sameera Oblig 4 report

Introduction:

This report is for oblig 4 “Convex Hull” in IN3030 2024.
I have tested my code on a laptop with an AMD Ryzen™ 7 4700U-prosessor 2.00 GHz with 8 cores. It is a Yoga Slim 7 with Base Frequency 2.0GHz and Max Frequency 4.1GHz. Ram is 16gb. Cache is: 4MB L2 / 8MB L3

User guide:

The program takes three parameters, two is required, and one is optional.

1. The required parameter n is used for number of points in the graph.
2. The 2 required parameter is for the seed of generating these n points.
3. The third parameter k is the number of threads that the program will use. If k is 0 or left empty, the program uses the number of cores on the machine.

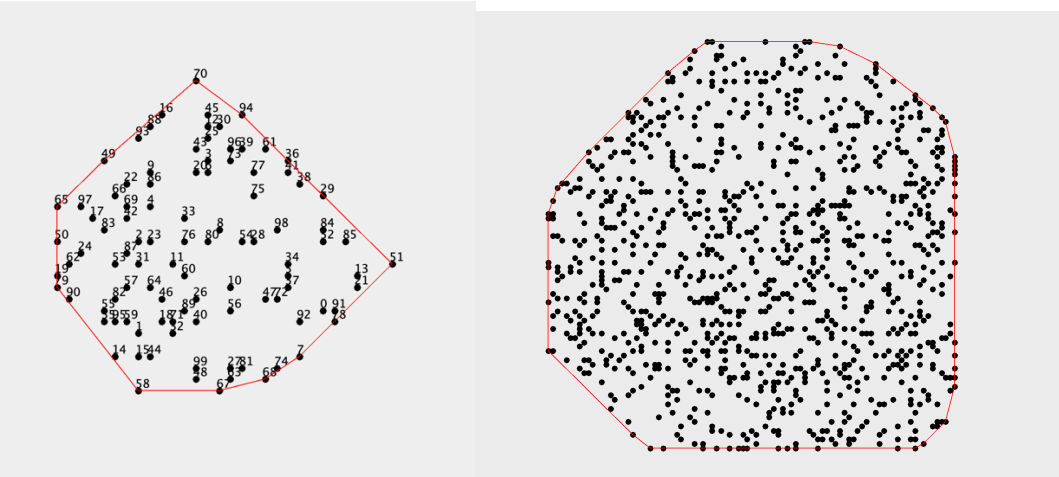
Example:

java Oblig4 n seed

Here the program uses the number of cores on the machine. The program executes both the sequential and parallel version and gives time in milliseconds.

Parallel Convex Hull:

The constructor takes the points coordinate and a list of point indices to be processed. The computeConvexHull method starts the process. It starts by identifying the indices of points with the minimum and maximum x-values. Then two threads, leftThread and rightThread, are launched, each handling half of the point set relative to a line defined by the extreme points. The processing is managed by the HullWorker class. After the threads complete, the results from each are combined to form the complete convex hull. Depending on the subset size and the recursive depth, HullWorker class decides whether to proceed with further parallel processing or switch to sequential computation. In the parallel execution it recursively partitions the points based on their positions relative to the central dividing line. If the subset is substantial and depth allows it further splits and processes the subsets in new threads.



Convex hull found for seed 42, with n=100 and n=1000

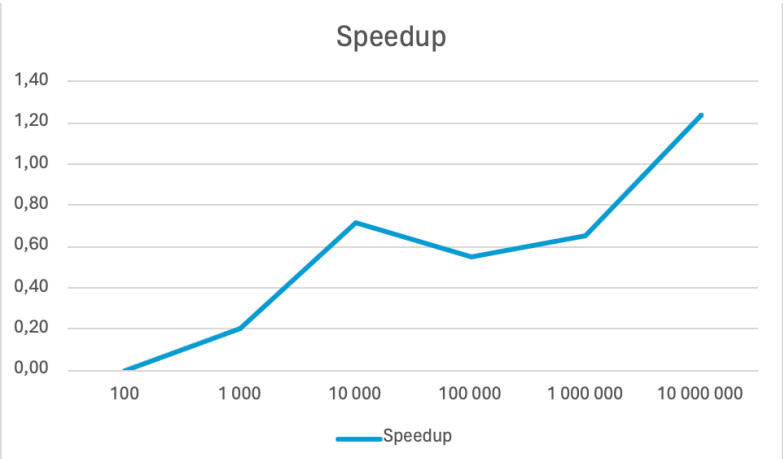
Measurements:

All times are in ms

Convex Hull			
n	Sequentia	Parallel	Speedup
100	0	2	0,00
1 000	1	5	0,20
10 000	5	7	0,71
100 000	22	40	0,55
1 000 000	129	198	0,65
10 000 000	803	647	1,24

The table shows that for a dataset of 10 million points, the parallel implementation of the convex hull computation is indeed faster than the sequential one, with the parallel approach taking 647 milliseconds compared to the sequential time of 803 milliseconds, therefor achieving a speedup of 1.24. For the other dataset sizes, the sequential approach outperforms the parallel one.

Graph showing speedup:



As n grows, the speedup fluctuates but remains below 1 until n reaches 10 million. At this point, there is a noticeable change where the speedup exceeds 1, indicating that the parallel version is performing better than the sequential. This suggests that the benefits of parallel processing become significant when handling very large datasets.

Conclusion:

The sequential approach performs better for smaller datasets, while the parallel implementation is better when handling very large datasets, particularly at 10 million points. This highlights the importance of choosing the right approach based on the dataset size to optimize efficiency.