LAB 8

Jarosław Kołodziej

Kod źródłowy:
Main:

```
public static void main(String[] args) throws Exception
{
   if(args.length != 7)
   {
       throw new IllegalArgumentException("prog <VECTOR_SIZE> <THREAD_FRAME> <NUMBER_OF_THREADS>
<IMAGE_N_SIZE> <IMAGE_M_SIZE> <TEMP>");
   }

   final int vecSize = Integer.parseInt(args[0]);
   final int threadFrame = Integer.parseInt(args[1]);
   final int numOfThreads_zad2 = Integer.parseInt(args[2]);
   final int n = Integer.parseInt(args[3]);
   final int m = Integer.parseInt(args[4]);
   final int sizeOfSourceArray = Integer.parseInt(args[5]);
   final int numbersPerThread = Integer.parseInt(args[6]);

   System.out.printf("Zad1 ---------------------------- vector size: %d, operations per one thread: %d\n",
vecSize,threadFrame);
   Zad1.zad1(vecSize, threadFrame);
   System.out.printf("Zad2 ---------------------------- number of threads: %d, n = %d, m = %d\n",
numOfThreads_zad2,n,m);
   Zad2.zad2(numOfThreads_zad2,n,m);
   System.out.printf("Zad3 ---------------------------- size of Source int array: %d, numbers per thread:
%d\n", sizeOfSourceArray,numbersPerThread);
   Zad3.zad3(sizeOfSourceArray, numbersPerThread);
}
```

Output z uruchomionego programu:

```
Zad1 ----------------------------
vector size: 20, operations per one thread: 3
       last element of vec1: 33
       last element of vec2: 36
       last element of vec result: 69
Zad2 ----------------------------
number of threads: 5, n = 50, m = 30
       Watek 3: { 2x
       Watek 3: ] 4x
       Watek 3: } 0x
       Watek 3: \ 2x
       Watek 3: | 1x
       Watek 3: ; 6x
       Watek 0: ~ 1x
       Watek 0: ` 2x
       Watek 0: ! 4x
       Watek 0: @ 2x
       Watek 0: # 1x
       Watek 2: ) 6x
       Watek 2: _ 2x
```

```
       Watek 2: - 2x
       Watek 2: = 5x
       Watek 2: + 3x
       Watek 2: [ 4x
       Watek 4: ' 4x
       Watek 4: , 0x
       Watek 4: < 0x
       Watek 4: . 5x
       Watek 4: > 3x
       Watek 4: ? 3x
       Watek 1: $ 2x
       Watek 1: % 3x
       Watek 1: ^ 5x
       Watek 1: & 2x
       Watek 1: * 1x
       Watek 1: ( 2x
Zad3 ----------------------------
size of Source int array: 100, numbers per thread: 20
[1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]

       Process finished with exit code 0
```

Ogólne objaśnienie zrównoplegnienia w Javie:
Klasa “Thread” dziedzicząca z klasy “Runnable” jest odpowiedzialna za obsługę programu na innym wątku. Każdy stworzony obiekt dziedziczący z klasy “Thread” (po wywołaniu metody start) będzie wykonywać się na innym wątku niż proces z którego stworzyliśmy ten obiekt.
Podczas tworzenia Obiektu takiej klasy podajemy do konstruktora obiekty które chcemy udostępnić innemu wątkowi.

Metoda run() jest odpowiedzialna za “entry point” dla takiego wątku.

W wątku rodzica używamy metody join() na obiekcie pochodnym z Thread by zaczekać na jego zakończenie.

Do każdego z zadań użyłem tej tego samego rozwiązania jeśli chodzi o zapewnienie równoległego rozłożenia danych na wątkach w przypadku gdy ilość_elementów_tablicy % ilosc_watków nie byłaby równa zeru.

Wykorzystuje tutaj właśnie operator modulo który daje mi dostęp do reszty elementów które nie mieściły by się w danym “frame”-mie dla wątku.

```
int num_threads = vecSize / thFrame;
int restOfThreads = vecSize % thFrame;
```

Następnie podczas przekazywania danych do wątku :

```
if(restOfElements > 0)
{
   ++to;
   --restOfElements;
}
```

“to” - ostatni index elementu do przekazania wątkowi z początkowej tablicy.

I dzięki temu największa różnica pomiędzy ilością elementów dla każdego wątku to 1