Assignment #1: Generic Algorithm for the Knapsack Problem

After reading and understanding the code the first thing was to change the Main.class program to run the sequential version of the program 30 times, registered the time of each iteration and at the end of all 30 and saved the times in a csv file where later I would use to make the datasets in python to evaluate the performance.

Afterwards I pondered which functions I would parallelize and decided only to apply parallelization and concurrency inside the run() function. The bestOfPopulation() and populateInitialPopulationRandomly() being relatively simple functions that require low computational intensity didn't seem worth to parallelize, not only that but the extra steps to synchronize the threads would impact the little performance advantage they would get. I actually tested in both new classes the difference between having populateInitialPopulationRandomly() parallelized and sequential and the sequential version was actually much faster. I did the same for the tournament() and bestOf Population() and didn't notice any significant difference between having this 2 methods parallelized and in sequential, while the newly parallelized code in run() made a very big difference as we are going to see next.

Meanwhile the tournament() function seamed promising at first due to the tournament size but in the code of the professor only occurs 3 times, for the same reason as the previous two I kept them in sequential, if the tournament size sample was a bigger number it would be more worth it to parallelize.

In short, I only parallelized step 1, 3 and 4 inside the run() function in 2 different ways in 2 different classes. On the first class KnapsackGa.java I created 3 auxiliary functions for each step where in each one I created the threads array and a phaser to parallelize the sequential code with phaser.arrive() at the end of each phaser and a phaser.awaitAdvance(0). and called the functions in run(). Then on the Main.java class I made 30 executions for each number of cores, in my case I used 4, 8 and 12 cores and registered the time.

Then on the second class KnapsackGaMode2.java I parallelized the code in the run() function without using any auxiliary functions and only created 1 list of threads that I used in all the 3 parallelized steps in order to see if it had any impactful performance issues. I also did not use a phaser like the last and used thread.join() with a for each cycle. In both classes I got very similar results in terms of performance and the same logic end results.

In conclusion, as we can see by the different results below I was capable of significantly improve the performance of the program running in almost a third of the time as the original.

Pc specs:

AMD Ryzen 7 3700Uwith Radeon Vega Mobile Gfx

8gb RAM memory

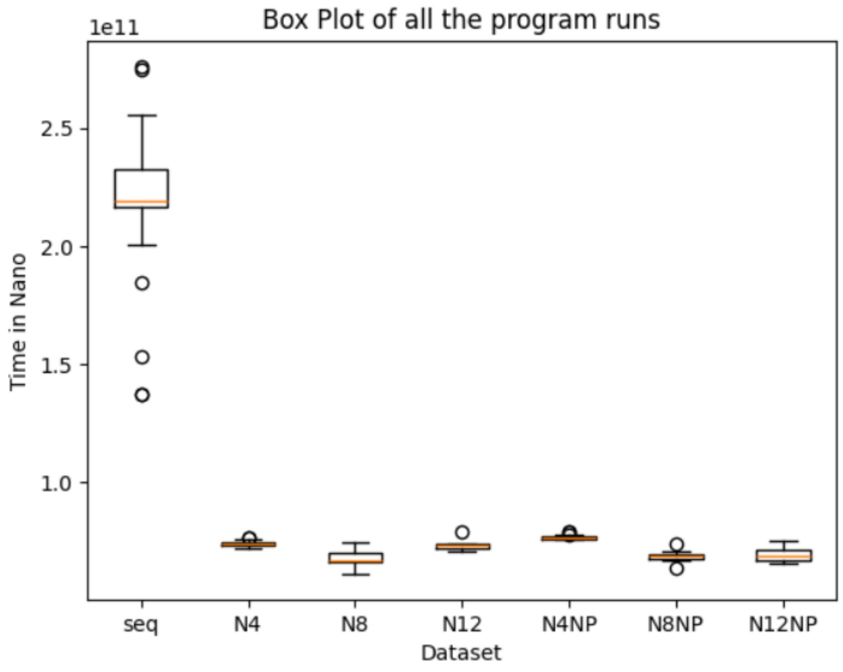
AMD Radeon™ RX Vega 10 Graphics

Mann-Whitney U Test between Sequencial and the version with the best time (8 cores and phaser):
P-value: [3.01985936e-11]

Since the P value is so small we can conclude that there is a significant difference between the two datasets (H1).

- The null hypothesis (H0) would be that there is no significant difference between the two sets of running times
- The alternative hypothesis (H1) is that there is a significant difference.

```
Running Times (minute):
Sequential:  109.410882065
Paralel w/ 4 cores and phahser:  37.14522738666667
Paralel w/ 8 cores and phahser:  33.94290766333334
Paralel w/ 12 cores and phahser:  36.49614688166666
Paralel w/ 4 cores and no phahser:  38.375125425
Paralel w/ 8 cores and no phahser:  34.35095742666667
Paralel w/ 12 cores and no phahser:  34.531714674999996
```


Box Plot of all the program runs
1e11
Time in Nano
seq
N4
N8
N12
N4NP
N8NP
N12NP
Dataset