On speeding up k-means clustering using graphics processing units

William Xavier Maukoski, Lilian Tais de Gouveia, Luciano Jose Senger[1]

[1]Universidade Estadual de Ponta Grossa (UEPG) – Ponta Grossa – PR – Brasil

william_maukoski@hotmail.com.br, ltgouveia@uepg.br, ljsenger@uepg.br

Abstract. *Due to advancements in technology, modern farms work differently than those from past. Over the years, there was an increase in the number of data collected by sensors, cameras and other systems. In this scenario, the implementation of data mining on parallel computing systems is crucial for ensuring system scalability and better performance as data continues to grow. This paper presents a case study on a parallel implementation of the K-means clustering. Clustering has been used in many applications including image processing, information retrieval and climatology. However, k-means clustering is knowing to be computationally expensive when applied to obtain clusters from large datasets. The parallel k-means implementation is target to general purpose graphics processing units. In order to implement the k-means clustering to this parallel architecture and to provide a better software platform to data science research, the Weka k-means implementation were chosen and adapted. First, a profiler was used to identify the most time-consuming portions of code. By using a profiler, it was possible to verify a 95.57% reduction in the number of lines of code that would need to be analyzed to rewrite the code. After, a parallel k-means clustering was implemented and evaluated. The results show that using the parallel k-means clustering and graphics processing units, data mining results can be achieved in reduced times. The speedup achieved was up to 26 when using all available execution cores.*

Keywords: Parallel Computing, K-means, General purpose graphics processing units.

1. Introduction

Modern farms work differently than those from past, primarily because of advancements in technology, including sensors, devices, machines and farming management systems. Today's agriculture uses technologies such as temperature and moisture sensors, images from unmanned aerial vehicles and geographic information systems [1]. These devices and technologies allow farms to be more efficient, safer and more environmentally friendly.

Over the years, there was an increase in the number of sensors in the field, resulting in an increase in the number of data collected [1]. To be able to work with this data flow, it is necessary to apply techniques known as Data Mining (DM). These techniques allow to extract patterns and relationships using machine learning algorithms. From the knowledge obtained with the DM, it is possible to predict behaviors and assist in decision making [1,2].

Due to large data volume available used as input, the implementation of data mining techniques on parallel computing systems is crucial for ensuring system scalability as data continues to grow. In this scenario, parallel computing has been a viable means to reduce data mining computing times. Among parallel architectures, General Purpose Graphics Processing Units (GPGPU) is an option to speed up data mining computing [2].

The number of cores in GPU hardware is massively greater than the number of processing cores in general purpose CPUs. GPUs are dedicated hardware for manipulating computer graphics. Due to the computing demand for 3D graphics modeling and rendering, GPUs have evolved into

an organization of highly parallel many-core processors [3]. The advances in GPU architectures have driven the development of general-purpose computing on GPUs (GPGPU).

Although it is mistakenly considered a recent technology, GPU programming is used from the early 1990's. There are challenges in algorithm design when using GPU manycore environments as target hardware. First, it is necessary a deep knowledge of the algorithm (parallel or sequential) that will be written into a new version, target to GPU hardware manycore. Also, it is necessary that the algorithm that will be executed in manycore hardware must present a high level of parallelism, because despite the massive number of cores, GPU clock rates are usually below CPU clock rates [4].

This paper presents a case study on an implementation of the k-means algorithm in a GPU computing environment. K-means clustering is a very popular unsupervised machine learning method. It has been used in many applications including image processing, information retrieval and climatology [5]. However, k-means clustering is knowing to be computationally expensive when applied to obtain clusters from large datasets.

In order to implement the k-means clustering target to GPU systems and to provide a better software platform to data science research, the Weka k-means implementation in the Java language were used [6]. Weka is a free software licensed under the GNU General Public license and the parallel version was built using the original Weka k-means source code. By this way, the parallel k-means code can be easily experimented by accessing the Weka GUI.

2. Related Work

The first k-means algorithm [7] was proposed in 1967 and since it is used in data science and machine learning. In the k-means algorithm, we are given a finite set S of points in $\Re^m$, and an integer $k \geq 1$, and we want to find k points (centroids) so as to minimize the sum of the square of the distance of each point in S to its nearest center. This problem is well as NP-hard [8] and thus k-means is highly time-consuming when data and cluster sizes are elevated.

Recently, as a general-purpose and high-performance parallel hardware, GPUs develop continuously and provide another computing system for improving k-Means performance. GPUs are dedicated hardware for manipulating into highly parallel many-core processors. In [9], the first parallel k-means was presented. Since that, many attempts have been made to develop clustering algorithms to take advantage of the high-performance parallel computing systems [10].

Jaros et al. [11] presented a parallel k-means algorithm for image segmentation. The authors implemented an k-means target to Many Integrated Core (MIC) architecture composed of Intel Xeon Phi coprocessors. The authors evaluated the performance of MIC, GPU, CPU and sequential implementations. The authors related speedups values nearly to 12 and 33 in MIC and GPU, respectively, using different image and clusters sizes for clustering heart and liver images from human bodies.

Lutz et al. [5] presented a GPU-optimized algorithm for k-means. The authors' algorithm is based on a distinct strategy for updating centroids on GPUs. The reported k-means approach scales to very large data sets.

Li et al. [12] proposed a framework for parallel k-means computation and deployed the k-means algorithm on a many-core processor. The parallel efficiency achieved was almost equal to ideal using three randomly generated datasets.

Li et al. [13] designed a parallel k-means algorithm for GPUs by using the Compute Unified Device Architecture (CUDA) programming model [14]. The authors concluded that the dimensionality (number of attributes) of the data set is an important factor to be considered. The authors employed distinct algorithms for low and high dimensional data sets.

3. Material and Methods

The computing system used in the experiments is an Intel i5 7400 processor (4 cores and 4 threads clocked at 3.5 Ghz and at 4.0 Ghz when turbo boost is activated) with 8GB of 2.4 Ghz memory and equipped with a Nvidia GTX 1060 GPU (6GB of memory and 1280 cores). The CUDA software version used is 10.0.130.

The k-means algorithm takes an input parameter k. It organizes a set of n data instances into k clusters according to a similarity measure. Each instance of data is considered a vector in Euclidean space and thus can have multiple attributes. The mean values of each cluster, also called centroids, are a summary measure of the similarity of the data instances associated to this cluster. At the beginning, the k-means algorithm randomly selects k of the data instances as the initial centroid for each cluster. In each iteration, k-means associates each data object with its nearest centroid, according to a similarity metric. The Euclidean Distance is a common choice to measure the similarity between data instances. Next, the k-means algorithm computes new centroids by taking the mean of all the data objects in each cluster respectively. The process repeats until the changes in the centroids values are less than some predefined threshold. In the experiments, the cluster size was fixed to k=10.

The Weka k-means implementation in the Java language were used and extended to parallel version in the experiments. By this way, the parallel implementation can be easily experimented by accessing the Weka GUI. Software profilers were used aiming to identify function call sequences and portions of code that are time consuming in Weka k-means implementation. After, CUDA libraries were used to implement the parallel k-means computer program in the Java Language.

The speedup values of the parallel k-means program were computed as defined by Equation 1:

$$S(p) = \frac{T\ seq}{T\ par} \quad (1)$$

where *S(p)* is the speedup using *p* cores, *Tseq* is the execution time of the sequential version and *Tpar* is the execution time of the k-means parallel version. Furthermore, the execution times were studied using the Amdhal's law [15]. The Amdhal´s law gives the theoretical speedup in latency of the execution of a parallel program at a fixed data set that can be expected of a parallel execution system whose resources (processing cores) are improved.

In the experiments, the k-means clustering sequential and parallel implementations use as input a database which describes the soybean productivity in agricultural areas of the USA [16]. This database is composed of 80 different attributes, 6 classes and 12800 instances. Data augmentation techniques were used to increase the data set size and to better exploit the parallelism.

4. Results

Initially, the JProbe profiler was used to identify how much time is spent in each method of the Weka k-means algorithm. The results using a small portion of the data set can be seen in Table 1. As one can see, the major time-consuming portion in the Weka k-means implementation is the *moveCentroids* method. Due profiling, there was a reduction in the number of lines of code that must be analyzed. The k-means class has 2472 lines, while the *moveCentroids* method has 102 lines. By this way, it was observed a 95.57% reduction in the number of lines to be manually analyzed by the programmer.

Table 1 – Execution times in weka k-means main methods

Weka method	Execution times	Percentage (%)
Weka.clusterers.SimpleKMeans.moveCentroid	00:11:30	50
Weka.core.instance.weight	00:02:51	12
Weka.core.instance.numAttributes	00:01:47	7
Weka.core.instance.attributes	00:01:40	7
Weka.core.instance.isMissing	00:01:29	6
Weka.core.instance.value	00:01:28	6
Weka.core.Attribute.isNumeric	00:00:49	4
Weka.core.DistanceFunction.distance	00:00:38	2

After identifying the most time-consuming method, a top-down analysis of this method was performed. The method implements two data structures and a repeat loop. The data structure used by the *moveCentroids* method is depicted in Figure 1. This data structure is a linked list of instances. One field of this data structure stores a linked list of attributes. In order to implement the parallel version of the code using the JIT java technology, five additional data structures were defined: a) a vector of Boolean values with size of N x M, where N is the number of instances and M is the number of attributes. This vector registers the presence of numeric values (not nominal attributes); b) a vector of Boolean values, which is set to True when the value is numeric; c) a vector with NxM size, used to store the attribute values; d) a value with the weight of each attribute; e) an auxiliary data structure used to store values updated by parallel threads and used in k-means clustering.

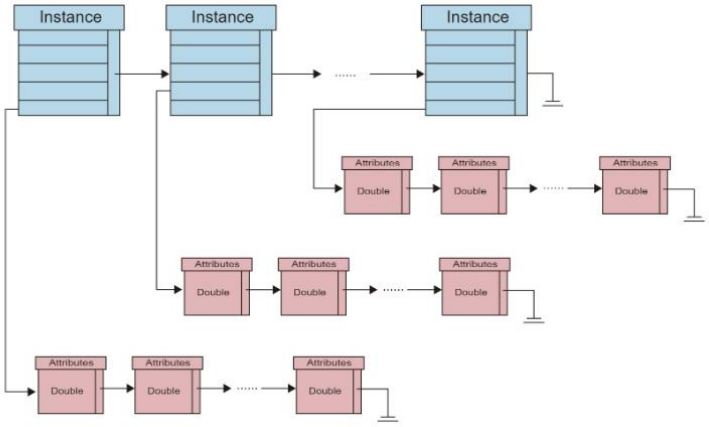

Figure 1 – The moveCentroids data structure.

The original Weka k-means implementation and the new parallel implementation were executed using the same data set and the results were compared. The Weka clustering report indicated that the two versions of k-means produced the same results (same number of clusters and same data distribution among clusters). The Table 2 show the computing times and the statistics using 1, 80, 160, 540 e 1080 threads. Each experiment was conducted 10 times for each number of threads.

The study group with 1080 GPU cores achieved the best execution times and the lowest associate standard deviation value. This number of cores achieved shortest response times, which implies lower variance. Also, due to number of cores used, this group is less susceptible from other system or application processes influence when clustering data.

The Figure 2 shows a plot of the mean execution times divided by the mean number of computing threads. A reduction in response times was observed as the number of threads increased, thus characterizing a horizontal asymptote with a limit imposed by the execution time of the sequential portion of the code, as defined by Amdhal's Law. The linear regression model, when using the number of threads by the execution times, explains nearly 71% of the experiment data ($R^2 = 0,7135$).

Table 2 – Parallel k-means program execution times.

Experiment/ #of threads	1	80	160	540	1080
1	03:46:15	00:23:42	00:16:14	00:12:05	00:08:46
2	03:47:22	00:23:56	00:16:10	00:12:01	00:08:38
3	03:47:18	00:24:05	00:16:05	00:11:43	00:08:41
4	03:47:23	00:24:08	00:16:17	00:11:48	00:08:50
5	03:46:25	00:23:45	00:16:37	00:11:43	00:08:41
6	03:46:40	00:23:22	00:16:06	00:11:59	00:08:46
7	03:46:25	00:23:31	00:16:37	00:12:09	00:08:43
8	03:46:06	00:23:41	00:16:29	00:11:42	00:08:38
9	03:45:54	00:23:49	00:16:12	00:12:10	00:08:44
10	03:47:26	00:23:55	00:16:05	00:11:46	00:08:41
Mean values	03:46:32	00:23:47	00:16:13	00:11:54	00:08:43
Standard deviation	0,59	0,24	0,20	0,18	0,06
Speedup		9,52	13,96	19,03	26,00

A logarithmic scale plot of the speedup values by number of threads are depicted in Figure 3. The speedup curve shows an asymptotic behavior. The regression model achieved a R^2 value nearly to 0,80. The speedup behavior allow to observe that better speedup values can be achieved when using more GPU cores.

5. Conclusion

This paper presents an implementation and a performance evaluation of a parallel k-means designed to GPU hardware. K-means is a common choice when data scientists aims to obtain new insights by exploiting machine learning data sets and, since GPUs have become faster, it is important to enable machine learning tools to exploit GPU parallel execution cores. Our implementation works with the widely adopted Weka data mining tools.

By using the profiler, it was possible to verify a 95.57% reduction in the number of lines of code that would need to be analyzed to rewrite the code for GPU hardware. Using the GPU hardware, data mining results can be achieved in short times. The use of the GPU hardware resulted in a speedup equal to 26 when using all available GPU cores.

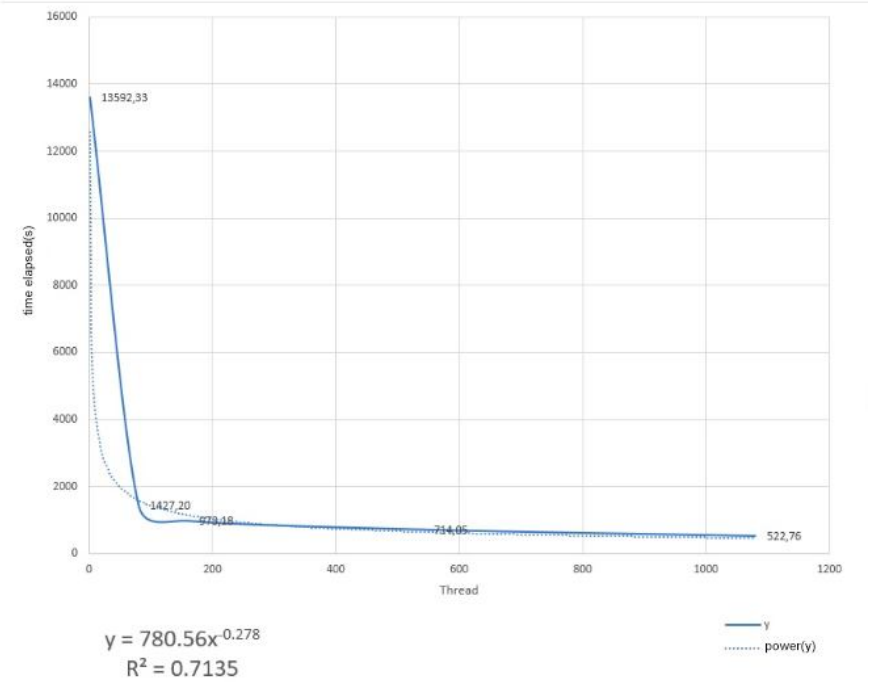

Figure 2 – Mean execution times when using distinct number of threads and the linear regression model.

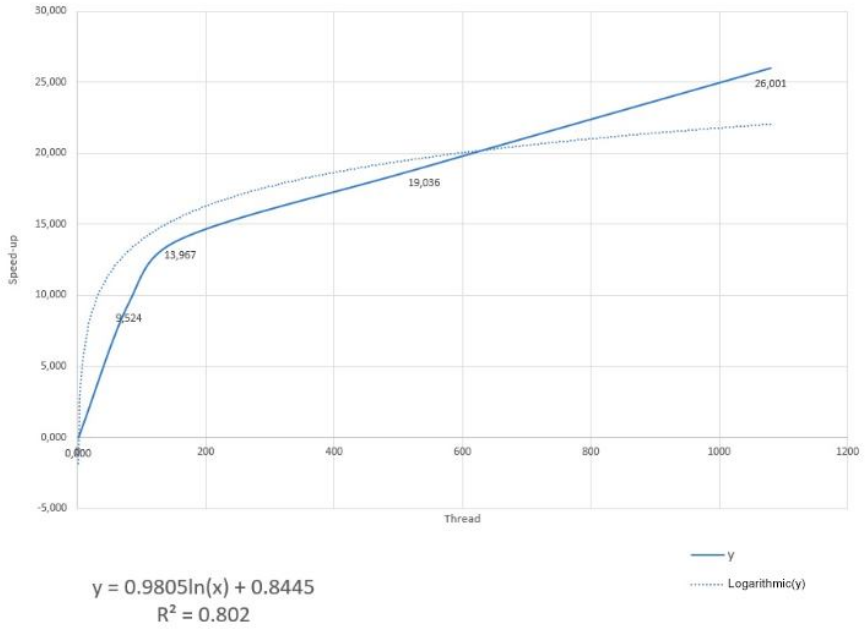

Figure 3 – Speedup values

6. References

[1] KAMILARIS, A.; PRENAFETA-BOLDU, F.X. Deep learning in agriculture: A survey. Computers and Electronics in Agriculture, v. 147, p. 70-90, 2018.

[2] RAMESH, V.; RAMAR, K; BABU, S. Parallel K-Means Algorithm on Agricultural Databases. IJCSI International Journal of Computer Science Issues, v.10, n. 1, 2013.

[3] LI, Y.; ZHAO, K.; CHU, X.; LIU, J. Speeding up k-Means algorithm by GPUs. Journal of Computer and System Sciences, v. 79, n. 2, 2013.

[4] GURCAN, I.; TEMIZEL, A. Heterogeneous CPU–GPU tracking–learning–detection (H-TLD) for real-time object tracking. Journal of Real-Time Image Processing, n.16, p. 339–53, 2019.

[5] LUTZ, C.; BREß, S.; RABL, T.; ZEUCH, S.; MARKL, V. Efficient and Scalable k-Means on GPUs. Datenbank Spektrum, n. 18, p. 157–169, 2018.

[6] FRANK, E; HALL, M. A.; WITTEN, I. H. The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques". Morgan Kaufmann, 4.ed., 2016.

[7] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probabilities, n.1, p. 281-296, 1967.

[8] LEE, E.; SCHMIDT, M.; WRIGHT, J. Improved and simplified inapproximability for k-means, Information Processing Letters, v. 120, p. 40-43, 2017.

[9] LI, X.; Fang, Z. Parallel clustering algorithms, Parallel Computing, v.11, n. 3, p. 275-290, 1989.

[10] SARDAR, T. H.; ANSARI, Z. An analysis of MapReduce efficiency in document clustering using parallel K-means algorithm. Future Computing and Informatics Journal, v. 3, n. 2, 2018.

[11] JAROS, M.; STRAKOS, P.; KARASEK, T.; RIHA, L.; VASATOVA, A.; JARASOVA, M.; KOZUBER, T. Implementation of K-means segmentation algorithm on Intel Xeon Phi and GPU: Application in medical imaging. Advances in Engineering Software, v. 103, p. 21-28, 2017.

[12] LI, M.; YANG, C.; SUN, Q.; MAO, W.; CAO, W.; AO, Y. Enabling highly efficient k-means computations on the SW26010 many-core processor of Sunway Taihu Light. Journal of Computer Science and Technology, v. 34, n. 1, p. 77-93, 2019.

[13] LI, Y.; ZHAO, K.; CHU, X.; LIU, J. Speeding up k-Means algorithm by GPUs. Journal of Computer and System Sciences, v. 79, n. 2, p. 216-29, 2013.

[14] NICKOLLS, J.; BUCK, I.; GARLAND, M.; SKADRON, K. Scalable Parallel Programming with CUDA. Queue, n. 6, p. 2, p. 40–53, 2008.

[15] AMDDHAL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. Spring Joint Computer Conference, p. 483-485, 1967.

[16] MICHALSKI, R. S.; CHILAUSKY, R. L. Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.