Report

During this lab we were asked to compare the performance of the code depending on the different parameters, such as number of cores, space dimensionality and etc. The total running time was retrieved by using profiling. The computer parameters are as following:

In the table below you can see the performance table of the code depending on the number of cores and search space dimensionalities for the sphere/parabola benchmark function:

Number of cores	Space Dimensionality	Total running time (s)	Total running time after improvement(s)	Best fitness
1	3	5.392	4.689	0
1	10	7.781	5.765	3.5465e-17
1	20	8.413	8.353	3.5079e-06
1	30	6.165	6.260	0.19714
2	3	6.618	6.068	0
2	10	6.452	7.590	3.5465e-17
2	20	10.472	7.185	3.5079e-06
2	30	7.708	7.206	0.19057

For benchmark function Y = sum(x^2) for the range [-100;100] global minimum is equal to zero. According to the table it can be seen that in case where Number of cores = 1 the best fitness is 0 for dimensionality 3 which shows the closest correct answer in comparison with D = 10,20,30. In case where Number of cores = 2, we can also see that the closest result to 0 was produced when D=3.

As per the improvement of code, I chose one loop (in figure shown below) and vectorized it and as a result we got less total running time of code for D = 3,10,30. Increasing number of cores didn't slow down the code, but increased accuracy of best function in case of D =30.

The code part that was changed from:

```
19      %We need to have one psoOut struct for each run: make a struct array with
20      %each element initialized to be the same as the first
21
22 -    for lpruns = 2:nRuns
23 -       psoOut(lpruns) = psoOut(1);
24 -    end
```

Into:

```
25
26      % optimize code by vectorizing the for loop
27 -    psoOut(2:nRuns)=psoOut(1);
28
```