if (localMax > serialMax) {  
 serialMax = localMax;  
 }  
 if (localMin < serialMin) {  
 serialMin = localMin;  
 }  
 }  
 long end = System.*nanoTime*();  
 long serialTime = (end - start) / 1000;  
 serialTimes[test\_i] = serialTime;  
  
  
 start = System.*nanoTime*();  
 ThreadCalc[] ThreadArray = new ThreadCalc[*NUMBER\_THREADS*];  
  
 for (int i = 0; i < *NUMBER\_THREADS*; i++) {  
 ThreadArray[i] = new ThreadCalc(array,  
 *ROWS* / *NUMBER\_THREADS* \* i,  
 i == (*NUMBER\_THREADS* - 1) ? *ROWS* : *ROWS* / *NUMBER\_THREADS* \* (i + 1));  
 ThreadArray[i].start();  
 }  
 for (int i = 0; i < *NUMBER\_THREADS*; i++) {  
 ThreadArray[i].join();  
 }  
  
 int parallelMax = *RAND\_MIN* - 1;  
 int parallelMin = *RAND\_MAX* + 1;  
 for (int i = 0; i < *NUMBER\_THREADS*; i++) {  
 localMax = ThreadArray[i].getMax();  
 localMin = ThreadArray[i].getMin();  
 if (localMax > parallelMax) {  
 parallelMax = localMax;  
 }  
 if (localMin < parallelMin) {  
 parallelMin = localMin;  
 }  
 }  
  
 end = System.*nanoTime*();  
  
 long parallelTime = (end - start) / 1000;  
 parallelTimes[test\_i] = parallelTime;  
  
 resultsTable[*TABLE\_HEADER* + test\_i] = new String[]{String.*valueOf*(serialMin), String.*valueOf*(serialMax), String.*valueOf*(serialTime), String.*valueOf*(parallelMin), String.*valueOf*(parallelMax), String.*valueOf*(parallelTime)};  
 }  
  
 resultsTable[*TABLE\_HEADER* + *TESTS*] = new String[]{"", "", "", "", "", ""};  
 resultsTable[*TABLE\_HEADER* + *TESTS* + 1] = new String[]{"", "Average: ", String.*valueOf*(Arrays.*stream*(serialTimes).average().orElse(-1)), "", "Average:", String.*valueOf*(Arrays.*stream*(parallelTimes).average().orElse(-1))};  
 System.*out*.println("Rows = " + String.*valueOf*(*ROWS*));  
 System.*out*.println("Cols = " + String.*valueOf*(*COLS*));  
 System.*out*.println("Number of threads = " + String.*valueOf*(*NUMBER\_THREADS*));  
 System.*out*.println("");  
  
 for (final Object[] row: resultsTable) {  
 System.*out*.format("%10s%10s%10s%10s%10s%10s%n", row);  
  
 }  
 }  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Лістинг 1. ThreadSample.java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import java.util.Arrays;  
  
class ThreadCalc extends Thread{  
  
 int[][] array;  
 int startIndex;  
 int endIndex;  
 int max = ThreadWork.*RAND\_MIN* - 1;  
 int min = ThreadWork.*RAND\_MAX* + 1;  
  
 public ThreadCalc(int[][] array, int startIndex, int endIndex) { this.array = array;  
 this.startIndex = startIndex;  
 this.endIndex = endIndex;  
 }  
  
  
 public int getMax() {  
 return max;  
 }  
 public int getMin() { return min; }  
  
 @Override  
 public void run(){  
 int localMax;  
 int localMin;  
 for(int i = startIndex; i<endIndex; i++ ){  
 localMax= Arrays.*stream*(array[i]).max().orElseThrow();  
 localMin= Arrays.*stream*(array[i]).min().orElseThrow();  
 if (localMax > max){  
 max = localMax;  
 }  
 if (localMin < min) {  
 min = localMin;  
 }  
 }  
 }  
}

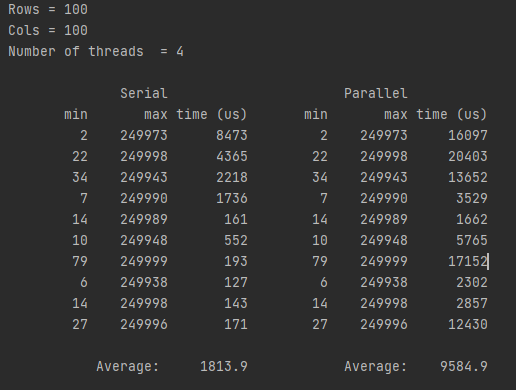
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Лістинг 2. ThreadCalc.java

1. Порівняння реалізацій

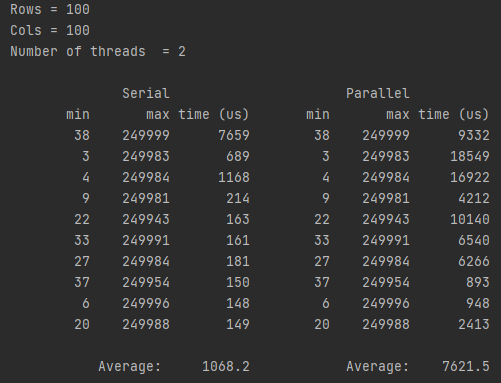
При значеннях:

ROWS = 100; COLS = 100; NUMBER\_THREADS = 4;

Рисунок 2.1 Таблиця з порівнянням для матриці 100×100 та 4-ма потоками

При значеннях:

ROWS = 100; COLS = 100; NUMBER\_THREADS = 2;

Рисунок 2.2 Таблиця з порівнянням для матриці 100×100 та 2-ма потоками

При значеннях:

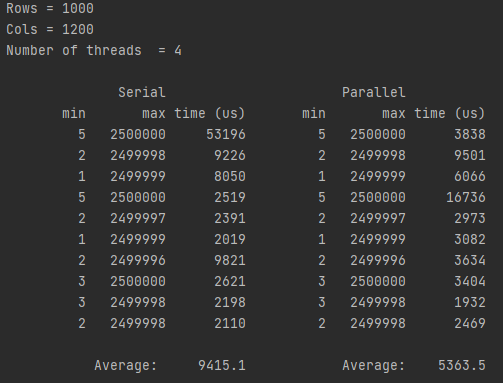
ROWS = 1000; COLS = 1200; NUMBER\_THREADS = 4;

Рисунок 2.3 Таблиця з порівнянням для матриці 1000×1200 та 4-ма потоками

При значеннях:

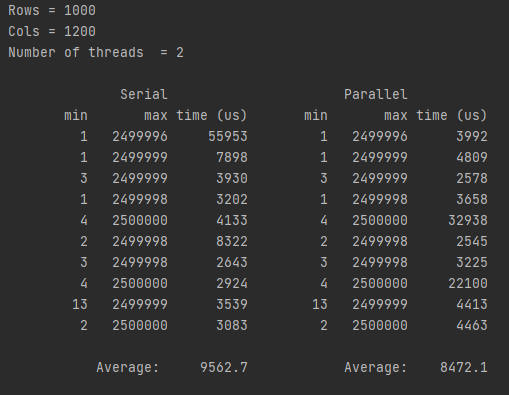
ROWS = 1000; COLS = 1200; NUMBER\_THREADS = 2;

Рисунок 2.4 Таблиця з порівнянням для матриці 1000×1200 та 2-ма потоками

1. Порівняння часу виконання

Таблиця 3.1 Порівняння часу виконання алгоритму

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ROWS | COLS | Serial | 2 Threads | 4 Threads |
| 100 | 120 | 1234.1 | 5239.9 | 4651.7 |
| 1000 | 1200 | 9562.7 | 8472.1 | 5363.5 |
| 2500 | 3000 | 20388.3 | 11229.2 | 10398.0 |
| 5000 | 6000 | 54398.3 | 29600.9 | 19759.3 |
| 10000 | 12000 | 183199.9 | 89374.0 | 54575.9 |

Рисунок 3.1 Графічне зображення порівняння часу виконання алгоритму

Висновки

У ході виконання даної лабораторної роботи я розробила програму, що реалізує варіант мого завдання як послідовно, так і паралельно, а саме пошук максимального та мінімального числа у масиві розмірністю n×m (n, m ≥ 100). Також було проведено порівняння результатів аби впевнитися, що результати, отримані при розв’язанні послідовно та паралельно однакові.

У результаті аналізу виконаної роботи було отримано таблицю залежності часу виконання від об’єму вхідних даних та кількості потоків, якими було розв’язано завдання. Також цю залежність було зображено графічно. Підсумовуючи проведений аналіз можу сказати, що використання декількох потоків при малих об’ємах даних недоцільно, оскільки прискорення виконання завдання спостерігається починаючи з масиву 1000×1200 елементів цілочисельного типу, а саме 4,5МБайтів даних.

Також, у зв’язку з моєю неуважністю до підрахунку об’єму даних мені довелося стикнутися з помилкою пам’яті, адже я намагалася використати масив 100000×120000, що еквівалентно 45ГБайтам пам’яті.