Proceedings of the 17th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2017
4–8 July, 2017.

Yinyang K-means clustering for hyperspectral image analysis

Mercedes Eugenia Paoletti[1], Juan Mario Haut[1], Javier Plaza[1] and Antonio Plaza[1]

[1] *Department of Technology of Computers and Communications, University of Extremadura, Escuela Politecnica, Avda. de la Universidad s/n*

emails: mpaolett@alumnos.unex.es, juanmariohaut@unex.es, jplaza@unex.es, aplaza@unex.es

Abstract

Hyperspectral images are widely used in remote sensing applications due to their wealth of information in the spectral domain, that allows for very detailed scene classification. Clustering is one of the most used unsupervised techniques for the analysis of these scenes. Popular clustering techniques such as K-means are computationally expensive, particularly when applied to hyperspectral images characterized by their large dimensionality. An efficient implementation of K-means is the so-called *Yinyang K-means*, which outperforms K-means algorithms by clustering the centers in the initial stage, and leveraging efficiently maintained lower and upper bounds between each point and the cluster centers. In this work, we have adapted an efficient implementation of this algorithm using graphics processing units (GPUs) for hyperspectral image analysis. We have carried out a comparison of this technique with other existing implementations with the aim of demonstrating its usefulness in hyperspectral imaging. Our obtained results suggest that this technique is ideal for working with big hyperspectral data repositories.

Key words: Hyperspectral imaging, k-means clustering, YinYang K-means, GPUs.

1 INTRODUCTION

Current Earth observation (EO) sensors acquire and produce high-dimensional data cubes with hundreds of spectral channels and millions of pixels. For instance, NASA's Jet Propulsion Laboratory's Airbone Visible/Infrared Imaging Spectrometer (AVIRIS) [1] measures the solar reflected spectrum from 400nm to 2500nm at intervals of 10nm. The

 ISBN: 978-84-617-8694-7

EO-1 Hyperion imaging spectrometer collects bands in the range of 400nm to 2500nm too [2, 3]. The resulting hyperspectral datasets [4] provide information corresponding to large observation areas on the surface of the Earth, using hundreds of contiguous spectral bands. As a result, these instruments can produce three-dimensional data cubes with size significantly larger than traditional images. These images can be exploited in many practical applications, such as monitoring and management of the environment and agriculture, urban and regional planning, detection of relevant geological zones (e.g. mineral detection) or defense and intelligence issues (e.g. target detection or mine detection).

However, hyperspectral images present many challenges in terms of storage and processing due to their large dimensionality. In addition, modern sensors are producing an almost continuous stream of data [3]. For example, AVIRIS has a data collection rate of 2.5 MB/s and Hyperion collects almost 71.9 GB/hour. On the other hand, most of the satellite missions that will be soon in operation, such as the environmental mapping and analysis program (EnMAP http://www.enmap.org/) present similar data collection ratios. This creates the need for scalable and efficient processing techniques for hyperspectral data in the context of different applications [3].

Many techniques (supervised and unsupervised) have been developed to address the aforementioned challenges. One of the most widely used unsupervised methods is clustering, which aims to organize the data so that pixels with similar spectral content are clustered together in the same class [5]. In this case, there is no need for labeled samples which are common in supervised techniques [6]. Although clustering offers an unsupervised alternative that has been widely used in various fields, it is also a very challenging task due to the large spectral variability and complex spatial structures present in hyperspectral images. The most popular and widely used family of clustering algorithms is represented by centroid-based clustering methods such as K-means [7].

K-means assumes that similar pixels always form clusters in feature space. By applying this method to hyperspectral images we can obtain satisfactory results, but K-means is hampered by its computational complexity. There are a handful of studies which aim to address this issue, either adapting the algorithm to parallel processing structures such as field-programmable gate array (FPGAs) [5, 8]or cloud computing architectures [9]. Other works aim at developing improved implementations such as K-means++ [10, 11], the $AFKMC^2$ [12], the K-means projective clustering [13] or the filtered K-means [7]. The *Yinyang* K-means [14] is a recent improvement of K-means. This method features a space-conscious elastic design that adaptively uses the upper and lower bound based filters while maintaining various space constraints. The upper and lower bound based filters are continuously and carefully maintained, and provide an efficient evolvement and interplay mechanism.

Our main goal in this paper is to adapt the *Yinyang* K-means to hyperspectral image processing. The implementation that we have adopted is optimized for GPUs using

 ISBN: 978-84-617-8694-7

NVIDIA Compute Device Unified Architecture (CUDA). To test the effectiveness of the implementation, we have compared it with other existing k-means implementations and made an exhaustive analysis of the pros and cons of all the implementations used.

The remainder of the paper is organized as follows. Section 2 will delve into the K-means method and the improvements provided by the *Yinyang* K-means version, presenting its theoretical foundations. Section 3 validates the *Yinyang* K-means algorithm by comparing it with other implementations in terms of execution times. Finally, section 4 concludes with some remarks and hints at plausible future research lines.

2 K-means method: an overview

2.1 The K-means clustering algorithm

K-means is one of the easiest unsupervised learning algorithms and most widely used method to group data in a specified number of clusters. Suppose a set of n observations $X = (x_1, x_2, ..., x_n)$, where each observation is $x_i \in \mathbb{R}^d$, i.e. $x_i = [x_{i_1}, x_{i_2}, ..., x_{i_d}]$ (where d is the number of spectral channels). The goal is to group each observation into a number of *clusters* k fixed a priori ($k <= n$). Iteratively, K-means calculates the centers of the k groups, optimizing the error of each group as $\min \sum_{j=1}^{k} \sum_{i=1}^{n_k} \parallel x_i^j - c_j, \parallel^2$ where $\parallel x_i^j - C_j \parallel^2$ is the euclidean distance between a data point x_i^j of the cluster j (n_k is the observations within each cluster) and the cluster center c_j, which if it is the point that minimizes the equation also called *centroid of cluster j*.

K-means algorithm successfully performs the task of obtaining useful information from the dataset, such as the best distance metric for the data [15]. However, the results can vary greatly due to a small change in parameters and in the choice of the initial centers. So, a proper initialization will result in a final best solution. In order to obtain a set of good initial cluster centers, several methods have been proposed, as K-means++ [10, 11]. This algorithm obtains a set of k initial centers which are generally very close to the final solution.

2.2 *Yinyang* K-means method

The main problem with the traditional K-means implementation is that it performs a lot of redundant work when recalculating distances corresponding to samples which are not going to change the cluster. With this in mind, *Yinyang* K-means optimizes two important points in the K-means algorithm: the assignment steps and the update steps. In order to do that, it implements two filters and a new center update method.

In the standard K-means, the assignment step computes the distances between every point x_i and every cluster center c_j in order to find out the closest center to each point.

 ISBN: 978-84-617-8694-7

The *Yinyang* K-means instead uses two filters to detect which distance calculations are unnecessary and avoids computing them. These filters are based on the triangle inequality:

- Group filtering groups the k clusters into t groups $G = g_1, g_2, ..., g_t$, where each $g_i \in G$ is a set of the clusters and t must no be greater than $k/10$. t provides a design knob for controlling the space overhead and redundant distance elimination. For each group, it calculates:

  1. Upper bound: for each point x in cluster j ($j = j(x)$) it sets the upper bound to $uj(x) = d(x, j(x))$, i.e. the distance between x and cluster. The upper bound is updated as $uj'(x) = uj(x) + \delta(j)$, where $\delta(j)$ is the distance of the cluster j.
  2. Lower bound: for each point x in cluster j ($j = j(x)$) it sets the lower bound $lj(x, g_i)$ as the shortest distance between x and all centers in g_i excluding $j(x)$. The lower bound is updated as $lj'(x, g_i) = lj(x, g_i) - max_{c \in g_i}\delta(c)$, $\delta(c) = d(c, c')$ is the shift of cluster center due to the center update.

  If the updated lower bound is major than the updated upper bound, $lj'(x, g_i) > uj'(x)$, no reassignment is needed for point x and all the group-level comparisons can be avoided.

- Local filtering is used for get the new centroid of a cluster, avoiding unnecessary distance operations. A new center $c' \in g'_i$ cannot be the closest center to a point x if there is another center $p' \neq c'$ such that $d(x, p') < lj(x, g_i) - \delta(c)$

On the other hand, in the updating step, K-means computes the new center for each cluster. In *Yinyang* K-means new centers are computed as $c' = \frac{c \cdot |V| - \left(\sum_{y \in V - OV} y\right) + \sum_{y' \in V' - OV} y'}{|V'|}$ where V' and V denote a cluster in the current and previous iteration, OV is $V \cap V'$, c is the old center and c' is the new center.

2.3 Parallel GPU *Yinyang* K-means

The parallel implementation of *Yinyang* K-means that we have adopted is available in a library. It has been optimized for low memory consumption and use of a large number of clusters, bearing in mind the limitations of the K-means algorithm related to the calculation of the centroids that requires having all the data be available in the same place. In the context of the CUDA-based GPU implementation, if the data occupies a large amount of storage, it is impossible to store them in the memory of a single GPU, so it is necessary to cut the samples into as many intervals as GPUs are available. As a result, in our adopted implementation each GPU will work with its own interval, calculating the distances and the local centroids, writing the local assignments and broadcasting its results to other GPUs.

The parameters used by our adopted implementation are the number of clusters, k, the number of cluster groups (t), and the value for *tolerance*, which will be used by the algorithm

 ISBN: 978-84-617-8694-7

to stop its execution (if the number of reassignments drop below the tolerance value). The parallel *Yinyang* K-means execution is initialized with random centroids (or it can also be initialized by intelligently produced centroids such as those produced by K-means++) and calculates the Euclidean distance (L2) to perform the centroid selection calculations as $d_2(\overrightarrow{x}, \overrightarrow{y}) = \sqrt[2]{\sum_i (x_i - y_i)^2}$.

The output of the method is a vector of centroids and a vector with the cluster index for each sample (numerical labels for each pixel).

3 Experiments and results

3.1 Experimental Configuration

In order to evaluate the performance of the adopted *Yinyang* K-means implementation, we use a hardware environment composed by a 6th Generation Intel® Core™i7-6700K processor with 8M of Cache and up to 4.20GHz (4 cores/8 way multitask processing), 32GB of DDR4 RAM with a serial speed of 2400MHz, a GPU NVIDIA GeForce GTX 1080 with 8GB GDDR5X of video memory and 10Gbps of memory frequency, a Toshiba DT01ACA HDD with 7200RPM and 2TB of capacity, and an ASUS Z170 pro-gaming motherboard. On the other hand, the software environment is composed by Ubuntu 16.04.4 x64 as operating system, CUDA 8 and Python.

3.2 Hyperspectral data sets

In our experiments, we use four different hyperspectral images. The first one was collected by AVIRIS [1] in 1992 over a set of agricultural fields with regular geometry and with a multiple crops and irregular patches of forest in Northwestern Indiana. This scene, Indian Pines, has 145x145 pixels with 224 spectral bands in the range from 400 to 2500nm, with 10nm of spectral resolution, 20nm moderate spatial resolution and 16 bits radiometric resolution. 4 zero bands plus 20 bands with lower signal-to-noise ratio (SNR) have been removed, retaining 200 spectral channels. Dataset has 16 ground-truth classes(Fig. 1).

Also, we use a larger version of the Indian Pines scene. This one has a much larger size of 2678×614 pixels. It was collected over the same area that small Indian Pines, but spanning a much larger extent. It contains 220 spectral bands in the range from 400 to 2500 nm, with spectral resolution of 10 nm, moderate spatial resolution of 20 nm and 16 bits of radiometric resolution. The total number of classes is 58. (Fig. 4).

The third dataset was collected by AVIRIS over Salinas Valley, California (Fig. 3). The area covered has 512×217 samples and the spatial resolution is 3.7 m per pixel. 204 out of the 224 bands are kept after 20 water absorption bands are removed. Dataset has 16 land-cover classes.

 ISBN: 978-84-617-8694-7

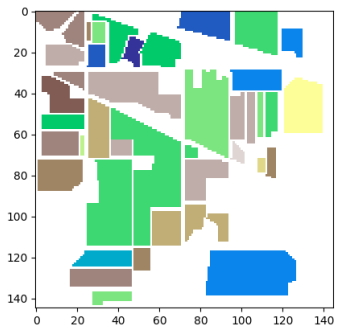

Figure 1: Ground-truth of small Indian Pines scene.

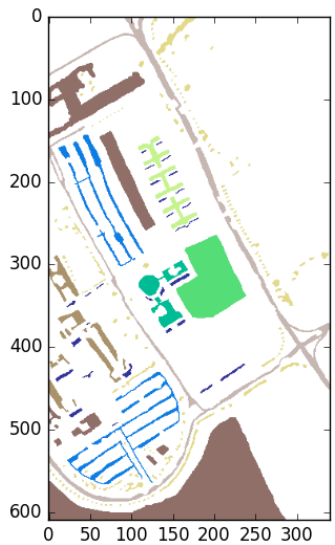

Figure 2: Ground-truth of University of Pavia.

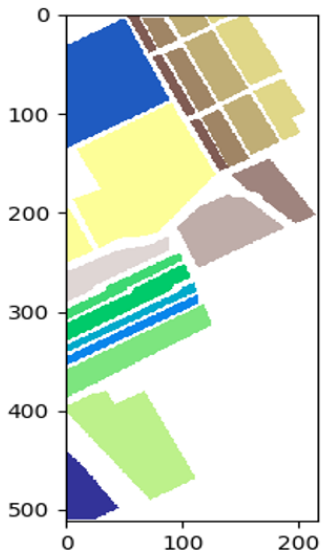

Figure 3: Ground-truth of Salinas scene.

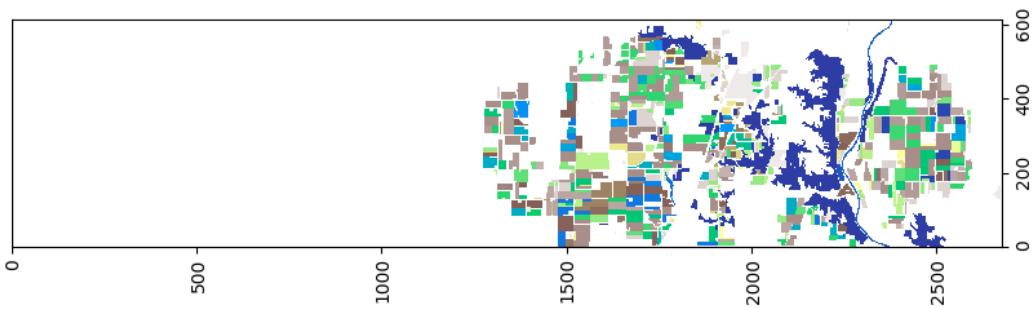

Figure 4: Ground-truth of big Indian Pines scene.

The fourth hyperspectral dataset was collected by the Reflective Optics System Imaging Spectrometer (ROSIS) sensor [16] during a flight campaign over Pavia, northern Italy. The dataset covers an urban environment, with various solid structures, natural objects and shadows (9 classes in total). The scene (see Fig. 2) contains 103 spectral bands of 610×340 pixels in the spectral range from 0.43 to 0.86μm, with spatial resolution of 1.3m/pixel.

3.3 Performance evaluation

In order to evaluate the performance of parallel *Yinyang* K-means, several experiments have been executed. The first one is a comparison between the GPU version of *Yinyang*

 ISBN: 978-84-617-8694-7

K-means and other iterative and parallel GPU implementations of the original (Lloyd) K-means algorithm setting the maximum number of centroids to the number of classes. The tolerance value was set to 0.001. To complete the experiment, we have tested two initiations of centroids: 1) completely random and 2) K-means++ method. Each version has been executed 10 times and the average times are reported for statistical significance. The obtained results are reported in Table 1.

Initialization	**Lloyd Iterative**		**Lloyd CUDA**		**Yinyan CUDA**	
	Small Indian Pines					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	0.183 (0.015)	1	0.421 (0.062)	0.435	0.441 (0.066)	0.415
random	0.185 (0.007)	1	0.520 (0.050)	0.355	0.511 (0.056)	0.361
	Pavia University					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	0.238 (0.017)	1	0.839 (0.189)	0.283	0.837 (0.242)	0.284
random	0.223 (0.07)	1	0.998 (0.124)	0.223	1.088 (0.159)	0.205
	Salinas					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	0.726 (0.078)	1	2.436 (0.616)	0.298	2.083 (0.567)	0.348
random	0.715 (0.5)	1	1.978 (0.331)	0.361	2.231 (0.384)	0.320
	Big Indian Pines					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	38.584 (2.694)	1	46.167 (10.158)	0.836	40.085 (10.753)	0.963
random	37.217 (2.960)	1	69.518 (6.057)	0.535	56.321 (11.650)	0.661

Table 1: Average time executions (standard deviation) and speed-up for each implementation of K-means initialized with random centroids and K-means++.

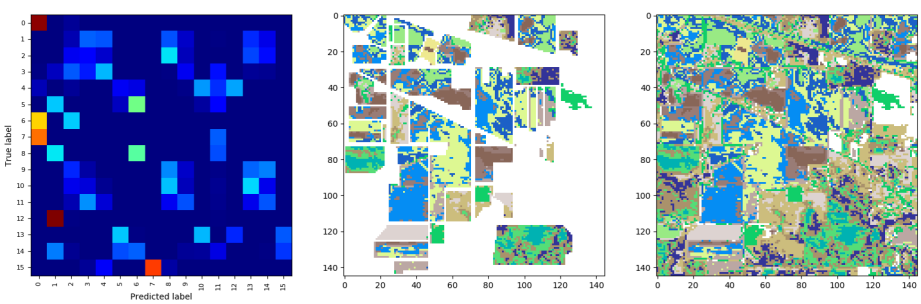

Figure 5: Small Indian Pines *Yinyang* K-means classification results: the confusion matrix and the classification maps without background and with background.

For small Indian Pines image, the fastest K-mean implementation is the original Lloyd

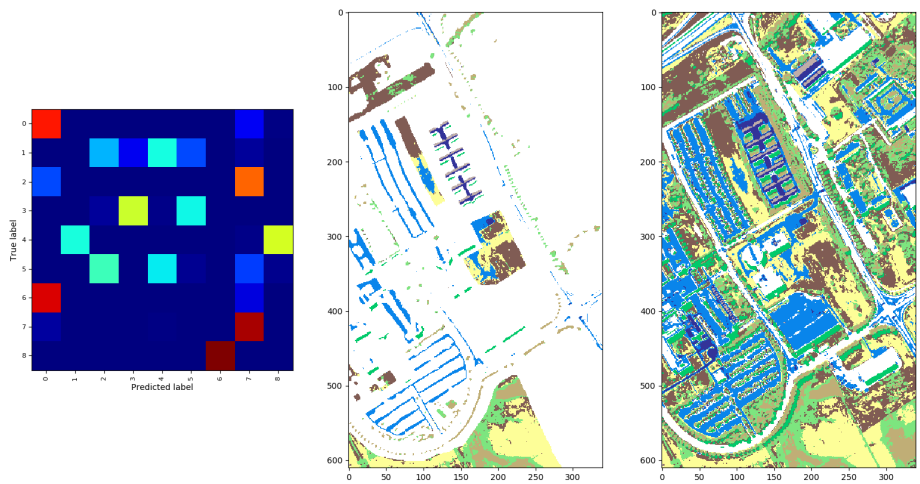

Figure 6: Pavia University *Yinyang* K-means classification results: the confusion matrix and the classification maps without background and with background.

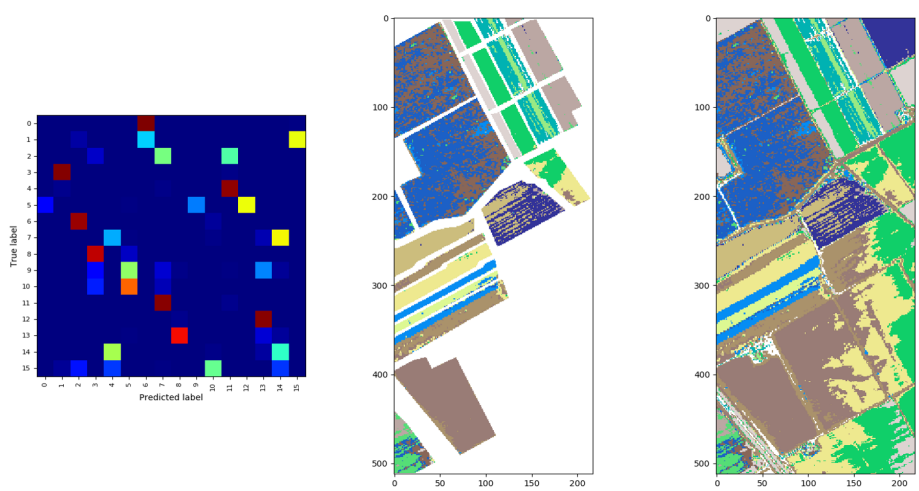

Figure 7: Salinas *Yinyang* K-means classification results: the confusion matrix and the classification maps without background and with background.

iterative version with K-means++ initialization. It is 2.41 times faster than the *Yinyang* cuda implementation with K-means++ initialization and 2.79 times faster than the same algorithm with random initialization. This is due to the small size of the image (only 145×145 pixels in only 16 groups), which is not enough to get the most out of GPU versions. In Fig. 5 we can see the classification results of *Yinyang* K-means. The confusion

matrix is a typical mechanism to evaluate unsupervised clustering methods, where in our representation warm colors indicate a high value and warm colors indicate a low value. The two classification maps show the clustering result without and with background pixels. As we can observe, the obtained result is reasonable for a K-means method, despite some noise at the borders of the classes.

For Pavia University dataset, the fastest implementation is also the iterative version and, among the two parallel GPU versions, the original LLoyd algorithm is faster than *Yinyang*. In this case we have more data than Indian Pines, but still enough complexity (only 9 centroids). In Fig. 6 we can observe the confusion matrix of the classification with *Yinyang*, whose results are better than in the Small Indian Pines. The classification maps reveal less noise at the borders.

For Salinas we have similar results: although we have a lot of data, the complexity is not enough (only 16 centroids). So, the iterative version is still the fastest. But we can see that *Yinyang* with K-means++ initialization is better than the original Lloyd algorithm parallelized in GPU: the differences between GPU versions are already starting to appear. In Fig. 7 we can see that the classification results with *Yinyang* are better than in the small Indian Pines and Pavia University images. Specifically, border pixels are better identified.

Finally, for big Indian Pines the fastest implementation is the iterative Lloyd algorithm (2678×614 with 58 centroids), but if we compare the two GPU versions, the *YinYang* is faster than the parallel GPU Lloyd. Since the classification results are similar to the ones already reported for the small Indian Pines image, we do not include these results for space considerations.

In summary, our results indicate that *YinYang* K-means works better than the CUDA version of Lloyd algorithm when more data needs to be processed, but higher complexity appears to be needed in order to improve the iterative version. So, we repeated the first experiment increasing the number of centroids to be calculated in a second experiment, which compares the parallel GPU implementation of *Yinyang* K-means with the same implementations of the original K-means algorithm, using a maximum number of centroids set to one hundred times the number of classes in each scene (i.e. 1600, 900, 1600 and 5800 centroids, respectively). Again, this is intended to increase the analysis complexity. The tolerance value is 0.001 in all cases. Again,we tested with random and K-means++ [10, 11] initializations. The obtained results are reported in Table 2.

For small Indian Pines dataset the fastest implementation is the *Yinyang* K-means with random initialization. It reaches a speed up of 7.071 over the iterative version. In the first experiment, with 16 classes the execution times were 0.441 and 0.511, at this time with 1600 centroids to calculate the execution times of *Yinyang* increase in just one second. However for iterative version, it needed 0.183-0.185 seconds and now it needs 6 seconds more. Also, for Pavia University scene *Yinyang* K-means is the fastest implementation, with K-means++ initialization. With the same number of pixels and 900 centroids, *Yinyang*

 ISBN: 978-84-617-8694-7

Initialization	Iterative		CUDA		Yinyan CUDA	
	Small Indian Pines					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	6.034 (0.310)	1	1.912 (0.134)	3.191	1.542 (0.024)	3.956
random	6.687 (0.245)	1	1.131 (0.050)	5.410	0.865 (0.025)	7.071
	Pavia University					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	14.124 (0.845)	1	4.166 (0.221)	3.350	2.884 (0.266)	4.839
random	12.941 (0.780)	1	3.526 (0.278)	4.119	2.573 (0.209)	5.644
	Salinas					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	40.308 (1.938)	1	13.230 (0.767)	3.077	5.879 (0.196)	6.926
random	37.507 (2.460)	1	13.630 (1.229)	3.155	5.602 (0.146)	7.678
	Big Indian Pines					
	Time	*Speed up*	*Time*	*Speed up*	*Time*	*Speed up*
k-means++	2331.929 (233.417)	1	461.223 (11.586)	5.022	121.726 (1.909)	19.029
random	2022.694 (260.634)	1	463.165 (16.392)	5.132	113.459 (3.653)	20.952

Table 2: Average time executions (standard deviation) and speed-up for each implementation of K-means initialized with random centroids and K-means++.

K-means needs only two more seconds. The iterative version needs 12 or 14 seconds more, from the first experiment where it needed 0.23-0.22 seconds. For Salinas dataset we observe the same behavior: with 1600 centroids to calculate, the fastest one is *Yinyang* in GPU reaching a speedup of 7.678. For the big Indian Pines image, *Yinyang* K-means reaches a significant speedup: a 20.95. These results show that, the more the complexity of the analysis, the better the performance of the *Yinyang* GPU which is intended for big data problems involving not only massive data repositories but also complex analysis scenarios.

4 Conclusions and Future Lines

In this paper, we have proved a recent variant of K-means, the *Yinyang* K-means algorithm, to hyperspectral image analysis, in particular a parallel GPU implementation, which has been shown to obtain good processing results in hyperspectral image analysis when compared with other popular K-means implementations. Specifically, our experimental results show the effectiveness of the parallel GPU implementation of *Yinyang* K-means using four different hyperspectral scenes. The algorithm performs particularly effectively when we need to process big data and calculate a large set of centroids. The method not only improves as more data become available, but also with the increase of the complexity of the clusterization. On the other hand, the ranking results are in line with those obtained by any K-means algorithm. As future work, we are planning on using the *Yinyang* K-means in

 ISBN: 978-84-617-8694-7

conjunction with other techniques for hyperspectral image classification (e.g. supervised and semi-supervised techniques) with the aim of improving the obtained classification results.

Acknowledgements

This work has been supported by Ministerio de Educación (Resolución de 26 de diciembre de 2014 y de 19 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan ayudas para la formación de profesorado universitario, de los subprogramas de Formación y de Movilidad incluidos en el Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016). This work has also been supported by Junta de Extremadura (decreto 297/2014, ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR15005).

References

[1] Robert O. Green, Michael L. Eastwood, Charles M. Sarture, Thomas G. Chrien, Mikael Aronsson, Bruce J. Chippendale, Jessica A. Faust, Betina E. Pavri, Christopher J. Chovit, Manuel Solis, Martin R. Olah, and Orlesa Williams. Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). *Remote Sensing of Environment*, 65(3):227–248, 1998.

[2] Amin Beiranvand Pour and Mazlan Hashim. ASTER, ALI and Hyperion sensors data for lithological mapping and ore minerals exploration. *SpringerPlus*, 3(1):130, 2014.

[3] A. Plaza, J. Plaza, A. Paz, and S. Sanchez. Parallel Hyperspectral Image and Signal Processing. *IEEE Signal Processing Magazine*, 28(3):119–126, 2011.

[4] Chein-I Chang. *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. Springer US, 2003.

[5] Miriam Leeser, Pavle Belanovic, Michael Estlick, Maya Gokhale, John J Szymanski, and James Theiler. Applying Reconngurable Hardware to the Analysis of Multispectral and Hyperspectral Imagery. In International Society for Optics and Photonics, editor, *International Symposium on Optical Science and Technology*, pages 100–107, 2002.

[6] Antonio Plaza, Javier Plaza, Gabriel Martín, and Sergio Sánchez. Hyperspectral Data Processing Algorithms. In Alfredo Huete Prasad S. Thenkabail, John G. Lyon, editor, *Hyperspectral Remote Sensing of Vegetation*, chapter 5, pages 121–137. Taylor & Francis, 2011.

[7] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[8] Abel Guilhermino, Da S Filho, Alejandro C Frery, Cristiano Coêlho De Araújo, Haglay Alice, Jorge Cerqueira, Juliana A Loureiro, Manoel Eusebio De Lima, Maria Das, Graas S Oliveira, and Michelle Matos Horta. Hyperspectral Images Clustering on Reconfigurable Hardware using the K-Means Algorithm. In *16th Annual Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 8–11, Sao Paulo (Brasil), 2003.

[9] J.M. Haut, M. Paoletti, J. Plaza, and A. Plaza. Cloud implementation of the K-means algorithm for hyperspectral image analysis. *Journal of Supercomputing*, 73(1), 2017.

[10] David Arthur and Sergei Vassilvitskii. k-means++: The Advantages of Careful Seeding. In ACM, editor, *Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[11] Olivier Bachem, Mario Lucic, S Hamed Hassani, and Andreas Krause. Approximate K-Means++ in Sublinear Time. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1459–1467, Phoenix, Arizona, 2016. AAAI Press.

[12] Olivier Bachem, Mario Lucic, S Hamed Hassani, and Andreas Krause. Fast and Provably Good Seedings for k-Means, 2016.

[13] K. Agarwal, Pankaj and Nabil H. Mustafa. K-Means Projective Clustering. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 155–165, Paris- France, 2004. ACM.

[14] Yufei Ding, Yue Zhao, Ncsuedu Xipeng Shen, Madanlal Musuvathi, and Microsoftcom Todd Mytkowicz. Yinyang K-Means: A Drop-In Replacement of the Classic K-Means with Consistent Speedup. In *Proceedings of the 32nd International Conference on Machine Learning*, page 579587, Lille, France, 2015. JMLR: W&CP.

[15] Justin Sunu. *Applications of K-means and Spectral Clustering to Hyperspectral Video Sequences*. PhD thesis, California State University, Long Beach, 2014.

[16] B. Kunkel, F. Blechinger, R. Lutz, R. Doerffer, and H. van der Piepen. ROSIS (Reflective Optics System Imaging Spectrometer) - A candidate instrument for polar platform missions. In J. Seeley and S. Bowyer, editors, *Optoelectronic technologies for remote sensing from space*, pages 134–141, 1988.

 ISBN: 978-84-617-8694-7