INTRODUCION

This report is about finding the convex hull in a set of points.
Since the last time I've changed my computer. Now I have this MacBook Pro with the ARM processor.

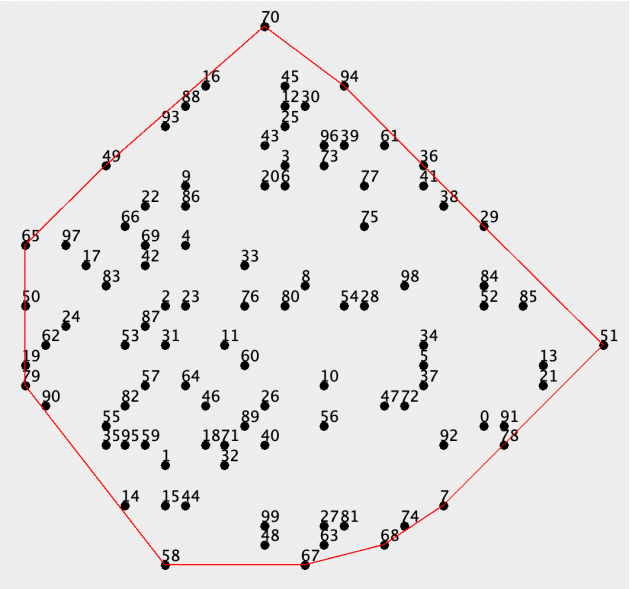
Clock Rate	3220 MHz
Level 1 Cache	2.3 MB
Level 2 Cache	28 MB
Level 3 Cache	16 MB
Cores / Threads	8 / 8
Transistors	33700 Milion
Technology	5 nm

USER GUIDE

To use the program you should input [n] (seed random number) (threadNumber) where:

- *n* is the number of points we use, it is mandatory.
- *seed* is the seed number used to generate the points. If not set, it is used 42 as seed.
- *threadNumber* is the number of threads we want to use. If not set, the program will choose the number of threads automatically, based on your machine.

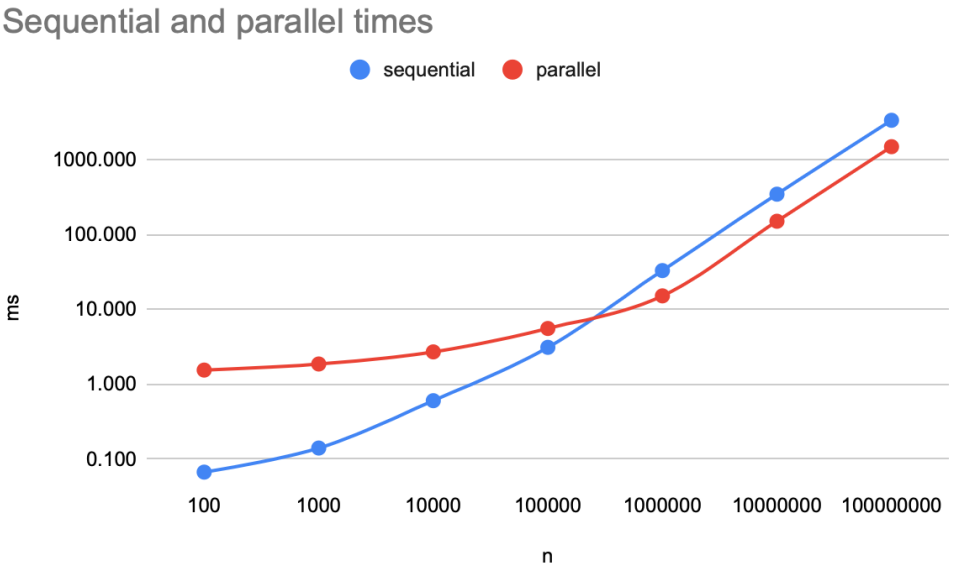
PARALLEL VERSION - IMPLEMENTATION

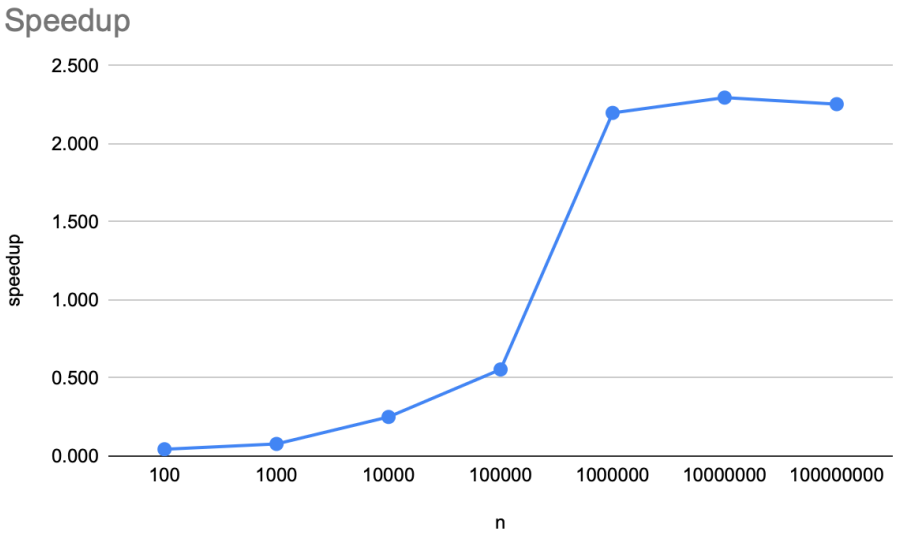


My parallel solution is based on having maximum *threadNumber* active threads at the same moment. So, it's possible that, in a certain moment, there are more than *tN* threads generated, but not everyone is working.
To do so I begin generating two threads, left and right side of the recursion, and each branch generates other left and right threads until it doesn't.
The branch will not generate other threads after a certain level; for instance, if we have 8 cores it will stop when the 8 threads of the 3rd level are generated.
To compute the number of levels that generate the threads I use *log2(threadNumber)*, which I code as: *(int) (Math.log(threadsNumber) / Math.log(2));* because there is no actual log2 in the Java Math library.
After that level the program will be executed as if it was sequential.
In order to synchronise, every level waits for the two children before appending the resulting points. Doing so, I always have all the points in the right order, left to right.

MEASUREMENTS

Number of points	Sequential [ms]	Parallel [ms]	speedup
100	0.067	1.548	0.043
1000	0.141	1.866	0.078
10000	0.604	2.707	0.251
100000	3.114	5.557	0.555
1000000	32.913	15.132	2.198
10000000	344.079	149.812	2.296
100000000	3331.097	1480.311	2.254





As always, creating a thread takes time. So it makes sense creating them only if the tasks are long enough.
As we see, after 5*10^5 the parallel solution beats the sequential one, because of that.

Also the speedup jumps at 2.3, but it stays there from number more than 5*10^5.

CONCLUSION

With big numbers the parallelization always beats the sequential solution.
Parallelizing having a large amount of numbers is never a waste of time.