Oblig 5 – Convex Hull

Report about the runtimes and implementation of a parallelized program to find the convex hull of a list of 2D points.

# Compiling and Running it

Compiling:

* javac \*.java

Running:

* java Oblig5 [n] [k]

N is the number of points, and k is the number of threads to use.

# Parallelization

I split the list of points into groups, one group per thread, and then I found the convex hull for each group. I then combined those lists and found the convex hull for all of them sequentially.

This means I do extra work to combine, however, the amount of sequential work is a fraction of the total number of points, and each thread deals with 1/k of the points before that.

However, it also means that the amount of sequential work depends on the number of threads, so the ideal number of threads depends even more on the amount of points, as the number of thread changes more than just the time it takes to create the threads.

# Measurements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Seq | Par | Speedup | Threads |
| 100 | 0,214 | 4,716 | 0,045 | 4 |
| 1000 | 0,949 | 5,046 | 0,188 | 4 |
| 10000 | 9,034 | 9,027 | 1,001 | 4 |
| 100000 | 22,624 | 15,163 | 1,492 | 4 |
| 1000000 | 202,632 | 90,141 | 2,248 | 8 |
| 10000000 | 1302,288 | 1054,778 | 1,235 | 16 |
| 100000000 | 16571,521 | 9772,186 | 1,696 | 32 |

All times are listed in milliseconds. All measurements were run on Austur at UiO.

It should be noted that the runtimes when testing can vary a lot since a lot of people can access and use that server. When testing you can get significantly worse speedup at random.

# Conclusion

This is a functional parallel implementation of the algorithm. It is not the most efficient, but does achieve a speedup. To gain better speed up you could use threads when combining the results as well (thread 1 combines thread 1 to 4, thread 5 combines 5 to 8, etc) which would mean you have less points to combine sequentially in the end.