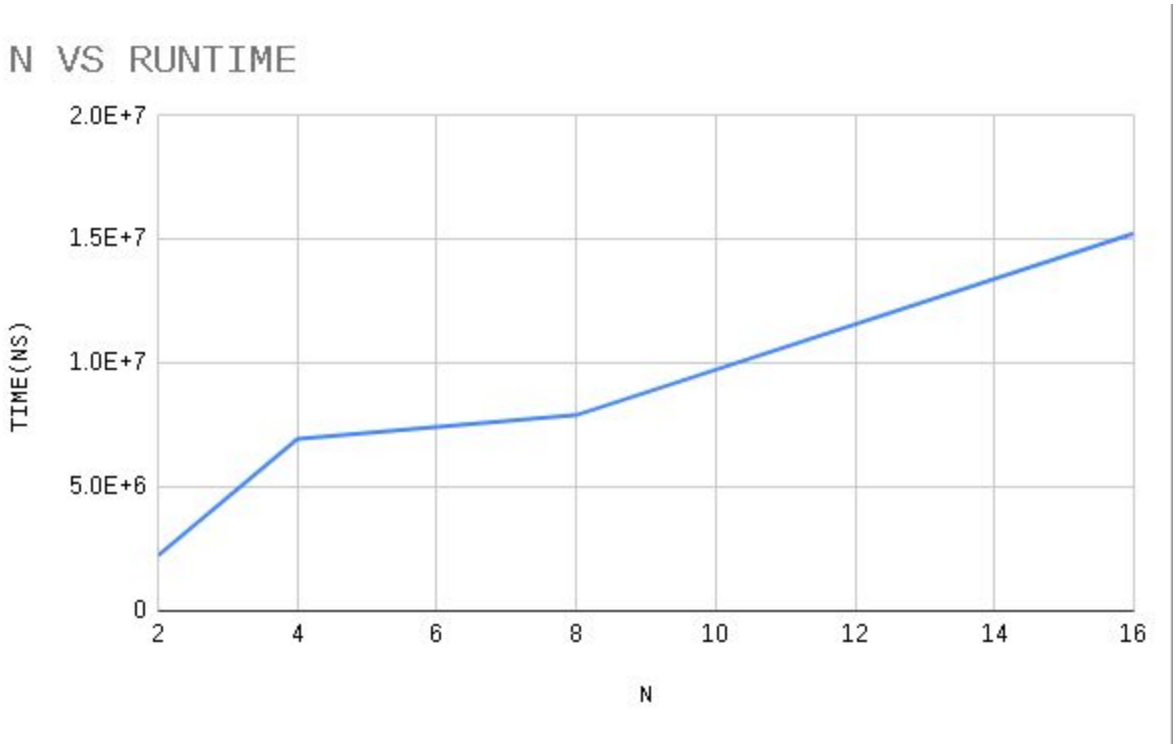
Collin Gros
11-13-2020
cs471
concurrency

CONCURRENCY: REPORT

GRAPHS AND TABLES



RAW				TRIAL#				
			2	3	4	5	avg	stddev
	2	1676676	2271752	3217300	1919587	2097810	2236625	590822.0845
N	4	6717601	10137682	3229606	11627020	3000761	6942534	3921969.639
	8	8196648	9742060	4086741	6135460	11372550	7906691.8	2880716.503
	16	14085444	16117891	9079551	21246792	15707144	15247364.4	4369048.138

CODE

Concurrency.java

```
/*
    collin gros
    11-11-2020
    cs471
    concurrency

    problem description: write a program in Java to create a square 2D matrix
                                            and calculate some basic statistics while
                                            measuring time.

    this program calculates the max, min, sum, and avg of random int values
    stored in a square matrix of a given size, and prints the time
    it takes to do so (in nanoseconds).

*/
```

```
import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.TimeUnit;


class Concurrency
{
    // we use static variables to help us connect the threads
    // to a common block
    private static ArrayList<Thread> arrThreads = new ArrayList<Thread>();

    // N is the size of the NxN matrix of random numbers, and the
    // number of threads created
    public static int N = 0;

    // A stores random numbers to analyze
    public static int[][] A;

    // stats stores statistics for each row analyzed; written to by threads
    // each entry is an array in the following format:
    //     max, min, sum
    public static float[][] stats;


    public static void main(String[] args)
    {
        try {
                // take 1 input from the user:
                //     (dimension of the square matrix [N])

                // create the array from input
                N = Integer.parseInt(args[0]);
                // create 2D NxN INTEGER matrix
                A = new int[N][N];
                // create stats matrix (N x 3)
                stats = new float[N][3];


                // randomly assign INTEGER values to each element
                for (int i = 0; i < N; ++i) {
                        for (int j = 0; j < N; ++j) {
                                // in the range of  between 2^(32-N) and 2^(31-N)
                                // trunc(  (2^(32-N) - (2^31-N))* Random()   + 2^(31-N))
                                double tmp = Math.pow(2, 32-N)
                                                                        - Math.pow(2, 31-N);
                                int x = (int) (Math.pow(2, 31-N)
                                                                + (Math.random() * tmp));
                                A[i][j] = x;
                        }
                }

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

        // create N threads (each is responsible for one row of the matrix)
        for (int i = 0; i < N; ++i) {
                Thread t = new Thread(new ThreadTest(i));
                t.start();
                arrThreads.add(t);
        }
```

```
        // each thread calculates max, min, and stats for summation and avg
        // and stores them in shared set of arrays (stats[][])
        // and main thread waits on all of them to finish
        for (int i = 0; i < N; ++i) {
               // WAITING... join to NULL when thread is done
               arrThreads.get(i).join();
        }


        // calculate the overall max, min, and average
        // (recall that) each entry is an integer array in
        // the following format:
        //     max, min, sum
        // initialize maximum to first max in matrix
        float max = Concurrency.A[0][0];
        // initialize minimum to first min in matrix
        float min = Concurrency.A[0][1];
        // initialize sum, avgs to 0
        float sum = 0, avgs = 0;
        for (int i = 0; i < N; ++i) {
               if (max < stats[i][0]) {
                      max = stats[i][0];
               }
               if (min > stats[i][1]) {
                      min = stats[i][1];
               }
               sum += stats[i][2];
               avgs += stats[i][3];
        }
        // calculate overall avg
        float avg = avgs / N;

        // save final results in array
        float[] totalstats = {max, min, sum, avg};

        // stop timer
        long stopTime = System.nanoTime();
        long duration = stopTime - startTime;


        // print final results
        System.out.println("total time(ns): " + duration);
        System.out.println("totalstats: max: "+totalstats[0]+" min: "
                                          +totalstats[1]+" sum: "+totalstats[2]
                                          +" avg: "+totalstats[3]);
        }
        catch (Exception e) {
               System.out.println(e.getMessage());
        }
    }
}
```

ThreadTest.java

```
/*
    collin gros
    11-11-2020
    cs471
    concurrency

    this class is used by the Concurrency class to assign threads a row
```

```
    and analyze it for statistics reported later in the Concurrency main()
    function.
*/

class ThreadTest implements Runnable
{
    int row;

    ThreadTest(int row)
    {
        // we are analyzing row #x
        this.row = row;
    }


    public void run()
    {
        // initialize maximum to first element in row
        float max = Concurrency.A[row][0];
        // initialize minimum to first element in row
        float min = Concurrency.A[row][0];
        // initialize sum to 0
        float sum = 0;

        // analyze row #row
        // loop thu row; Concurrency.N = length/width of row
        for (int i = 0; i < Concurrency.N; ++i) {
//              System.out.println("\t"+row+":\tA["+row+"]["+i+"]"
//                                                      + ": " + Concurrency.A[row][i]);
                float val = Concurrency.A[row][i];
                // get min, max, sum
                if (max < val) {
                        max = val;
                }
                if (min > val) {
                        min = val;
                }

                sum += val;
        }
        // calculate avg with sum and N
        float avg = sum / Concurrency.N;


        // store values in an integer array, which is then stored
        // in the shared array Concurrency.stats
        float [] stats = {max, min, sum, avg};
//      for (int i = 0; i < 4; ++i) {
//              System.out.println(row+":\t stats["+i+"]: "+stats[i]);
//      }

        try{
        Concurrency.stats[row] = stats;
        }
        catch (Exception e) {
                System.out.print(e);
                return;
        }
//      System.out.println(row+":\t max: "+max+" min: "+min+" sum: "+sum
//                                                      +" avg: "+avg);
    }
}
```

run.sh

```
#!/bin/bash
# collin gros
# 11-13-2020
# cs471
# concurrency
#
#
# script is used to compile and run Concurrency.java, and passes
# its first command line argment to Concurrency.java.
#
# mainly made this because it sucks having to manually enter the commands
# to compile and run. (poor pinkies)

javac Concurrency.java && java Concurrency $1
```

ANALYSIS

I broke up my code into two classes: Concurrency.java and TestThread.java. In Concurrency.java, I create the 2D matrix of the given size and insert random values into each index, using the method suggested from the problem description. I then start N threads with a for loop, and each thread examines its given row for its maximum, minimum, sum, and average. They 'return' these values by storing them in a shared static array, *stats*. Then, Concurrency.java gets the maximum, minimum, sums, and averages from all of those values (dividing the individual averages by the total number of threads to get an average). Lasly, Concurrency.java prints the time it took (from the timer that was started/stopped during all of this execution), as well as the values in *totalstats* (the maximum, minimum, sum, and average of ALL values inserted into the 2D matrix).

I ran into a couple of problems; I first encountered an overflow issue with using integer typed values in the thread and main() calculations, where they would overflow into a negative number. Then, after changing all of those values' types to float, everything worked properly. It confused me at first how the numbers calculated were decreasing as N increased, but I realized it was part of the random number generation, where 2^(32-N) is used (the number decreased because N was subtracted from the exponent). It surprised me how high the standard deviation was for all of my input, but I guess I can see that a difference in 590822 nanoseconds is a difference in 0.000590822 seconds, which is extremely small.

OUTPUT

```
collin@collin-workstation:~/Documents/school/cs-471/concurrency$ ./run.sh 2
total time(ns): 3808806
totalstats: max: 1.02115718E9 min: 5.7896166E8 sum: 3.26265626E9 avg: 8.1566406E8


collin@collin-workstation:~/Documents/school/cs-471/concurrency$ ./run.sh 4
total time(ns): 5121243
totalstats: max: 2.65897664E8 min: 1.44721152E8 sum: 3.09742822E9 avg: 1.93589264E8
```

```
collin@collin-workstation:~/Documents/school/cs-471/concurrency$ ./run.sh 8
total time(ns): 11650676
totalstats: max: 1.6607969E7 min: 8410366.0 sum: 7.9136979E8 avg: 1.2365153E7


collin@collin-workstation:~/Documents/school/cs-471/concurrency$ ./run.sh 16
total time(ns): 16529604
totalstats: max: 65450.0 min: 32855.0 sum: 1.2499538E7 avg: 48826.32
```