Parallel Computing
K-Means Algorithm

1st Rodrigo Rodrigues
Informatics Department
University of Minho
Braga, Portugal
pg50726@alunos.uminho.pt

2nd Daniel Azevedo
Informatics Department
University of Minho
Braga, Portugal
pg50311@alunos.uminho.pt

***Abstract*—*K-means* clustering is a method of vector quantization that aims to partition N observations into K non-overlapping subgroups (clusters) in which each observation belongs to the cluster with the nearest mean (cluster centroid), serving as a prototype of the cluster. In an effort to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible, the *K-Means* algorithm computes centroids and repeats until the optimal centroid is found. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster. In this paper we will analyse an optimized sequential version of the algorithm developed with the goal of minimizing the total execution time. Furthermore, the performance in terms of execution time will be measured considered different metrics.**

***Index Terms*—K-means, algorithm, performance, optimization, metrics**

I. Introduction

The main goal of this assignment was to evaluate the learning of code optimization techniques, including the code analysis techniques/tools. Using C, an optimized sequential version of the *K-Means* algorithm was developed, based on Lloyd's algorithm, aiming to partition N observations into K subgroups (clusters) in which each observation belongs to the cluster with the nearest mean.

We started off by developing the first version of the algorithm, which served as the basis for applying optimization techniques, such as the exploration of spatial and temporal locality, with the help of code analysis tools.

Furthermore, we performed tests using different versions of the algorithm each time with better efficiency due to use of optimization techniques. Lastly, the obtained results were analysed and compared with the expect results.

II. *K-Means*

A. Implementation

The sequential version of *K-means* consists of four main steps. For the first step, we want to specify the number of clusters and data points, initialize the data with random values, initialize the K clusters with the coordinates of the first K samples and then assign each sample to the closest cluster using Euclidean distance.

Secondly, we proceed to calculate the "centroid" of each cluster by taking the average of the all data points that belong to each cluster.

In the third step, each sample is assigned to the closest cluster using Euclidean distance.

In the last step, the "centroid" value, calculated in step two, is compared to its previous value and, if the values are equal we will go back to step 2, otherwise the algorithm ends.

During the implementation of the *K-Means* algorithm we tried different versions in an effort to reach the most accurate results.

The first version strictly follows what has just been described above and was implemented without taking the performance into account, with the single goal of achieving the correction of the algorithm. However, there was a lot of room for improvement regarding spatial locality and memory accesses.

In the second version, we avoided calling the function responsible for adding the distances of the points, instead we performed this calculation in the section where it was called. Performing this optimization allows the program to make fewer memory accesses and there are also no instructions responsible for invoking the function.

Finally, in the third and last version of the algorithm we replaced the calls to the pow function, from the math library, with multiplications. The calls to the sqrt function were also removed since in the algorithm the distances between points and clusters can be calculated without the square root because the goal is to compare them in order to find out the closest cluster.

TABLE I
Different versions of K-Means

Version	T_exe(s)	#CC	#I	CPI	L1_DMiss
1	25,793567831	84356595283	147561449440	0.6	849973549
2	12,964531861	40825039410	55269464352	0.7	523566428
3	3.965874504	13056362363	27112069822	0.5	53462245

```
PONTO soma_pontos(PONTO a, PONTO b){
    PONTO res = malloc(sizeof(struct _ponto));
    res->x = (a->x) + (b->x);
    res->y = (a->y) + (b->y);
    res->k = a->k;
    return res;
}
```

As can be seen above, upon the removal of the function *soma_pontos*, the number of instructions decreases due to the fact that no longer exist instructions responsible for invoking this function. It was also possible to reduce the number of memory accesses, since now there isn't a need to pass input to the function.

From the second to the third version of the algorithm we can see that the number of L1_DMiss decreases by about 90%, which could be due to the removal of struct PONTO and instead using float arrays to store the points.

As we know, Spatial locality is the concept that the likelihood of referencing a resource is higher if a resource near it was just referenced. In this version of the program we changed the way points were stored by replacing the previously used data structure (an array of struct POINT) with two float arrays, one for the x coordinates and the other for the y coordinates. This change allowed us to explore the spatial locality of the data.

Other metrics such as the number of instructions and number of clock cycles also had a decrease positively affecting the CPI of the algorithm.

B. Analysis of Possible Optimizations

There are some optimizations that can be applied to this algorithm, in order to decrease execution time, such as loop unrolling and vectorization. These techniques are applied by the gcc compiler, when using specific flags to achieve the desired purpose.

We analysed the impact of possible optimizations using gcc compiler flags like -O0, -O1, -O2 and -O3. Each of the flags serves a certain purpose: -O0 reduces compilation time but does not use any optimizations; -O1 enables the core optimizations in the compiler; -O2 optimizes even more and produces vector instructions; -O3 turns on all optimizations specified by -O2 and also turns on the -finline-functions, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload and -ftree-vectorize options.

The table below demonstrates the execution time of the algorithm, considering an input size of 10000000 points and 4 clusters, using different code optimization flags. As expected, from -O0 to -O2 the execution time decreases as the level of optimization increases. -O2 clearly performs better than -O0 and -O1 which can be seen in the execution time of the program. However this is not the case for -O3 when compared to -O2. Unlike expected, even though -O3 optimizes yet more, more satisfactory results were obtained with -O2. By analyzing clock cycles in the -O3 version we can see that there are fewer instructions and we can verify that the instructions executed with -O3 are more complex than the instructions executed with -O2, which could be the reason for a substantial increase in execution time.

TABLE II
OPTIMIZATIONS IN SEQUENTIAL K-MEANS

Optimization	T_exe(s)	#CC	#I	CPI	L1_DMiss
-O0	19,890122882	64997234707	85775371528	0.8	56873083
-O1	5,915169665	17152044779	33919860456	0.5	53921414
-O2	3.965874504	13056362363	27112069822	0.5	53462245
-O3	8,316428529	26585737338	19813661043	1.3	54520189

III. CONCLUSION

Given the completion of the project, we present a critical and comprehensive view of the work developed. On one hand, we consider that the final sequential version developed for the K-Means algorithm proved much more efficient due to the application of concepts such as spatial and temporal locality and other optimization techniques. We also consider that the tests and metrics applied to the different versions are diversified enough to correctly evaluate the algorithm presented.

On the other hand, the project could benefit from some upgrades like improving parallelization of the algorithm which is a study that can be done in the future regarding the benefits presented in shared memory.

Furthermore, this report shows a detailed explanation of the implemented K-Means algorithm and tests developed, allowing the reader to fully understand the choices made and each reason behind them. To sum up, we consider that the work developed is complete, the problems that arose were overcome and the goals of the project were achieved.