Національний Технічний Університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №3

Роботу виконала:

Желєзнова В.С.

Група: ДА­-81

Перевірив: Яременко В. С.

Київ – 2021

ЗМІСТ

[Мета роботи 3](#_Toc73568726)

[Завдання 3](#_Toc73568727)

[1. Лістинг програми мовою Java 4](#_Toc73568728)

[3. Приклад роботи програми 9](#_Toc73568729)

[Висновки 9](#_Toc73568730)

Мета роботи

Розробка і реалізація паралельного алгоритму для задач із паралелізмом даних.

Завдання

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 (пошук середнього числа)

1. Лістинг програми мовою Java
2. import java.util.Random;  
   import java.util.concurrent.atomic.AtomicInteger;  
   import java.util.stream.*IntStream*;  
     
   public class xor {  
     
    public xor(int N){  
    int[] array = new Random().ints(10000L \* N).toArray();  
    int serialXOR = serialXORFunc(array);  
    int atomicXOR = atomicXORFunc(array);  
     
    System.*out*.println("Serial XOR: " + serialXOR);  
    System.*out*.println("Atomic XOR: " + atomicXOR + "\n");  
    }  
     
    private int serialXORFunc (int[] array){  
    return *IntStream*.*of*(array).reduce((a, b) -> a ^ b).orElseThrow();  
    }  
     
    private int atomicXORFunc (int[] array){  
    AtomicInteger atomic = new AtomicInteger();  
    *IntStream*.*of*(array).parallel().forEach(arrayElement -> {  
    int oldXOR;  
    int newXOR;  
    do {  
    oldXOR = atomic.get();  
    newXOR = oldXOR ^ arrayElement;  
    } while (!atomic.compareAndSet(oldXOR, newXOR));  
    });  
    return atomic.get();  
    }  
   }

Лістинг 1. Xor.java

import java.util.concurrent.atomic.AtomicLong;  
import java.util.stream.*LongStream*;  
import java.util.Random;  
  
public class minMax {  
  
 public minMax(int N){  
 long[] array = new Random().longs(10000L\*N).toArray();  
 long serialMin = serialMinFunc(array);  
 long serialMax = serialMaxFunc(array);  
 long atomMin = atomicMinFunc(array);  
 long atomMax = atomicMaxFunc(array);  
  
 System.*out*.println("Serial min: " + serialMin);  
 System.*out*.println("Atomic min: " + atomMin);  
  
 System.*out*.println("Serial max: " + serialMax);  
 System.*out*.println("Atomic max: " + atomMax + "\n");  
 }  
  
 private long serialMinFunc(long[] array){  
 return *LongStream*.*of*(array).min().orElseThrow();  
 }  
  
 private long serialMaxFunc(long[] array){  
 return *LongStream*.*of*(array).max().orElseThrow();  
 }  
  
 private long atomicMinFunc(long[] array){  
 AtomicLong min = new AtomicLong(Long.*MAX\_VALUE*);  
 *LongStream*.*of*(array).parallel().forEach( arrayElement -> {  
 long oldMin;  
 long newMin;  
 do{  
 oldMin = min.get();  
 if (arrayElement < oldMin){  
 newMin = arrayElement;  
 }  
 else {  
 newMin = oldMin;  
 }  
 }while(!min.compareAndSet(oldMin, newMin));  
 });  
 return min.get();  
 }  
  
 private long atomicMaxFunc(long[] array){  
 AtomicLong max = new AtomicLong(Long.*MIN\_VALUE*);  
 *LongStream*.*of*(array).parallel().forEach( arrayElement -> {  
 long oldMax;  
 long newMax;  
 do{  
 oldMax = max.get();  
 if (arrayElement > oldMax){  
 newMax = arrayElement;  
 }  
 else {  
 newMax = oldMax;  
 }  
 }while(!max.compareAndSet(oldMax, newMax));  
 });  
 return max.get();  
 }  
}

Лістинг 2. MinMax.java

import java.util.\*;  
import java.util.*List*;  
import java.util.concurrent.atomic.AtomicIntegerArray;  
import java.util.stream.Collectors;  
import java.util.stream.*LongStream*;  
  
public class moda {  
  
 public moda(int N){  
 long[] array = new Random().longs(10000L \* N, 0, 1000).toArray();  
 int[] freqArr = new int[1000];  
 int serialModa = serialModaFunc(array);  
 int atomicModa = atomicModaFunc(array, freqArr);  
 long median = medianFunc(array, freqArr);  
  
 System.*out*.println("Median: " + median);  
 System.*out*.println("Serial moda: " + serialModa);  
 System.*out*.println("Atomic moda: " + atomicModa + "\n");  
 }  
  
 private int serialModaFunc(long[] array){  
 int[] freq = new int[1000];  
 *LongStream*.*of*(array).forEach(arrayElement -> {  
 freq[(int) arrayElement]++;  
 });  
 return Arrays.*stream*(freq).boxed().collect(Collectors.*toList*()).indexOf(Arrays.*stream*(freq).max().orElseThrow());  
 }  
  
 private int atomicModaFunc(long[] array, int[] freqArr){  
 AtomicIntegerArray atomicFreqArr = new AtomicIntegerArray(1000);  
 *LongStream*.*of*(array).parallel().forEach( arrayElement -> {  
 int oldValue;  
 int newValue;  
 do{  
 oldValue = atomicFreqArr.get((int) arrayElement);  
 newValue = oldValue + 1;  
 }while(!atomicFreqArr.compareAndSet((int) arrayElement, oldValue, newValue));  
 });  
  
 for(int i=0; i < 1000; i++){  
 freqArr[i] = atomicFreqArr.get(i);  
 }  
 return Arrays.*stream*(freqArr).boxed().collect(Collectors.*toList*()).indexOf(Arrays.*stream*(freqArr).max().orElseThrow());  
 }  
  
 private long medianFunc(long[] array, int[] freqArr){  
 HashMap<Integer, Integer> hashMap = new HashMap<>();  
 for (int i=0; i < 1000; i++){  
 hashMap.put(i, freqArr[i]);  
 }  
  
 *List*<*Map*.*Entry*<Integer, Integer>> sortedList = new ArrayList<>(hashMap.entrySet());  
 sortedList.sort(*Comparator*.*comparingInt*(*Map*.*Entry*::getKey));  
 int medianIndex = array.length / 2;  
 int index = 0;  
 long median = 0;  
 for (*Map*.*Entry*<Integer, Integer> integerIntegerEntry : sortedList) {  
 index += integerIntegerEntry.getValue();  
 if (index >= medianIndex) {  
 median = integerIntegerEntry.getKey();  
 break;  
 }  
 }  
 return median;  
 }  
}

Лістинг 3. Moda.java

import java.util.concurrent.atomic.AtomicLong;  
import java.util.stream.*IntStream*;  
import java.util.Random;  
  
public class scalarProduct {  
  
 public scalarProduct(int N){  
 int[] array1 = new Random().ints(10000L\*N, 0, 100).toArray();  
 int[] array2 = new Random().ints(10000L\*N, 0, 100).toArray();  
 long serialScalar = serialProduct(array1, array2);  
 long atomicScalar = atomicProduct(array1, array2);  
  
 System.*out*.println("Serial scalar: " + serialScalar);  
 System.*out*.println("Atomic scalar: " + atomicScalar + "\n");  
 }  
  
 private long serialProduct(int[] array1, int[] array2){  
 return *IntStream*.*range*(0, array1.length).mapToLong(index -> (long) array1[index] \* array2[index]).sum();  
 }  
  
 private long atomicProduct(int[] array1, int[] array2){  
 AtomicLong scalar = new AtomicLong();  
 *IntStream*.*range*(0, array1.length).parallel().forEach(index -> {  
 long oldScalar;  
 long newScalar;  
 do {  
 oldScalar = scalar.get();  
 newScalar = oldScalar + (long) array1[index] \* array2[index];  
 } while (!scalar.compareAndSet(oldScalar, newScalar));  
 });  
 return scalar.get();  
 }  
}

Лістинг 4. scalarProduct.java

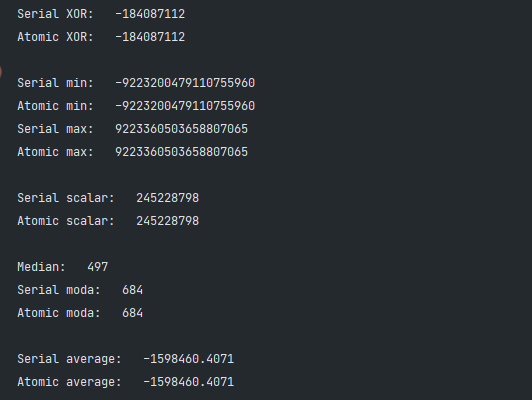
import java.util.Random;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.stream.*IntStream*;  
  
public class average {  
  
 public average(int N){  
 int[] array = new Random().ints(10000L \* N).toArray();  
 double serialAverage = serialAvgFunc(array);  
 double atomicAverage = atomicAvgFunc(array);  
  
 System.*out*.println("Serial average: " + serialAverage);  
 System.*out*.println("Atomic average: " + atomicAverage + "\n");  
 }  
  
 private double serialAvgFunc(int[] array){  
 return *IntStream*.*of*(array).average().orElseThrow();  
 }  
  
 private double atomicAvgFunc(int[] array){  
 AtomicLong sum = new AtomicLong();  
 *IntStream*.*of*(array).parallel().forEach(arrayElement -> {  
 long oldSum;  
 long newSum;  
 do {  
 oldSum = sum.get();  
 newSum = oldSum + arrayElement;  
 } while (!sum.compareAndSet(oldSum, newSum));  
 });  
 return (double) sum.get() / array.length;  
 }  
}

Лістинг 5. Average.java

public class Main {  
  
 public static final int *N* = 10;  
 public static void main(String[] args) {  
 new xor(*N*);  
 new minMax(*N*);  
 new scalarProduct(*N*);  
 new moda(*N*);  
 new average(*N*);  
 }  
}

Лістинг 6. Main.java

1. Приклад роботи програми



Висновки

У ході виконання даної лабораторної роботи було розроблено програму, що реалізує умову завдання.

Найважливіше, що отримані результати для послідовного та паралельного розв’язків співпадають.