Contemporary Engineering Sciences, Vol. 13, 2020, no. 1, 359 - 366
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ces.2020.91615

Parallel Implementations of k-Means in Matlab

Dimitris Varsamis

Department of Informatics Engineering, Computer and Telecomunications
International Hellenic University, 62124 Serres, Greece

Alkiviadis Tsimpiris

Department of Informatics Engineering, Computer and Telecomunications
International Hellenic University, 62124 Serres, Greece

Abstract

In this article a novel parallel function of k-means algorithm is introduced reduceing significantly the computation time, compared to the serial equivalent of k-means in Matlab. A data set generator of hyper-geometrical shapes clusters is additionally implemented for the evaluation of k-means algorithms in different datasets. The implemented parallel variations of k-means algorithm are functional at two alternative states, the state(`k-meansRCP`) where the centroids are assigned randomly and the state (`k-meansRAP`) where the data are assigned randomly to k clusters. The performance of the introduced `k-meansRCP` and `k-meansRAP` were compared to the serial implementations of k-means, on synthetic datasets of varying dimensionality.

Keywords: parallel algorithm, k-means, data generator, hyper-geometrical shapes

1 Introduction

k-means is a popular unsupervised learning algorithm used for solving clustering problems with computational complexity $O(ndki)$ depending on data dimensionality (n rows and d columns), the clusters (k) and the number of iterations (i)[11]. The basic parameter of k-means, is the number of clusters k where the data are classified after an iterative procedure. Specifically, k-means

is very popular in Feature Selection [12], [3], dimension reduction problems and generally in any problem that requires unsupervised machine learning [14].

Large datasets of high dimensionality, drives data clustering algorithms to become computationally expensive and slow [1], so the development of parallel clustering algorithms is a solution for this problem. There are many scientific works introducing parallel clustering algorithms [9], [4], [8] and implementations of k-means using parallel techniques [2].

Matlab is a software tool that supports, conveniently, numerical computations and parallel techniques [6], [10], [13]. Additionally, Matlab supports parallel computations either in cluster of computers or in multicore CPUs [7]. This article presents parallel implementations of the k-means algorithm designed to run on Windows operating systems using the Matlab platform.

The rest of this article is organized as follows: In section 2 there are the methods of serial and parallel implementation of k-means in Matlab and the data set generator algorithm. Section 3 presents results and discussion of k-means, serial and parallel performance using multidimensional datasets. Finally in Section 4, conclusions and future work are presented.

2 Methodology and Datasets

The parallel k-means clustering algorithm programs that used at this study, were developed in order to increase the computational speed compared to the alternative Matlab serial k-means algorithms. Two variants of k-means Matlab functions are presented corresponding to two different ways of assigning the centroids. The parameters that were used as input for all the k-means functions are the the dataset matrix of n patterns and d dimensions, the k number of classes and t the number of iterations of the algorithm for a successful convergence. The Euclidean distance is adopted as the metric distance, due to its predominant use in the majority of researches.

2.1 Serial k-means implementations

A native serial implementation of k-means in Matlab, denoted as `k-meansRC`, select randomly k patterns and sets them as the initials centroids. Iterations of minimizing the sum of the distances of all patterns from the updated centroids of the groups to which they belong drives to he best solution for this approach. A faster implementation of k-means in Matlab, denoted as `k-meansRA`, assigns the patterns of the dataset to k classes randomly and the centroids are calculated after the initial random assignments. Iterations that drives to he best solution minimize, in the same way as above, the sum of the distances of all patterns from the updated centroids. The build in Matlab, serial k-means function, denoted as `Mk-means`, used as a benchmark for computational time comparison between the implemented k-means functions.

2.2 Parallel k-means implementation with Random Centroids and Random Assignments

The execution of k-means faces large computational cost in cases of high dimensional datasets, due to the complexity of the algorithm. Parallel implementations of k-means reduces the computational cost and the execution time. A simple solution to this problem is parallel execution of the algorithm iterations (i) until convergence [5], [13].

At this study, parallel versions of Random Centroids and Random Assignments (`k-meansRCP` and `k-meansRAP` accordingly) are introduced using the principles of parallel programming in Matlab [6], [10] with appropriate loops, variables, statements, indexing, matrices etc.

Both of these algorithms use the "$parfor$" Matlab command to perform iterations at the available cores of the computer system, speeding up the calculations in the loops. The Matlab code of `k-meansRCP` and `k-meansRAP`, with the appropriate commands and functions for parallel random assignment of k centroids and parallel random assignment of the data to k classes, is presented at Algorithm 1. The difference between the code of `k-meansRAP` and `k-meansRCP` is focused at the call of function `selectionInitialCendroid`.

2.3 Datasets

An algorithm for generation of multidimensional data that form simple geometric shapes was developed for the needs of this study. The geometrical data sets that generated by this algorithm were also used for educational purposes and mainly for understanding the performance of data mining techniques. The algorithm has the ability to produce geometric shapes in multidimensional space. We define the dimensions of the space for which the data set will be generated, these dimensions refer to the number of rows (templates) and the number of columns (features), the radius of the clusters and the coordinates of the clusters in a multidimensional space. These parameters are defined by $mat3$ matrix of Algorithm 2 where n are the samples of each cluster, R is the radius of the hyper-shapes and x, y, z are the coordinates of a 3D cluster centre.

The algorithm can be configured to produce data (patterns) that will form dense or sparse geometric shapes, such as hyper-cubes or hyper-spheres in many dimensions. These hyper-shapes are generated around a center with predefined hyper-sides or hyper-radius. The algorithm determines the distance between the geometric hyper-shapes as well as the number of points that will be contained in each hyper-shape. The above algorithm is presented at Algorithm 2. Figure 1 presents two data sets generated by `DrawDataGen` where a) the dataset of $n = 20.000$ samples is divided to seven clusters drawing 2D squares with different densities of samples and b) the number of samples

Algorithm 1 Algorithm of parallel implementation of k-means in Matlab

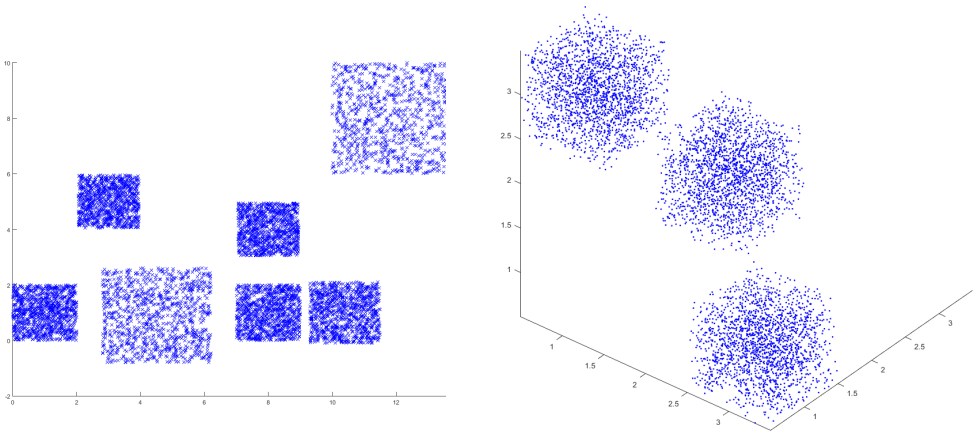
```
function [assignment, fCentroid, minDist] = KmeansP(data, nClusters, tolerance,status)
limitWhile=50;
[m, n] = size(data);
parfor iter=1:tolerance
    centroid=selectionInitialCendroid(status,nClusters,n,m,data)
    flag = 1.0;
    count=0;
    while flag ~= 0 && count <= limitWhile
        count=count+1;
        diffT = zeros(m,nClusters);
        for c = 1 : nClusters
            diffT(:,c) = (data(:,1)-centroid(c,1)).^2;
            for d = 2:n
                diffT(:,c) = diffT(:,c) + (data(:,d)-centroid(c,d)).^2;
            end
        end
        [dist, idx] = min(diffT, [], 2);
        distAll{iter} = dist;
        assignment=idx';
        oldCentroid = centroid;
        plithos = accumarray(idx,1,[nClusters,1]);
        for i = 1:n
            centroid(:, i) = accumarray(idx,data(:, i),[nClusters,1]) ./ plithos;
        end
        indZero = find(plithos == 0);
        if ~isempty(indZero)
            for i=1:n
                for j=indZero
                    centroid(j,i) = data(randi(m), n);
                end
            end
        end
        flag = sum (sum( abs(centroid - oldCentroid) ) );
    end
    totalAssignment{iter}=assignment;
    totalCentroid{iter}=centroid;
end
parfor i=1:tolerance
    sumDist(i) = sum(distAll{i});
end
[minDist, id] = min(sumDist);
assignment= totalAssignment{id}';
fCentroid = totalCentroid{id};
end
```

```
function centroid=selectionInitialCendroid(status,nClusters,n,m,data)
% status=1 RCP
% status=2 RAP
% status other ones
centroid = ones(nClusters, n);
if (status==1)
    for j=1:nClusters
        centroid(j,:) = data(randi(m), :);
    end
elseif(status==2)
    assignment = randi(nClusters, [m, 1]);
    plithos = accumarray(assignment,1,[nClusters,1]);
    for i = 1:n
        centroid(:, i) = accumarray(assignment,data(:, i),[nClusters,1]) ./ plithos;
    end
end
```

($n = 6.000$) is divided into three clusters of equal density 3D cubes. The `mat3` matrix parameters of Algorithm 2 refer to a dataset of 3 cubic clusters in Figure 1(b).

The hardware that used for the evaluation of the k-means functions, was a blade computing system with Intel Xeon E5640 64x 2.67GHz (16 cores)CPU and 16GB RAM. The execution time of k-means over different datasets, was calculated according the following equation

$$Time = \frac{t_1 + t_2 + t_3 + \ldots + t_T - t_{Max} - t_{Min}}{10}$$



(a) (b)

Figure 1: Datasets generated by `DrawDataGen`.(a) A 2D synthetic data set divided into 7 rectangular clusters. (b) A 3D synthetic data set divided into 3 cubic clusters

Algorithm 2 Algorithm of `DrawDataGen` in Matlab

```
function out=DrawDataGen(mat)
[row, col] = size(mat);
dim = col-2;
out = [];
for i=1:row
    n=mat(i, 1);
    R=mat(i, 2);
    xc=zeros(1, dim);
    for j=1:dim
        xc(j)=mat(i,j+2);
    end
    xr=[];
    for k=1:n
        temp = ((rand([1, dim]) * 2 * R ) - R) + xc;
        xr = [xr; temp];
    end
    out=[out;xr];
end
```

```
%       n     R    x  y  z
mat3 = [2000, 0.5, 3, 1, 1
        2000, 0.5, 1, 3, 1
        2000, 0.5, 1, 1, 3]
```

where $t_i(i = 1, ..T)$ is the execution time of an iteration of k-means on a dataset.

3 Results

For the evaluation of the parallel k-means algorithms against the serial k-means, 30 datasets of different length (n), feature dimension (d) and clus-

ters (k) were generated with Algorithm 2. The notation for the datasets is $D_k^d(n)$ where $n = 20000, 100000$, $d = 1, \ldots, 5$ and $k = 2, 4, 5$. For example $D_2^4(100000)$ represents the dataset with $n = 100000$ samples, $k = 2$ classes and $d = 4$ dimensions. All k-means algorithms executed for $i = 100$ iterations for convergence, using as metric the euclidean distance. Table1 presents the performance of the algorithms in the datasets and the execution time, which is the average value of the 5 executions for each dataset.

It is observed that `k-meansRAP` is, in average, 4.0 times, faster than the serial equivalent `k-meansRC`, 3.7 times faster than `k-meansRA`, 1.1 times faster than `k-meansRCP` and 5.7 times faster than `Mk-means`. The maximum difference in execution time is observed at $D_4^1(20000)$ dataset between `k-meansRAP` and `Mk-means` where `k-meansRAP` was 9.6 times faster. Is also observed that the performance of `k-meansRCP` and `k-meansRAP` is equivalent but the second one seems a bit faster.

Table 1: Execution time(sec) of `k-meansRC`, `k-meansRA`, `k-meansRCP`, `k-meansRAP` and `Mk-means` application on $D_k^d(n)$ datasets

$D_k^d(n)$	RC	RA	RCP	RAP	M
$D_2^1(20000)$	1.192	1.394	0.422	0.397	2.116
$D_2^2(20000)$	1.911	1.633	0.508	0.436	2.680
$D_2^3(20000)$	2.681	1.713	0.661	0.482	3.162
$D_2^4(20000)$	2.301	1.917	0.614	0.527	2.541
$D_2^5(20000)$	3.828	2.745	0.915	0.701	3.770
$D_4^1(20000)$	3.761	3.376	0.884	0.756	7.275
$D_4^2(20000)$	5.053	4.706	1.155	1.022	8.138
$D_4^3(20000)$	4.289	4.616	0.986	1.017	5.401
$D_4^4(20000)$	4.946	5.186	1.113	1.103	5.906
$D_4^5(20000)$	6.142	6.353	1.373	1.373	6.583
$D_5^1(20000)$	5.019	4.829	1.165	1.091	10.430
$D_5^2(20000)$	7.828	7.764	1.729	1.621	12.405
$D_5^3(20000)$	8.603	8.810	1.890	1.845	11.352
$D_5^4(20000)$	7.243	7.298	1.558	1.559	8.492
$D_5^5(20000)$	10.361	10.292	2.194	2.206	11.222
$D_2^1(100000)$	5.616	5.888	2.104	2.049	10.813
$D_2^2(100000)$	7.790	7.042	2.654	2.393	12.464
$D_2^3(100000)$	7.391	6.030	2.563	2.132	10.552
$D_2^4(100000)$	11.301	9.201	3.841	2.980	13.461
$D_2^5(100000)$	11.979	10.215	3.744	3.233	14.053
$D_4^1(100000)$	23.007	22.898	7.889	7.883	47.417
$D_4^2(100000)$	33.424	28.770	10.247	8.936	53.660
$D_4^3(100000)$	23.723	24.344	7.399	7.445	32.541
$D_4^4(100000)$	31.413	31.817	9.215	9.399	38.531
$D_4^5(100000)$	34.885	34.917	10.066	9.870	38.850
$D_5^1(100000)$	35.838	27.987	12.469	9.626	73.579
$D_5^2(100000)$	42.452	39.701	13.603	12.719	72.945
$D_5^3(100000)$	35.372	34.708	11.076	10.580	51.210
$D_5^4(100000)$	35.492	36.548	10.894	10.898	46.114
$D_5^5(100000)$	45.632	46.891	13.552	13.795	56.031

4 Conclusions

We have introduced two parallel implementations of k-means in Matlab in order to take advantage of computer CPU cores and speed up computations in many clustering problems. A numerical data generator has also presented, generating multidimensional clusters of geometric hyper-shapes. The `k-meansRAP` seems a better performance than `k-meansRCP` and the execution time in parallel versions of k-means is inversely proportional to the number of CPU cores. In a future extension of this work, the execution time of k-means implementations in clusters of CPUs with datasets of extreme dimensional space will be examined compared to other parallel implementations of k-means.

Acknowledgements. The authors wish to acknowledge financial support provided by the Research Committee of the International Hellenic University (former Technological Education Institute of Central Macedonia), under grant SAT/IC/07022018-30/13.

References

[1] D. Arthur, S. Vassilvitskii, How Slow is the K-means Method?, *Proceedings of the Twenty-second Annual Symposium on Computational Geometry, Sedona, Arizona, USA,* (2005), 144–153. https://doi.org/10.1145/1137856.1137880

[2] G. Forman, B. Zhang, Distributed data clustering can be efficient and exact, *AACM SIGKDD Explorations Newsletter,* **2** (2) (2000), 34–38. https://doi.org/10.1145/380995.381010

[3] I. Guyon, A. Elisseeff, An Introduction to Variable and Feature Selection, *J. Mach. Learn. Res.,* **3** (2003), 1157–1182.

[4] X. Li, Z. Fang, Parallel Clustering Algorithms, *Parallel Computing,* **11** (1989), 275–290. https://doi.org/10.1016/0167-8191(89)90036-7

[5] C. Lin, L. Snyder, *Principles of Parallel Programming*, Addison-Wesley, 2008.

[6] P. Luszczek, Parallel Programming in MATLAB, *International Journal of High Performance Computing Applications*, **23** (3) (2009), 277–283. https://doi.org/10.1177/1094342009106194

[7] C. Moler, *Parallel MATLAB: Multiple processors and multiple cores*, The MathWorks News & Notes, 2007.

[8] C.F. Olson, Parallel Algorithms for Hierarchical Clustering, *Parallel Computing,* **21** (8) (1995), 1313–1325. https://doi.org/10.1016/0167-8191(95)00017-i

[9] E.M. Rasmussen, P. Willett, Efficiency of Hierarchical Agglomerative Clustering Using the ICL Distributed Array Processor, *Journal of Documentation,* **45** (1) (1989), 1–24. https://doi.org/10.1108/eb026836

[10] G. Sharma, J. Martin, MATLAB : A Language for Parallel Computing, *International Journal of Parallel Programming*, **37** (1) (2009), 3–36. https://doi.org/10.1007/s10766-008-0082-5

[11] P. Sneath, R. R. Sokal, *Numerical Taxonomy: The Principles and Practice of Numerical Classification*, San Francisco: W.H. Freeman, 1973.

[12] A. Tsimpiris, D. Kugiumtzis, Feature selection for classification of oscillating time series, *Expert Systems,* **29** (5) (2012), 456–477. https://doi.org/10.1111/j.1468-0394.2011.00605.x

[13] D. Varsamis, P. Mastorocostas, A. Papakonstantinou, N. Karampetakis, A parallel searching algorithm for the insetting procedure in Matlab Parallel Toolbox, *Federated Conference on Computer Science and Information Systems (FedCSIS),* (2012), 587–593.

[14] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, Top 10 algorithms in data mining, *Knowledge and Information Systems,* **14** (1) (2008), 1–37. https://doi.org/10.1007/s10115-007-0114-2

Received: November 12, 2020; Published: November 30, 2020