МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра системного проектування

**ЛАБОРАТОРНА РОБОТА №3**

з дисципліни: «ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ»

Виконав:

студент 3 курсу

групи ДА-81

Головін Б. О.

Варіант 5

Київ – 2021

***Лабораторна робота 3***

**Неблокуючі алгоритми.**

***Мета роботи:***

Розробка і реалізація паралельного алгоритму для задач із паралелізмом даних.

***Завдання*** Варіант 5

Розробити програму, яка за допомогою AtomicInteger і метода compareAndSet виконує наступні операції для одновимірного массиву (для потоків використовувати ExecutorService або parallelStream).

Можливе виконання будь-якою мовою програмування з використанням відповідних конструкцій неблокуючих алгоритмів.

Розмір масиву брати за формулою 10000\*N, де N – номер вашого варіанту.

Створити паралельні функції для знаходження:

- контрольної суми із використанням XOR для елементів масиву типу int.

- мінімального та максимального елементів масиву типу long а також їх індекси;

- моди і медіани елементів масиву типу long а також їх індекси;

- скалярного добутку двох векторів;

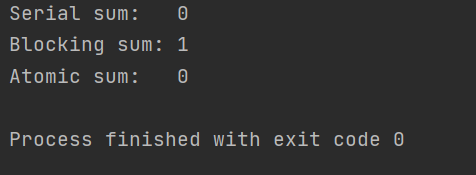
- вашого власного завдання.

**Лістинг програми та її результати роботи**

*- контрольної суми із використанням XOR для елементів масиву типу int.*

***Main.java***

import java.util.Arrays;  
import java.util.concurrent.atomic.AtomicInteger;  
import java.util.stream.IntStream;  
public class Main {  
 static int *sum* = 0;  
  
 public static void main(String[] args) {  
 int array[] = new int [50000]; /\* long array[] = new Random().longs(SIZE).toArray();\*/  
  
 Arrays.*fill*(array,1);  
 int serialSum = IntStream.*of*(array).reduce(0, (a, b) -> a ^ b);  
  
 AtomicInteger atomicSum = new AtomicInteger(); /\*створення атомарної змінної (atomic variable)\*/  
 IntStream.*of*(array).parallel().forEach( arrayElement -> { /\*вбудований метод для виконання паралельних обчислень у Java\*/  
 *sum* ^= arrayElement; /\*знаходження суми без використання атомарної змінної\*/  
 int oldValue;  
 int newValue;  
 do{ //зміна суми з використанням методу compareAndSet  
 oldValue = atomicSum.get();  
 newValue = oldValue ^ arrayElement;  
 }while(!atomicSum.compareAndSet(oldValue , newValue));  
 });  
  
 System.*out*.println("Serial sum: " + serialSum);  
 System.*out*.println("Blocking sum: " + *sum*);  
 System.*out*.println("Atomic sum: " + atomicSum.get());  
 }  
}

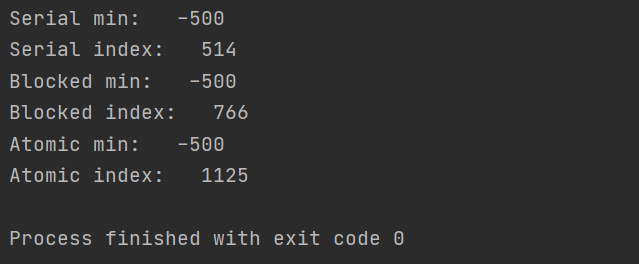
****

- мінімального та максимального елементів масиву типу long а також їх індекси;

***Варіант 1***

***Main.java***

import java.util.ArrayList;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.concurrent.atomic.AtomicReference;  
  
public class Main {  
 static int *SIZE* = 50000;  
  
 static long *minSerial* = 0;  
 static int *minSerialIndex* = 0;  
  
 static long *minBlocked* = 0;  
 static int *minBlockedIndex* = 0;  
  
 public static ArrayList<Long> getRandomArray(ArrayList<Long> array) {  
 long randMin = -501L, randMax = 500L;  
 for (int i = 0; i < *SIZE*; i++) {  
 array.add((long) (randMin + (Math.*random*() \* randMax)));  
 }  
 // System.out.println(array.toString());  
 return array;  
 }  
  
 public static void main(String[] args) {  
 ArrayList<Long> array = new ArrayList<>();  
 *getRandomArray*(array);  
  
 *minSerial* = array.get(0);  
 array.forEach(element -> {  
 if (element < *minSerial*)  
 *minSerial* = element;  
 *minSerialIndex* = array.indexOf(element);  
 });  
  
 AtomicLong atomicMin = new AtomicLong();  
 AtomicLong atomicMinIndex = new AtomicLong();  
  
 *minBlocked* = array.get(0);  
 atomicMin.compareAndSet(atomicMin.get(), array.get(0));  
 atomicMinIndex.compareAndSet(atomicMin.get(), 0);  
  
 array.parallelStream().forEach(element -> {  
 // Blocked  
 if (element < *minBlocked*)  
 *minBlocked* = element;  
 *minBlockedIndex* = array.indexOf(element);  
 // Atomic  
 long oldValue, newValue, oldIndex, newIndex;  
 long index = array.indexOf(element);  
 do {  
 oldValue = atomicMin.get();  
 oldIndex = atomicMinIndex.get();  
 if (element >= oldValue)  
 break;  
 newValue = element;  
 newIndex = index;  
 } while (!(atomicMin.compareAndSet(oldValue, newValue) && atomicMinIndex.compareAndSet(oldIndex, newIndex)));  
 });  
  
  
 System.*out*.println("Serial min: " + *minSerial*);  
 System.*out*.println("Serial index: " + *minSerialIndex*);  
 System.*out*.println("Blocked min: " + *minBlocked*);  
 System.*out*.println("Blocked index: " + *minBlockedIndex*);  
 System.*out*.println("Atomic min: " + atomicMin.get());  
 System.*out*.println("Atomic index: " + atomicMinIndex.get());  
 }  
}

******

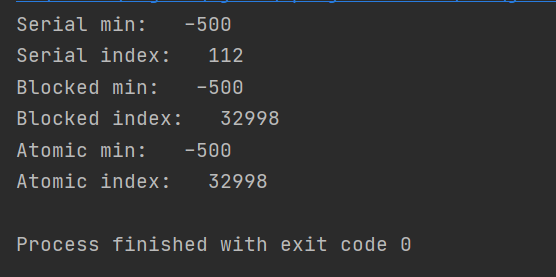
***Варіант 2***

***Main.java***

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.concurrent.atomic.AtomicReference;  
import java.util.stream.Stream;  
  
public class Main {  
 static int *SIZE* = 50000;  
  
 static Item *minSerial* = new Item(0,0);  
  
 static Item *minBlocked* = new Item(0,0);  
  
  
 public static Item[] getRandomArray(Item[] array) {  
 long randMin = -501L, randMax = 500L;  
 for (int i = 0; i < *SIZE*; i++) {  
 array[i] = new Item((long) (randMin + (Math.*random*() \* randMax)), i);  
 }  
 //array[200].data =-501;  
 return array;  
 }  
  
 public static void main(String[] args) {  
 Item[] array = new Item[*SIZE*];  
 *getRandomArray*(array);  
  
 *minSerial* = array[0];  
 Arrays.*stream*(array).forEach(element -> {  
 if (element.data < *minSerial*.data)  
 *minSerial* = element;  
 });  
  
 AtomicReference<Item> atomicMin = new AtomicReference<>();  
 atomicMin.compareAndSet(atomicMin.get(),array[0]);  
  
 Stream.*of*(array).parallel().forEach(element -> {  
 if (element.data < *minBlocked*.data)  
 *minBlocked* = element;  
 // Atomic  
 Item oldValue, newValue;  
 do {  
 oldValue = atomicMin.get();  
 if (element.data >= oldValue.data)  
 break;  
 newValue = element;  
 } while (!atomicMin.compareAndSet(oldValue, newValue));  
 });  
  
 System.*out*.println("Serial min: " + *minSerial*.data);  
 System.*out*.println("Serial index: " + *minSerial*.index);  
 System.*out*.println("Blocked min: " + *minBlocked*.data);  
 System.*out*.println("Blocked index: " + *minBlocked*.index);  
 System.*out*.println("Atomic min: " + atomicMin.get().data);  
 System.*out*.println("Atomic index: " + atomicMin.get().index);  
 }  
}

***Item.java***

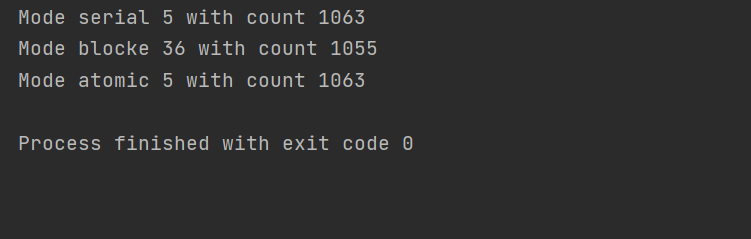
public class Item {  
 long data;  
 long index;  
  
 public Item (long data, long index) {  
 this.data= data;  
 this.index = index;  
 }  
}

******

- моди і медіани елементів масиву типу long а також їх індекси;

***Main.java***

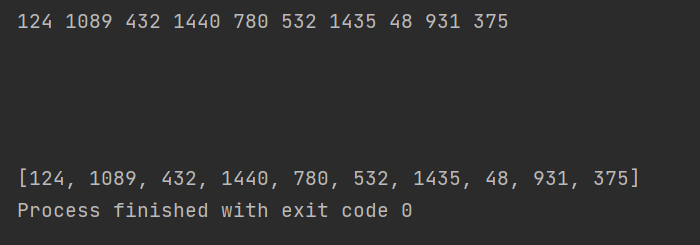
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.ConcurrentHashMap;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.concurrent.atomic.AtomicLongArray;  
import java.util.concurrent.atomic.AtomicReference;  
import java.util.stream.LongStream;  
import java.util.stream.Stream;  
  
public class Main {  
 static int *SIZE* = 50000;  
  
 public static long[] getRandomArray(long[] array) {  
 long randMin = 0L, randMax = 50L;  
 for (int i = 0; i < *SIZE*; i++) {  
 array[i] = ((long) (randMin + (Math.*random*() \* randMax)));  
 }  
 // System.out.println(array.toString());  
 return array;  
 }  
  
 public static void main(String[] args) {  
 long array[] = new long[*SIZE*];  
 *getRandomArray*(array);  
  
 long max = LongStream.*of*(array).max().getAsLong();  
  
 long[] dictionarySerial = new long[(int) (max+1)];  
 long[] dictionaryBlocked = new long[(int) (max+1)];  
 AtomicLongArray dictionary = new AtomicLongArray((int) max + 1);  
  
 LongStream.*of*(array).forEach(element -> {  
 dictionarySerial[(int) element]++;  
 });  
  
 LongStream.*of*(array).parallel().forEach(element -> {  
 dictionaryBlocked[(int) element]++;  
 long oldValue;  
 long newValue;  
 do {  
 oldValue = dictionary.get((int) element);  
 newValue = oldValue + 1;  
 } while (!dictionary.compareAndSet((int) element, oldValue, newValue));  
 });  
  
 long mode = 0, modeSerial = 0, modeBlocked = 0;  
 long modeIndex = 0, modeSerialIndex = 0, modeBlockedIndex = 0;  
 for (int i = 0; i < dictionary.length(); i++) {  
 if (mode < dictionary.get(i)) {  
 mode = dictionary.get(i);  
 modeIndex = i;  
 }  
 if (modeSerial < dictionarySerial[i]) {  
 modeSerial = dictionarySerial[i];  
 modeSerialIndex = i;  
 }  
 if (modeBlocked < dictionaryBlocked[i]) {  
 modeBlocked = dictionaryBlocked[i];  
 modeBlockedIndex = i;  
 }  
 }  
  
 System.*out*.println("Mode serial "+ modeSerialIndex + " with count " + modeSerial);  
 System.*out*.println("Mode blocke "+ modeBlockedIndex + " with count " + modeBlocked);  
 System.*out*.println("Mode atomic "+ modeIndex + " with count " + mode);  
 }  
}



- скалярного добутку двох векторів;

***Main.java***

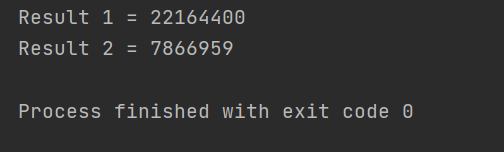
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.ConcurrentHashMap;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.concurrent.atomic.AtomicLongArray;  
import java.util.concurrent.atomic.AtomicReference;  
import java.util.stream.LongStream;  
import java.util.stream.Stream;  
  
public class Main {  
 static int *SIZE* = 10;  
  
 public static long[] getRandomArray(long[] array) {  
 long randMin = 0L, randMax = 50L;  
 for (int i = 0; i < *SIZE*; i++) {  
 array[i] = ((long) (randMin + (Math.*random*() \* randMax)));  
 }  
 // System.out.println(array.toString());  
 return array;  
 }  
  
 public static long[] getNumberArray(long[] array) {  
 for (int i = 0; i < *SIZE*; i++) {  
 array[i] = i;  
 }  
 // System.out.println(array.toString());  
 return array;  
 }  
  
  
 public static void main(String[] args) {  
 long array1[] = new long[*SIZE*];  
 long array2[] = new long[*SIZE*];  
 long arrayAdd[] = new long[*SIZE*];  
  
 *getRandomArray*(array1);  
 *getRandomArray*(array2);  
 *getNumberArray*(arrayAdd);  
  
 long resultSerial[] = new long[*SIZE*];  
  
 LongStream.*of*(arrayAdd).forEachOrdered(i -> {  
 resultSerial[(int) i] = array1[(int) i] \* array2[(int) i];  
 });  
  
 AtomicLongArray result = new AtomicLongArray(*SIZE*);  
  
 LongStream.*of*(arrayAdd).parallel().forEachOrdered(i -> {  
 long oldValue;  
 long newValue;  
 do {  
 oldValue = result.get((int) i);  
 newValue = array1[(int) i] \* array2[(int) i];  
 } while (!result.compareAndSet((int) i, oldValue, newValue));  
 });  
  
 LongStream.*of*(resultSerial).forEach(i -> System.*out*.print(i + " "));  
 System.*out*.println("\n\n\n\n");  
 Stream.*of*(result).forEach(i -> System.*out*.print(i + " "));  
 }  
}



- вашого власного завдання. Перемноження двох векторів, та якщо добуток парний то записати у першу суму, ні – другу.

***Main.java***

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.ConcurrentHashMap;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.concurrent.atomic.AtomicLongArray;  
import java.util.concurrent.atomic.AtomicReference;  
import java.util.stream.LongStream;  
import java.util.stream.Stream;  
  
public class Main {  
 static int *SIZE* = 50000;  
  
 public static long[] getRandomArray(long[] array) {  
 long randMin = 0L, randMax = 50L;  
 for (int i = 0; i < *SIZE*; i++) {  
 array[i] = ((long) (randMin + (Math.*random*() \* randMax)));  
 }  
 // System.out.println(array.toString());  
 return array;  
 }  
  
 public static long[] getNumberArray(long[] array) {  
 for (int i = 0; i < *SIZE*; i++) {  
 array[i] = i;  
 }  
 // System.out.println(array.toString());  
 return array;  
 }  
  
  
 public static void main(String[] args) {  
 long array1[] = new long[*SIZE*];  
 long array2[] = new long[*SIZE*];  
 long arrayAdd[] = new long[*SIZE*];  
  
 *getRandomArray*(array1);  
 *getRandomArray*(array2);  
 *getNumberArray*(arrayAdd);  
  
 AtomicLong result1 = new AtomicLong();  
 AtomicLong result2 = new AtomicLong();  
  
 LongStream.*of*(arrayAdd).parallel().forEachOrdered(i -> {  
 long oldValue;  
 long newValue;  
 long Value;  
 while (true){  
 Value = array1[(int) i] \* array2[(int) i];  
 if (Value % 2 == 0){  
 oldValue = result1.get();  
 newValue = oldValue + Value;  
 if (result1.compareAndSet(oldValue, newValue)) break;  
 } else {  
 oldValue = result2.get();  
 newValue = oldValue + Value;  
 if (result2.compareAndSet(oldValue, newValue)) break;  
 }  
 }  
 });  
  
 System.*out*.println("Result 1 = " + result1.get());  
 System.*out*.println("Result 2 = " + result2.get());  
 }  
}



**Висновки**

У даній лабораторній роботі ми ознайомитись з поняттями використання неблокуючих алгоритмів у Java. А також проводилась розробка і реалізація програм яка моделюють такі задачі:

- контрольної суми із використанням XOR для елементів масиву типу int.

- мінімального та максимального елементів масиву типу long а також їх індекси;

- моди і медіани елементів масиву типу long а також їх індекси;

- скалярного добутку двох векторів;

- вашого власного завдання. Перемноження двох векторів, та якщо добуток парний то записати у першу суму, ні – другу.