|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Matrix Size** | **Run 1** | **Run 2** | **Run 3** | **Run 4** | **Run 5** | **Average Run Time** | **Standard Deviation** |
| 2 | 45280163ns | 42907156ns | 44380051ns | 44252345ns | 43973432ns | 44158629.4ns | 763692.543 |
| 4 | 43659907ns | 46199623ns | 49088793ns | 48931209ns | 50071576ns | 47590221.6ns | 2348711.154 |
| 8 | 65640643ns | 60567908ns | 66148423ns | 65267696ns | 63382855ns | 64201505ns | 2043114.589 |
| 16 | 60707656ns | 63300549ns | 56744088ns | 60420706ns | 60265107ns | 60287621.2ns | 2089450.126 |

Given the data, we can see a generally linear increase in runtime as matrix size increases. However, we notice that the run time average actually decreases going from matrix size 8 to 16. I find this odd as there seemed to have been a linear increase in time. I would attribute this to the processor architecture of the lab computers. The processor might handle more threads more efficiently after a certain point. This experiment does show that having a separate thread handle its own row of the matrix is very fast and better than using big loops to fill the matrices and do the calculations.

/\*

 \* 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.

 \*/

/\*

\*Joseph Camacho-Terrazas

\*11/14/2020

\*Input: An integer argument that specifies the number of threads

\*Output: Thread info, minimum, maximum, and average values, and the run time

\*Precondition: User should enter valid integer input

\*Postcondition: Results of the thread test will be printed properly

\*/

import java.util.\*;

import java.math.\*;

public class concurrency {

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

    //We use static variable to help us connect the threads to a common block.

    public static int N = 0;

    public static int[][] A;

    //Arrays to store minimum, maximum, and average

    public static float[] Average;

    public static int[] Minimum;

    public static int[] Maximum;

    //Main entry point for the process

    public static void main (String[] args) {

        try {

            //Local tracking variables

            float mainAvg = 0;

            int mainMin = 0;

            int mainMax = 0;

            //Size comes from user argument

            int size = Integer.parseInt(args[0]);

            N = size;

            //Create the array from input as well as storage arrays

            A = new int[size][size];

            Average = new float[size];

            Minimum = new int[size];

            Maximum = new int[size];

            //Calculate the maximum exponentional range to help when filling the array

            int minRange = (int) (Math.pow(2, (31-N)));

            int maxRange = (int) (Math.pow(2, (32-N)));

            int range = maxRange - minRange;

            //Fill the array with random values

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

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

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

                }

            }

            //Take the start time in nanoseconds

            long startTime = System.nanoTime();

            //Create N threads to work on each row

            for (int x = 0; x < size; x++){

                Thread T1 = new Thread(new ThreadTest(x));

                T1.start(); //Standard thread start

                arrThreads.add(T1);

            }

            // Wait for each thread to complete

            for (int x = 0; x < arrThreads.size(); x++)

                {arrThreads.get(x).join();}

            //Take the finish time in nanoseconds

            long finishTime = System.nanoTime();

            //Set mainMin as the first index of Minimum

            mainMin = Minimum[0];

            //Retrieve minimum, maximum, and average values

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

                if (Minimum[x] < mainMin) {

                    mainMin = Minimum[x];

                }

                if (Maximum[x] > mainMax) {

                    mainMax = Maximum[x];

                }

                mainAvg = mainAvg + Average[x];

            }

            //Final results printouts

            System.out.println("Main Thread has N as value " + N);

            System.out.println("Time: " + (finishTime - startTime) + " nanoseconds");

            System.out.println("Maximum: " + mainMax + " Minimum: " + mainMin + " Average: " + mainAvg);

            //This for loop will not stop execution of any thread, only it will come out

            //when all threads are executed.

            System.out.println("Main thread exiting.");

        } catch(Exception e) {

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

        }

    }

} // End Main

class ThreadTest implements Runnable {

    private int i;

    //Local minimum, maximum, and average values for ThreadTest

    private float TTavg = 0;

    private int TTmin = 0;

    private int TTmax = 0;

    ThreadTest(int ind) {

        i = ind;

    }

    public void run() {

        try {

            TTmin = concurrency.A[i][0];

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

            //Loop finds minimum, maximum, and average values

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

                if (concurrency.A[i][x] < TTmin){

                    TTmin = concurrency.A[i][x];

                }

                if (concurrency.A[i][x] > TTmax){

                    TTmax = concurrency.A[i][x];

                }

                TTavg = TTavg + (concurrency.A[i][x]/ (concurrency.N \* concurrency.N));

            }

            //Store value in global Minimum, Maximum, and Average

            concurrency.Average[i] = TTavg;

            concurrency.Minimum[i] = TTmin;

            concurrency.Maximum[i] = TTmax;

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

        } catch(Exception e){

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

        }

    }

}















