Report Oblig 5 IN3030

Haiyuec

Processor:

Intel I7 – 8550U

Base frequency: 1.80 GHz

Max Turbo frequency: 4.00 GHz

4 cores 8 Threads

Cache: 8MB SmartCache

RAM: 16GB

How to run the program:

javac *.java

java Main {flag} {N}

Flags:

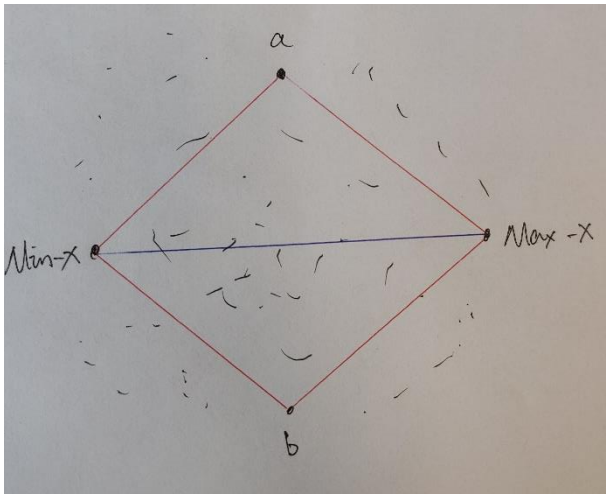
- test-seq:
  Uses the TegnUt to draw the sequential enfolding to the screen.
- test-para:
  Uses the TegnUt to draw the parallel enfolding to the screen.
- run:
  Runs the program and prints the running times.

Parallel Implementation explanation:

- Step 1:
  - Start max number of threads and find the index of max-x and min-x in parallel.
  - Main thread initiates 4 RekWorkers before waiting to collect the results.
- Step 2:
  - All normal worker-threads find the two points that are the furthest away from the "middle line", one above the line and one under the line.

- Normal workers would also divide the points into two subsets of above and under the line.
- The normal workers would terminate after this step.
- Main thread collects the results and set the line-start and the line-end variables in the 4 RekWorkers. The main thread also set the “where_to_search” list of indices in for the RekWorkers.
- Main thread starts the RekWorkers.

- Step 3:
  - The RekWorkers do the recursion in parallel, each searching a subset of the total points.
  - When the additional calls of recursion is needed, the RekWorker would check the amount of available threads.
    - If there is exists less threads than the pc has cores, this RekWorker would start a new RekWorker object, and assign one of the recursion calls to the newly started RekWorker.
      - The RekWorker executes the other recursion call and collects the result of the child-thread when it is finished.
    - Else
      - The RekWorker would execute both of the recursive calls it self.
  - Main thread stiches the results together when all the Rekworkers have finished.

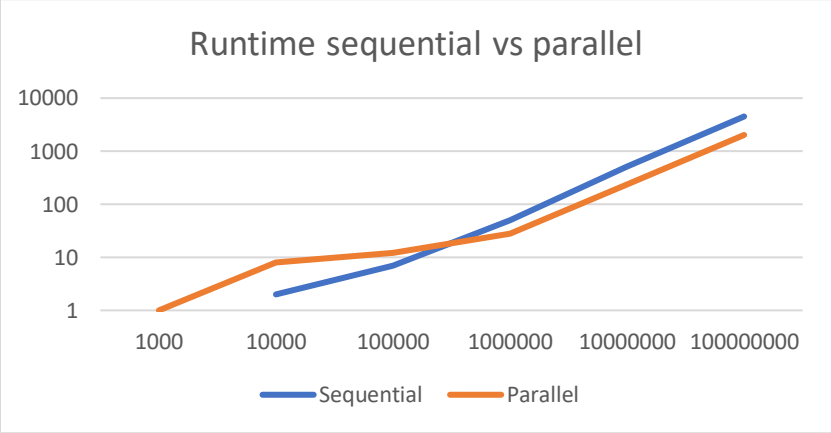


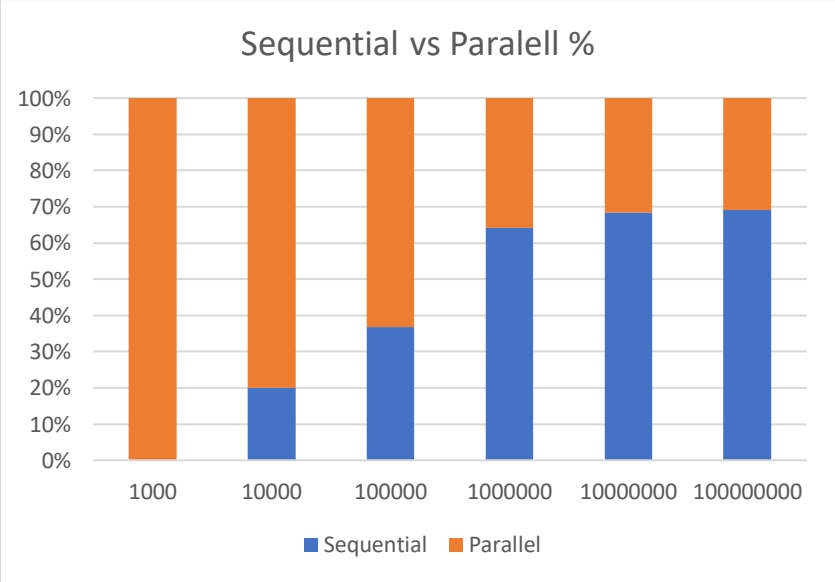
I have chosen to always start 4 RekWorkers, because by finding the points max-x, min-x, a and b, we can already form 4 separate lines, which means that the problem is divided in 4 pieces. This is true for this problem no matter the number of points.

Runtime data:

Runtime measured in ms

n	Sequential	Parallel	speedup
1000	0	1	0
10000	2	8	0.25
100000	7	12	0.5833333
1000000	50	28	1.7857143
10000000	511	237	2.1561181
100000000	4508	2023	2.2283737





Conclusion:

We can see that the parallel implementation is clearly faster than the sequential implementation when n exceeds 1mil. I think that this can be improved on by having another method for assigning tasks to threads. Here I would start a completely new thread for the new recursion calls, which takes time. I have also noticed that the running time for this particular implementation is not very stable. Because the new Threads are started during the recursion, and there is a condition for starting new threads, the location of where the new threads are working on is different every time we run the program.

To solve these two problems, I have been thinking about another implementation. I Suppose that we could implement a type of context switch.

The form of recursion in this problem is a tree recursion, which means that we can save the partial results from each call (we would find a new point in every recursion call) as a node in a tree, for the particular case of a 2d plane, this would be a binary tree. We would need to have a monitor that stores the tree, a list of task (call-parameters) and their position in the tree. The threads would check the monitor for new tasks, and then executes the function call with the call-parameters from the monitor.

For each recursive call, the partial result is saved in the tree. If this call spawns new recursive calls, the call parameters of one of the recursive calls would be pushed to the monitor, while the other call is executed by the current thread. The threads would check the monitor until there are no more tasks.

When the threads have finished generating the tree, a infix-traversal of the tree would generate a list of the indices. This traversal could also be parallelized.

I think that this implementation would give a stable run time, but I also see that this implementation would involve a lot of synchronization, which probably is not a good sign for its runtime.