Xiana Lara
Concurrency
November 13, 2020

Concurrency

Concurrency is becoming a very important programming concept due to the explosion of multi-threaded, multi-core processors. Hence, for this lab I had to create a square two-dimensional matrix and calculate some basic statistics while measuring the time.

Code:

```
/*
 * xiana lara
 * nov 13 2020
 * a java concurrent program with a square two
 * dimensional matrix and calculate some basic
 * statistics while measuring the time.
 * input: size of matrix N
 * output: Thread starting, exiting, value
 * Time to complete, min, max, and average
 *
 * From: http://www.letmeknows.com/2017/04/24/wait-for-threads-to-finish-java/
 *
 * Adopted By: Shaun Cooper
 * Last Updated Nov 2020
 *
 * We need static variable pointers in the main class so that we can share these
 * values with the threads. The threads are address separate from us, so we need
 * to share pointers to the objects that we are sharing and updating.
 */

import java.util.*;
import java.math.*;


public class concur{

    private static ArrayList<Thread> arrThreads = new ArrayList<Thread>();

    // we use static variables to help us connect the threads
    // to a common block
    public static int N = 0;
    public static int[][] A;

    // max, min, and avg arrays
    public static int[] maximum;
    public static int[] minimum;
    public static float[] average;
```

```
    // main entry point for the process
    public static void main (String[] args){
        try{

            // local min/max/avg variables
            int locMin = 0;
            int locMax = 0;
            float locAvg = 0;

            // user input
            Scanner scnr = new Scanner(System.in);
            System.out.println("Enter Array Size N: ");
            int size = scnr.nextInt();
            N = size;

            // create the array from input
            A = new int[size][size];
            minimum = new int[size];
            maximum = new int[size];
            average = new float[size];

            // max and minimum exponential
            int max = (int) (Math.pow(2, (32-N)));
            int min = (int) (Math.pow(2, (31-N)));

            // exponential range
            int range = max - min;

            // fill array with random values
            for(int i = 0; i < A.length; i++){

                for(int y = 0; y < A.length; y++){

                    A[i][y] = (int)(range * Math.random() + 1);

                } // of for

            } // of for

            // start timer
            long startTime = System.nanoTime();

            // create N threads
            for(int i = 0; i < size; i++){
```

```
            Thread T1 = new Thread(new ThreadTest(i));

            // thread start
            T1.start();
            arrThreads.add(T1);

        } // of for

        // Wait for each thread to complete
        for(int i = 0; i < arrThreads.size(); i++){

            arrThreads.get(i).join();

        } // of for

        // end time measuring
        long endTime = System.nanoTime();

        // locMin = minimum at index 1
        locMin = minimum[0];

        // For loop to check the array and find the min, max, and average.
        for(int i = 0; i < N; i++){

            // minimum
            if(minimum[i] < locMin){

                locMin = minimum[i];

            } // of if

            // maximum
            if(maximum[i] > locMax){

                locMax = maximum[i];

            } // of if

            // average
            locAvg = locAvg + average[i];

        } // of for

        // check min and max values
        System.out.println(Arrays.toString(minimum));
```

```
            // time for calculations
            System.out.println("Time to complete: " + (endTime - startTime) + "ns
");

            // min, max, avg
            System.out.println("max: " + locMax + "\tmin: " + locMin + "\taverage
: " + locAvg);
            System.out.println("Main Thread has N as value " + N);

            // this for loop will not stop execution of any thread,
            // only it will come out when all thread are executed
            System.out.println("Main thread exiting ");

        } // of try

        catch(Exception e){

            System.out.println(e.getMessage());

        } // of catch

    } // of main

} // of concur

class ThreadTest implements Runnable{

    private int i;

    // local values
    private int minim = 0;
    private int maxim = 0;
    private float avg = 0;

    ThreadTest(int ind){

        i = ind;

    } // of ThreadTest

    public void run(){

        try{
```

```
            minim = concur.A[i][0];

            System.out.println("Thread is started " + i + " Array is " + concur.A
[i][0]);


            for(int x = 0; x < concur.N; x++){

                // minimum
                if(concur.A[i][x] < minim){
                    minim = concur.A[i][x];
                }

                // maximum
                if(concur.A[i][x] > maxim){
                    maxim = concur.A[i][x];
                }

                // average
                avg = avg + (concur.A[i][x]/ (concur.N * concur.N));


            }

            // values in minimum, maximum, and average
            concur.minimum[i] = minim;
            concur.maximum[i] = maxim;
            concur.average[i] = avg;

            Thread.sleep(1000);
            System.out.println("Thread is exiting " + i);

        }  // of try

        catch(Exception e){

            System.out.println(e.getMessage());

        } // of catch

    } // of run

} // of ThreadTest
```

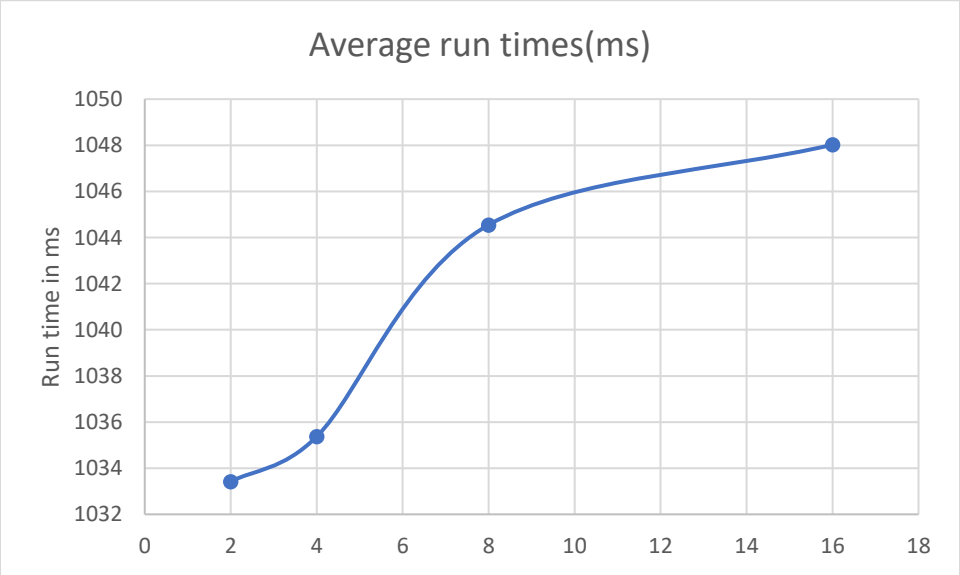
Xiana Lara
Concurrency
November 13, 2020

Output:

```
xlara@lovelace:~/Documents/pls/conc> java concur
Enter Array Size N:
2
Thread is started 0 Array is 312029934
Thread is started 1 Array is 489736332
Thread is exiting 0
Thread is exiting 1
[138190049, 173011362]
Time to complete: 1032201618ns
max: 489736332  min: 138190049  average: 2.7824192E8
Main Thread has N as value 2
Main thread exiting
```

Graphs:

	2	4	8	16
Run 1	1033662100	1037977497	1049442598	1042990591
Run 2	1034798444	1034657988	1044254585	1043749719
Run 3	1032793223	1033878533	1039829649	1051940693
Run 4	1032201618	1034795896	1045304855	1050183625
Run 5	1033654072	1035549869	1043909530	1051253724
standard deviation	986255.0622	1572609.664	3438258.69	4302332.652



From these results you can see nested for loops are not as fast as concurrency with 2D arrays. This is very clearly seen in how the results rise linearly instead of exponentially like nested for loops.