Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до лабораторної роботи № 1

з дисципліни

*Паралельні обчислення та розподілені системи*

на тему:

**“Способи розпаралелювання та організації обчислень. Послідовні алгоритми.”**

Виконав: студент ОІ-32

Світящук Н. О.

Прийняв: асистент каф. АСУ

Бадзь В.М.

Львів – 2025

**Мета**: оволодіти практичними прийомами розробки алгоритмів та програм із застосуванням ітерації.

Послідовність роботи

1. Використовуючи послідовні алгоритми, написати програму розв’язання індивідуального завдання.
2. При створенні програми намагатись забезпечити якнайбільшу незалежність програмного коду від операційної системи та середовища програмування.
3. Передбачити можливості:

* формування вхідних даних заданого розміру, наприклад, за допомогою
* генератора випадкових чисел;
* збереження вхідних даних у файлі із заданою назвою;
* зчитування вхідних даних із заданого файлу;
* виведення результатів на екран або у файл.

1. Відлагодити програму на прикладі з невеликим об’ємом вхідних даних, результати для якого можуть бути перевірені перерахунком поза програмою.
2. Підготувати приклад вхідних даних, для якого час на розв’язання задачі складатиме приблизно 5 секунд і перевірити програму на цьому прикладі.
3. Визначити часові характеристики роботи програми (сумарний час на виконання обчислень, не враховуючи формування вхідних даних, введення та виведення).
4. Розв’язати те ж саме завдання з використанням паралельних обчислень (multithreading). Повторити пункт 6 для цього варіанту програми.

Індивідуальне завдання

Варіант №18

Впорядкувати числовий масив за методом вибору.

Текст програми на Java

package nulp.laboratorywork;  
  
import java.io.\*;  
import java.util.\*;  
  
public class SelectionSort {  
 public static void selectionSort(int[] arr) {  
 int n = arr.length;  
 for (int i = 0; i < n - 1; i++) {  
 int minIdx = i;  
 for (int j = i + 1; j < n; j++) {  
 if (arr[j] < arr[minIdx]) {  
 minIdx = j;  
 }  
 }  
 int temp = arr[minIdx];  
 arr[minIdx] = arr[i];  
 arr[i] = temp;  
 }  
 }  
  
 public static void parallelSelectionSort(int[] arr, int[] arr2) throws InterruptedException {  
 Thread thread1= new Thread(() -> selectionSort(arr));  
 Thread thread2= new Thread(() -> selectionSort(arr2));  
  
 thread1.start();  
 thread2.start();  
  
 thread1.join();  
 thread2.join();  
 }  
  
 public static int[] generateRandomArray(int size, int bound) {  
 Random rand = new Random();  
 int[] arr = new int[size];  
 for (int i = 0; i < size; i++) {  
 arr[i] = rand.nextInt(bound);  
 }  
 return arr;  
 }  
  
 public static void saveArrayToFile(int[] arr, String filename) throws IOException {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {  
 for (int num : arr) {  
 writer.write(num + " ");  
 }  
 }  
 }  
  
 public static int[] readArrayFromFile(String filename) throws IOException {  
 List<Integer> list = new ArrayList<>();  
 try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {  
 String[] numbers = reader.readLine().trim().split(" ");  
 for (String num : numbers) {  
 list.add(Integer.parseInt(num));  
 }  
 }  
 return list.stream().mapToInt(i -> i).toArray();  
 }  
  
 public static void main(String[] args) {  
 String filename = "data.txt";  
 int size = 100;  
 int bound = 1000;  
  
 try {  
 int[] arr = generateRandomArray(size, bound);  
 saveArrayToFile(arr, filename);  
 arr = readArrayFromFile(filename);  
 int[] arr2 = readArrayFromFile(filename);  
  
 long startTime = System.nanoTime();  
 selectionSort(arr);  
 selectionSort(arr2);  
 long endTime = System.nanoTime();  
 saveArrayToFile(arr, "sorted\_arr1");  
 saveArrayToFile(arr2, "sorted\_arr2");  
 System.out.println("Послідовне сортування завершено. Час виконання: "  
 + (endTime - startTime) / 1e6 + " мс");  
  
 arr = readArrayFromFile(filename);  
 arr2 = readArrayFromFile(filename);  
 startTime = System.nanoTime();  
 parallelSelectionSort(arr, arr2);  
 endTime = System.nanoTime();  
 saveArrayToFile(arr, "parallel\_sorted\_arr1");  
 saveArrayToFile(arr2, "parallel\_sorted\_arr2");  
 System.out.println("Паралельне сортування завершено. Час виконання: "  
 + (endTime - startTime) / 1e6 + " мс");  
 } catch (IOException | InterruptedException e) {  
 System.out.println("Йой, неочікувана помилка...");  
 }  
 }  
}

Покликання на Github: <https://github.com/onsviii/Laboratory-works-PCDS>

Результат

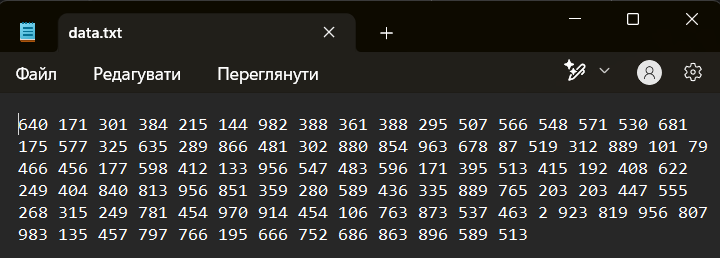


Рис. 1 Початкові дані

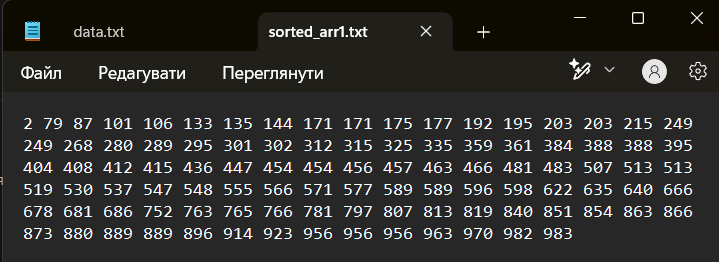


Рис. 2 Посортовані масиви послідовним алгоритмом

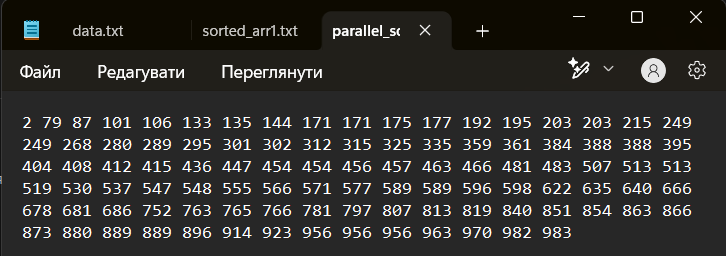


Рис. 3 Посортовані масиви з використання багатопотоковості

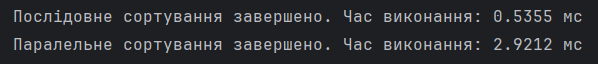


Рис. 4 Часові характеристики



Рис. Часові характеристики сортування з більшою кількістю елементів

Висновок

У ході виконання лабораторної роботи було застосовано послідовні алгоритми для розв'язання індивідуального завдання, а саме: впорядкування числового масиву за методом вибору.

Також було розв'язано те ж саме завдання з використанням паралельних обчислень та повторно визначено часові характеристики.

Результати роботи показали, що послідовне сортування було виконано за 0.5355 мс, тоді як паралельне сортування зайняло 2.9212 мс. Це свідчить про те, що в даному випадку послідовний алгоритм виявився більш ефективним за часом виконання. Це пов'язано з особливостями реалізації паралельного алгоритму, накладними витратами на створення та синхронізацію потоків, або ж з недостатнім розміром вхідних даних для ефективного паралелізму.