K-Means Clustering Optimization with OpenMP and Comparison Study

Burak Topçu
CENG444 - Parallel Programming Patterns *Term Project*

Department of Computer Engineering
Izmir Institute of Technology

***Abstract*—K-means clustering algorithm [1] is a contemporary algorithm being used by document clustering, identifying crime-prone areas, customer segmentation, insurance fraud detection, public transport data analysis, clustering of IT alerts…etc. With respect to the workload and application, the execution time for this clustering can be a significant factor and tradeoff for the industrial and academic works. In this paper, I tried to capture the patterns can be issued as parallel and made a comparison study between the other optimizations. The code I implemented is written in C programming language and benefits from the OpenMP functions detailed in the following sections of the paper.**

Keywords—parallel patterns, OpenMP, C programming language

I. INTRODUCTION

K-means clustering algorithm is a popular algorithm that is being used to bring together of similar data samples. For a dataset, each cluster specifies a class that has similar in terms of identifying similarity type such as closeness on a surface o pixel density and so on. This clustering method is highly beneficial and utilized for the data preprocessing in data mining field to eliminate irrelevant samples from data and separating dataset into inter class sets in deep learning applications.

In this work, it is tried to optimize K-means clustering algorithm with the help of parallelization tools implemented in OpenMP library. The optimization efficiency depends on the belonging computer hardware resources since the parallelization amount will increase in parallel with respect to the quality of the computer resources. There are 2 parallel programming patterns come forward in the K-means as mapping and reduction methods. Furthermore, fusing method in the context of parallel programming is implemented to fuse mapping and reduction methods. For each of those steps, performance results are recorded. In the following sections, occupying parallel patterns, experimental procedures and results, and performance comparisons is described.

II. REDUCTION, MAPPING AND FUSING METHODS

A. *Reduction Method*

The reduction clauses are data-sharing attribute clauses that can be used to perform some forms of recurrence calculations in parallel as in figure 1 and figure 2. Reduction clauses include reduction scoping clauses and reduction participating clauses. Reduction scoping clauses define the region in which a reduction is computed. Reduction participating clauses define the participants in the reduction. Reduction clauses specify a reduction-identifier (+, -, *, ^, && and ||) and one or more list items [2].

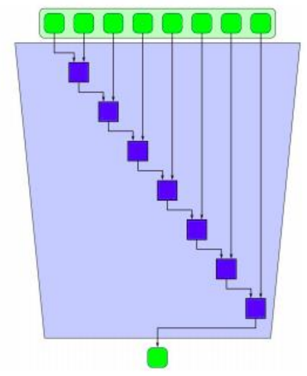


Fig. 1. Visualization of a serial process [4].

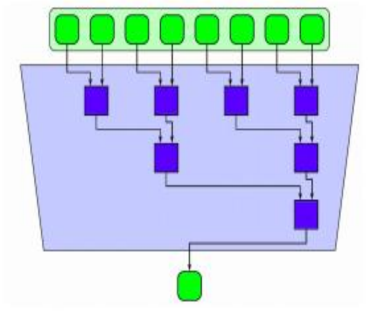


Fig. 2. Visualization of the reduced process mentioned in fig 1 [4].

Mapping Method

Mapping is a method to use hardware resources especially cores with the independent data blocks efficiently. As name implies task or the data is partitioned and distributed to the cores in a parallel and balanced way as in figure 4.

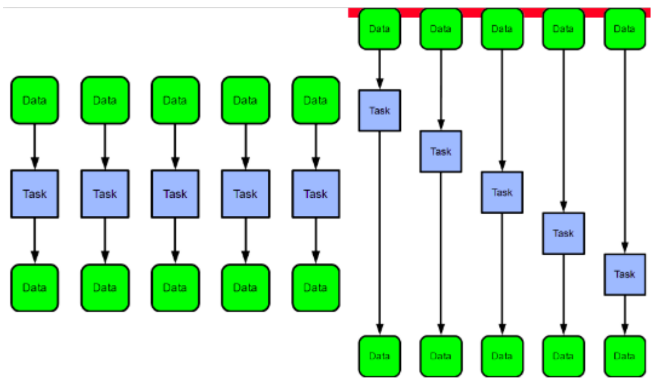
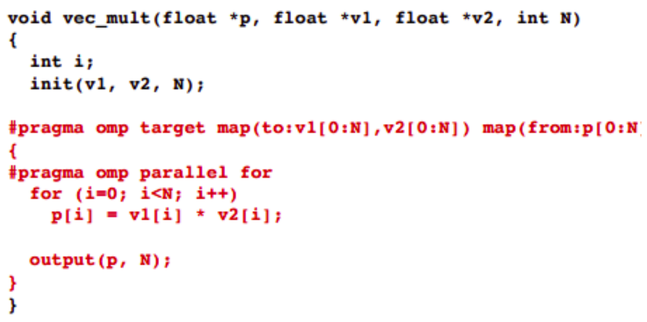


Fig. 3. Visualization for the mapping (left one is the mapped, right one is the sequential) [4]

OpenMP has a *map* clause [2] that carries out the described task internally or users can create their own mapping themselves. In the code example shared in figure 4 includes 3 different vectors pointed by v1, v2 and p pointers and N variable. Each of the arrays have N number of elements. The map(to:v1[0:N], v2[0:N]) clause is used to specify that vector1 and vector2 are the operand vectors and map(from:p[0:N]) is used to specify that p is the resultant array.



```
void vec_mult(float *p, float *v1, float *v2, int N)
{
  int i;
  init(v1, v2, N);

#pragma omp target map(to:v1[0:N],v2[0:N]) map(from:p[0:N
{
#pragma omp parallel for
  for (i=0; i<N; i++)
    p[i] = v1[i] * v2[i];

  output(p, N);
}
}
```

Fig. 4. Mapping example code implemented onto C programming language with OpenMP map clause [3] .

B. *Fusing Method*

Fusing is the method that fuses two sequential patterns by removing the store operation of the firstly calculated results and cascading them with the second pattern. Fusing can be used many ways apart from the mentioned work but, this is the main contribution of the fusing. For example, fusing 2 for loops that carry out same job just reduces the amount of overhead of the thread specializer. In figure 5, the effect of the fuse method is visualized. The unnecessary store operations are removed from at the last step of the mapping and the output of this mapping is cascaded with the reduction as I did for my work.

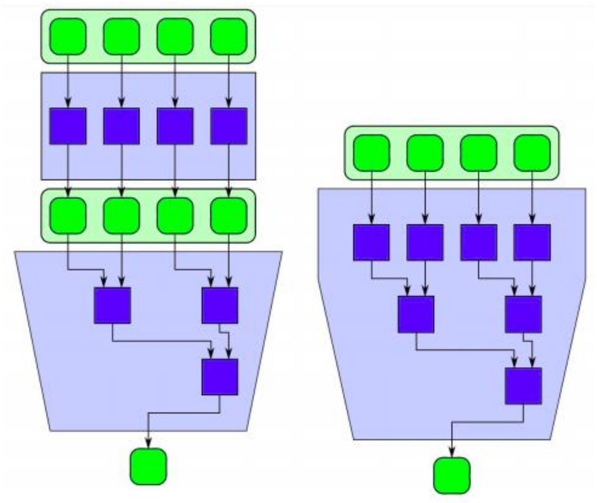
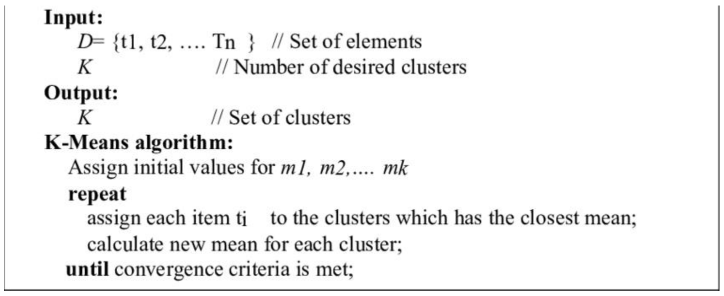


Fig. 5. Fusing of the mapping and reduction patterns [4].

There is no specific fusing clause in the OpenMP. Instead, the user has to realize and implement its code by being aware of the fusing of patterns.

III. Experiments

Experiments on K-means clustering is enhanced by introducing mapping and reduction patterns and fusing them. Also, performance is measured in terms of the elapsed time to complete clustering. At first without any parallelization, the K-means algorithm is implemented with respect to the pseudo-code given in figure 6.



```
Input:
    D= {t1, t2, …. Tn }   // Set of elements
    K                     // Number of desired clusters
Output:
    K                     // Set of clusters
K-Means algorithm:
  Assign initial values for m1, m2,…. mk
  repeat
    assign each item ti  to the clusters which has the closest mean;
    calculate new mean for each cluster;
  until convergence criteria is met;
```

Fig. 6. K-means clustering algorithm pseudo-code.

I have used randomly initialized 1 billion points as dataset and tried to cluster among them. Also, I have carried out my experiments with specifying 10 clusters. Furthermore, I have exported the results into excel files and visualized the clustering results in python to check the correctness. Since our data samples are randomly created points on 2-dimension, it is expected that the plot has to be divided into 10 equal pieces.

Later, I have tried to optimize the code by implementing required mappings, reductions and fusing. Mapping is used in assigning data samples to the clusters which has the closest mean. Also, while creating the dataset and recording the results, mapping is utilized. For calculating the new mean for each cluster, reduction and mapping methods are fused. The reason why we should reduction while calculating the new mean for clusters is that each data sample contributes to the new mean for each iteration. Also, reduction efficiency will increase for the increasing number of clusters. Instead of sequential updates on new mean, parallelized updates with the help of the reduction method are used. Further details are shared in the code.

There are some important criterions that must be taken into account while implementing the K-means clustering. Mapping criterion is significant, since too diverse mapping can result in degrading on the spatial locality usage of higher-level cache. There is a tradeoff between scheduling criterion and the scheduling overhead such that more complicated scheduling has more overhead even if it can provide better spatial locality usage. For example, dynamic or guided scheduling with quarter of the spatial locality fetch sizes result in better accuracy compared to the static scheduling where small number of mappings such as between 4 and 8 activated at the same time. Results for both non-optimized and optimized versions of K-means clustering algorithm are shared in the table 1.

The results are taken from the clustering among 1 billion points and 10 clusters. I did not specify any finalizing criterion since randomness can affect outer iterations such that beneficial randomized dataset can finalize earlier or vice versa. Instead, I have applied 1000 iterations for both re-assigning points to the clusters and updating the centers of the clusters.

TABLE I. Execution time resutls of K-means clustering algorithm

Applied Patterns	*Execution Time*
Without any pattern	424.632 seconds

With mappings and without reduction	73.391 seconds
Without mappings and with reduction	391.561 seconds
With fused mappings and reductions	66.445 seconds

These measurements are highly dependent on the computer hardware resources and interconnecting networking conditions.

IV. Comparisons of the Other Related Works

In the literature, this algorithm is used for different fields to be part of a different sub tasks as mentioned. Thus, the comparison study has to be done by changing just the code flow with the implementation of parallelism techniques. Other things have to be kept as the same. In my work, I have just created random dataset in to test the algorithm. Also, the programming language used to implement algorithm and the library used for parallelization tasks have to be the same since internal parameters of the library such as definitions and structures of the algorithms can disrupt the comparison balance. For example, there are lots of unmatched parameters as in scikit k-means clustering algorithm and mine in terms of implementation details even if I used the same dataset for both.

However, the parallelization patterns for the K-means clustering algorithm are the same that mapping, reduction and fusing them as it can be shown in the Structured Parallel Programming [5] book. In this book, this algorithm is implemented on top of the TBB library. Since speed up amount is not shared, I cannot compare it my results also.

Instead, I can find other implementations that are not published any conference but, they used the same library. One of them [6] can reach 4 times speed up with respect to the simple implementation mentioned in pseudo code of K-means clustering. The other [7] can also reach 3,8 times speed up with respect to non-optimized K-means clustering by using the OpenMP. In my implementation, I can reach 6,4 times speed up with the optimized version of K-means algorithm compared to the non-optimized one.

As a result, I have implemented all parallelization patterns mentioned in the Structured Parallel Programming book with respect to the results among the other implementations. Also, resultant k-means clustering plot is shown in figure 7.

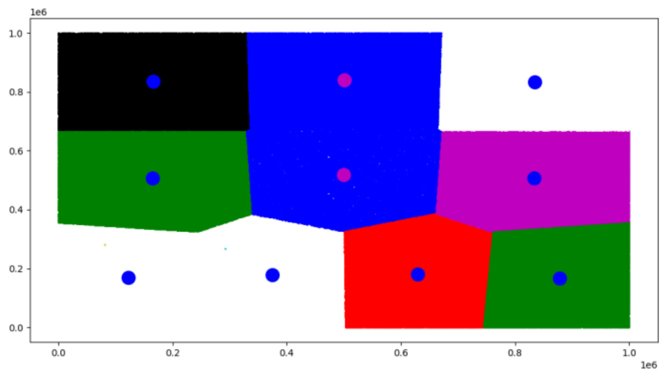


Fig. 7. K-means clustering on the randomly generated dataset.

V. Conclusions

As a result, K-means clustering algorithm can be parallelized by introducing mapping, reduction and fusing of both. For re-assigning the points to the clusters can be mapped and executed on multiple cores in parallel since the belonging data samples are independent from each other. Furthermore, reduction is implemented for updating the centers of the clusters. While updating centers of the clusters, points in each cluster are added iteratively then divided by the number of points for each cluster respectively. This iterative addition can be parallelized with reduction instead of sequential execution. Furthermore, fusing is introduced by combining them. For the future, a dataset benchmark that includes different data types and different parallelization library can be tried together on this algorithm and the effectiveness for different libraries can be observed.

References

[1] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 881-892, July 2002, doi: 10.1109/TPAMI.2002.1017616.

[2] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. IEEE Comput. Sci. Eng. 5, 1 (January 1998), 46–55. DOI:https://doi.org/10.1109/99.660313

[3] Christian Terboven Michael Klemm James C. Beyer Kelvin Li Bronis R. de Supinski, "Advanced OpenMP Tutorial, OpenMP for Heterogeneous Computing slides", 2018

[4] ÖZ, I 2021, Parallel Programming Patterns:"Mapping, Reduce & Scan and Fusing" lecture notes, Izmir Institutue of Technology, delivered 2021.

[5] Michael McCool, James Reinders, and Arch Robison. 2012. Structured Parallel Programming: Patterns for Efficient Computation (1st. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[6] https://github.com/arneish/parallel-k-means/blob/master/Lab1_Report.pdf

[7] https://github.com/JiaweiZhuang/CS205_final_project