Oblig 4 IN3030 – Finding the Convex Hull of a set of Points in a plane.

Introduction

This oblig is about finding the convex hull of a set of point in a plane. The convex should form a polygon which is lines between a subset of the points where every point is either in the subset or inside the convex. All angels of the polygon are <= 180. The algorithm is performed sequentially or parallel using Threads, and the timing is stored and compared.

Computer specs

- Prosessor: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 3192 Mhz, 6 kjerne(r), 12 logiske prosessor(er)
- RAM: 16GB

User guide

Start with compiling every file in the folder using: javac *.java

Then you just run the Oblig4 file with your two arguments: Oblig4 {n}

N is the number of points which is 100, 1000, 10 000, etc.

Sequential version

The sequential version is starting of creating an empty list for the complete convex that is going to be filled by the recursion. Then we add the MAX_X point and then call the recursion method to find the convex above the line between MAX_X and MIN_X and later below the line. In the recursion method it starts off with calculating the values for a, b and c which is used for in the formula for finding the distance to a point from the line. Then some more variables are initiated to help keep track of the index and distance to the furthest point from the line. After that a for loop is starting and inside the for loop the distance to a point is calculated and if the distance is positive it is added to a list and the two variables created before the loop is updated. As the for loop does its thing, after x iterations a check is in place to find out if it is necessary to do more recursion calls to find the convex hull.

Parallel version

For the parallel version it is started off with creating the result list and then starting two threads. Why two threads? It is because I want to do only two recursion calls that can be done with one thread each. Inside the Worker-class's run method we are checking what level of recursion we are on and if it is more or equal to the set max level of recursion the current thread does the job in a sequential matter. It is only when the depth level is lower that the recursion is done using more threads. When the depth is not max the method starts the same way as the sequential but when the sequential version is calling more recursion the parallel version creates more threads that does the work. Everything else is pretty like the sequential version.

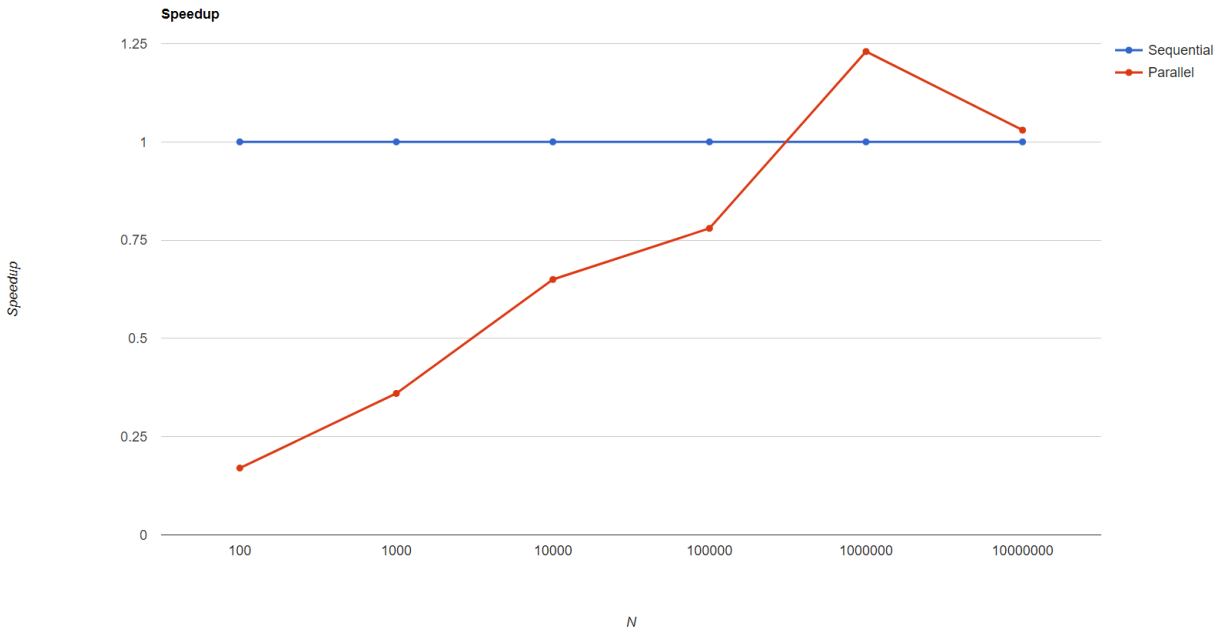
Implementation

The program works with starts off with getting the number n from the user input. Then it creates the necessary variables and starts off with the sequential version. It is done 7 times because the JVM need some warmup to work more efficiently. For each run the time is taken and stored in an array including the actual call to the sequential version. What the version is doing is explained more above in the sequential version chapter. After the loop is done the times are sorted and the median time is printed out. Then the same is done just by calling the parallel version instead. After that loop is done the times are sorted and then the median time is printed out along the calculation of the speedup. After each loop there is an if-check to draw the convex hull if n = 100, but as of now it is commented out because it is disturbing the flow of the program. And that is it.

Measurements

Here are some measurements of the different version.

	100	1000	10 000	100 000	1 000 000	10 000 000
Sequential	0.29 ms	0.68 ms	2.73 ms	10.24 ms	104.32 ms	1736.47 ms
Parallel	1.76 ms	1.93 ms	4.19 ms	13.20 ms	84.58 ms	1683.42 ms
Speedup	0.17x	0.36x	0.65x	0.78x	1.23x	1.03x



I think this visualizes the different version okay. If I would comment on something it would be that the speedup is never that great and even went down for the last value of N. This might be some inefficiency in the parallel program or just a hiccup in the code where it is stuck a bit longer than it should.

Conclusion

In conclusion the sequential version I got was a lot cleaner and better than what I initially delivered and made it easier for me to try and do a parallelization. I tried my best and got it to run but the convex is not correct. There are way more points added to the final convex than it should be. I have tried my best to figure out the problem, but I cannot manage to find a solution. Because of this I have a suspicion that it might be more efficient when the correct amount and correct points is in the result.