Homework 4 – results

Group 17

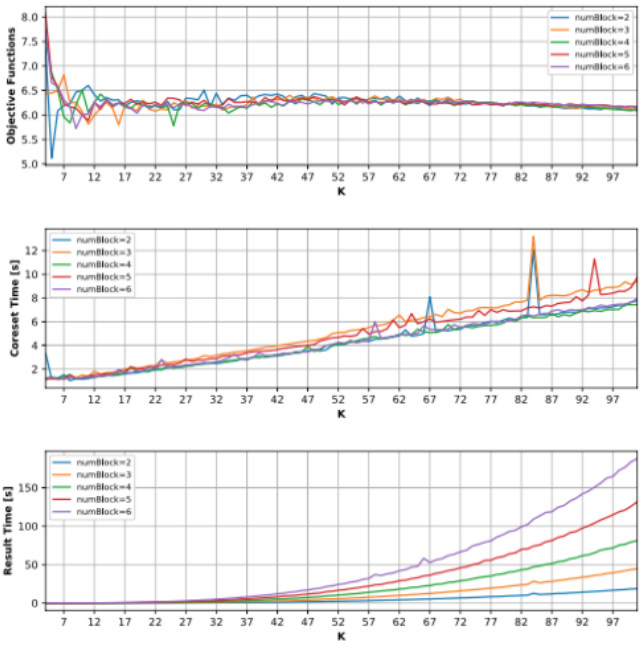
First, we tested our code for different configurations of k and $numBlock$ locally with a smaller dataset. The results displayed in the graphs below show a constant trend for the objective function for values of $k > 10$, while, as we expected, the time for the elaboration increases: the coreset time presents a linear increase while the result time a polynomial one.

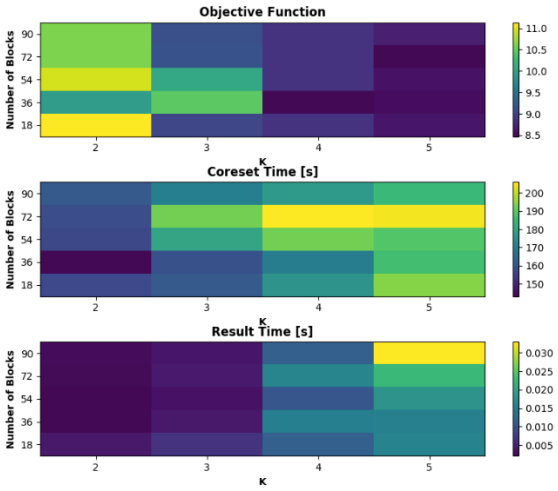
Then we evaluated our code on the cluster with the dataset of 10^6 samples; we first decided to perform a brief grid search on the following parameters:

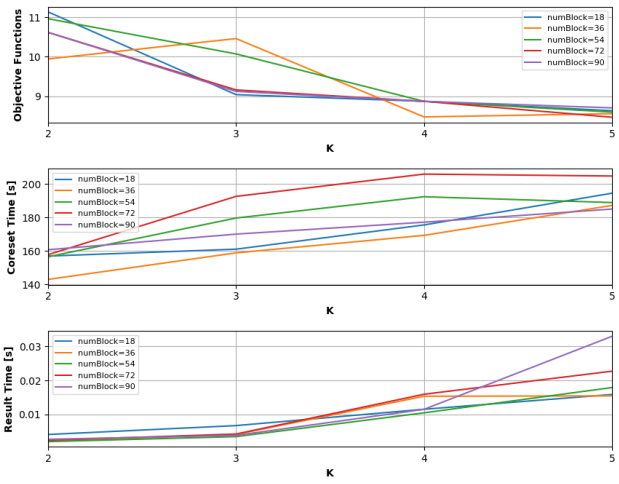
$$k = [2, 3, 4, 5];$$

$$numBlocks = [18, 36, 54, 72, 90];$$

using 2 cores for each node in the cluster, for a total of 18 cores. One thing to highlight, which we expected, was that for lower value of k the obj. function assumes higher values; however, as we noticed above, with a higher value of k, time also increases. For this reason, we tried different values of $numBlocks$ keeping k small; in fact, we found that using as number of blocks a multiple of the cores reduces the execution times by better-balancing the work load of each core. What we found is displayed in the graphs below:







At last we executed our code on all 72 cores, using as parameters $numBlocks = 72,\ k = 4$. The results we found are: **Obj. function** = 7.42; **Coreset time** = 218.62 s; **Result time** = 2.81 s. The objective function is small enough and computed in a reasonable time thanks to the exploitation of a number of blocks high enough and equal to the number of cores. Doubling the number of blocks instead we found the same obj. function and higher computation times: Obj. function = 7.42; Coreset time = 230.03 s; Result time = 8.49 s. This tells us that there's no need to further increase $numBlock$ since it produces only a greater overhead on the number of necessary computations.