Національний Технічиний Університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №1

Роботу виконав:

Дєрюгін Є. О.

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 9](#_Toc64843218)

[Висновки 10](#_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 7.

Створити вектор з N>=10000 елементами з випадкових чисел. Знайти норму вектора.

1. Лістинг програми мовою Java

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

import java.util.Arrays;  
  
public class ThreadSample {  
 public static int *SIZE* = 6000;  
 public static int *NUNMBER\_THREADS* = 8;  
 public static int *TABLE\_HEADER* = 2;  
 public static int *TABLE\_FOOTER* = 2;  
 public static int *TESTS* = 10;  
 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[]{"Norm","Time(us)","Norm","Time(us)"};  
 long[] parallelTimes = new long[*TESTS*];  
 long[] serialTimes = new long[*TESTS*];  
  
 for(int test\_i=0;test\_i<*TESTS*;test\_i++) {  
  
  
 double vectA[] = new double[*SIZE*];  
 int rand\_min = 1;  
 int rand\_max = 42;  
 for (int i = 0; i < *SIZE*; i++) { *//початкове заповнення векторів випадковими величинами з зазначеного проміжку* vectA[i] = rand\_min + (int) (Math.*random*() \* rand\_max);  
 }  
  
 long start = System.*nanoTime*();  
  
 double serialResult = 0;  
 for (int i = 0; i < *SIZE*; i++) {  
 serialResult += Math.*abs*(vectA[i]);  
 }  
  
 long end = System.*nanoTime*();  
 long serialTime = (end - start) / 1000;  
 serialTimes[test\_i] = serialTime;  
  
 ThreadCacl TreadArray[] = new ThreadCacl[*NUNMBER\_THREADS*];  
  
 start = System.*nanoTime*();  
  
  
 for (int i = 0; i < *NUNMBER\_THREADS*; i++) { *//розбиття на потоки* TreadArray[i] = new ThreadCacl(vectA,  
 *SIZE* / *NUNMBER\_THREADS* \* i,  
 i == (*NUNMBER\_THREADS* - 1) ? *SIZE* : *SIZE* / *NUNMBER\_THREADS* \* (i + 1)); *//тернарна умовна операція* TreadArray[i].start();  
 }  
 for (int i = 0; i < *NUNMBER\_THREADS*; i++) { *//очікування завершення усіх потоків* TreadArray[i].join();  
 }  
 double parallelResult = 0;  
 for (int i = 0; i < *NUNMBER\_THREADS*; i++) { *//збір результатів паралельної роботи* parallelResult += TreadArray[i].getResult();  
 }  
  
 end = System.*nanoTime*();  
  
 long parallelTime = (end - start) / 1000;  
 parallelTimes[test\_i] = parallelTime;  
  
 resultsTable[*TABLE\_HEADER* + test\_i] = new String[]{String.*valueOf*(serialResult),String.*valueOf*(serialTime),String.*valueOf*(parallelResult),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("Size =" + String.*valueOf*(*SIZE*));  
 System.*out*.println("Number of threads = " + String.*valueOf*(*NUNMBER\_THREADS*));  
 System.*out*.println("");  
 for(final Object[] row : resultsTable){  
 System.*out*.format("%10s%10s%10s%10s%n",row);  
 }  
 }  
}

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

Лістинг 1. ThreadSample.java

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

class ThreadCacl extends Thread {  
  
 double vectA[];  
 int startIndex;  
 int endIndex;  
 double result;  
  
 public ThreadCacl(double[] vectA, int startIndex, int endIndex) { *//конструктор класу, приймає дані для обчислень* this.vectA = vectA;  
 this.startIndex = startIndex;  
 this.endIndex = endIndex;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 @Override  
 public void run() { *//обрахунки, що здійснюватимуться в зазначеному потоці* for (int i = startIndex; i < endIndex; i++) {  
 result += Math.*abs*(vectA[i]);  
 }  
 }  
}

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

Лістинг 2. ThreadCalc.java

1. Порівняння реалізацій

При значеннях:

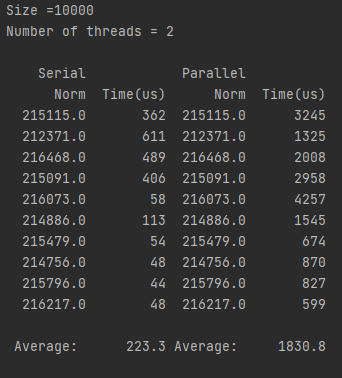
SIZE = 10000, NUMBER\_THREADS = 2

Рисунок 2.1 Таблиця з порівнянням для вектору 10000 та 2-ма потоками

При значеннях:

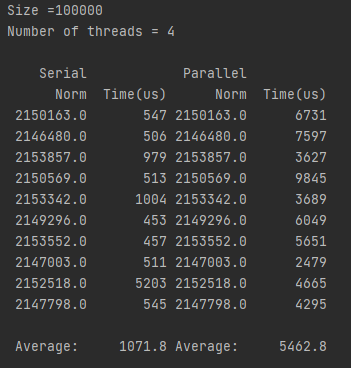
SIZE = 100000, NUMBER\_THREADS = 4

Рисунок 2.2 Таблиця з порівнянням для вектору 10000 та 4-ма потоками

1. Порівняння часу виконання

Таблиця 3.1 Порівняння часу виконання алгоритму

|  |  |  |  |
| --- | --- | --- | --- |
| SIZE | Serial | 2 Threads | 4 Threads |
| 10000 | 264.4 | 2070.0 | 1789.7 |
| 15000 | 233.2 | 7506.4 | 3672.8 |
| 20000 | 473.6 | 3118.1 | 3087.2 |
| 50000 | 1011.6 | 4777.2 | 4593.6 |
| 100000 | 1852.0 | 3814.0 | 3301.5 |

Рисунок 3.1 Графічне зображення порівняння часу виконання алгоритму

Висновки

У ході виконання даної лабораторної роботи я розробив програму, що реалізує варіант мого завдання як послідовно, так і паралельно, а саме розрахунок норми вектора розмірністю N>=10000. 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.