ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-909

Evolutionary Clustering on CUDA

Pavel Krömer and **Jan Platoš** and **Václav Snášel**[1]

Abstract. Unsupervised clustering of large data sets is a complicated task. Due to its complexity, various meta-heuristic machine learning algorithms have been used to automate the clustering process. Genetic and evolutionary algorithms have been deployed to find clusters in data sets with success. The GPU computing is a recent programming paradigm introducing high performance parallel computing to general audience. This work presents an acceleration of a genetic algorithm for density based clustering on the GPU using the nVidia compute unified device architecture (CUDA).

1 INTRODUCTION

Clustering represents a fundamental data analysis task of separation of objects to classes with a plenty of practical applications. The most often used clustering approaches include the hierarchical clustering, the centroid (medoid) based clustering, and the density based clustering [7]. The density based clustering is popular for its ability to discover clusters with arbitrary shapes. Informally, a density based cluster C is a set of points in the problem space that are *density connected*, i.e. for each pair of points in C there is a chain of points with distance between two consecutive points smaller than a constant ϵ.

Various evolutionary algorithms have been used to find meaningful clusters in data [5, 3]. The design of an evolutionary algorithm for clustering involves, among others, the definition of candidate solution (clustering) encoding and the choice of suitable fitness function that would evaluate the quality of candidate solutions.

2 GENETIC ALGORITHM FOR CLUSTERING ON CUDA

The genetic algorithm (GA) for clustering proposed in this study uses real encoded chromosomes with variable length, a parallel density based clustering approach, and the Dunn index (DI) as cluster validity measure. The real encoding [5] uses real numbers to encode arbitrary points in the problem domain. The points provide a representation of clusters in the encoded partitioning. The DI is an internal clustering validity measure defined by [4]:

$$D = \min_{1\leq i\leq n}\left\{\min_{1\leq j\leq n, i\neq j}\left\{\frac{d(i,j)}{\max_{i\leq k\leq n} d'(k)}\right\}\right\} \quad (1)$$

where $d(i,j)$ is the distance between clusters i and j and $d'(k)$ is the diameter of cluster k:

$$d(i,j) = \min_{a\in i, b\in j}\{dist(a,b)\}, \quad d'(k) = \max_{a,b\in k}\{dist(a,b)\} \quad (2)$$

The Euclidean distance $dist(a,b)$ was used to express the distance between objects in the data set and a pre-computed $N \times N$ distance matrix D defined by $D_{ij} = dist(i,j)$ was used to avoid repeated evaluation of the distances.

Three CUDA-C kernels implement the clustering algorithm. The *cudaPlacePins* kernel maps all cluster representatives (pins) encoded in the chromosome to the closest object in the data set (seed). It calculates the distance between the pin and every object in the data set because the distance matrix D cannot be used as the location of the pins changes during the evolution. The parallel implementation launches a thread block for each pin in the chromosome and uses each thread in the block to compute the distance between the pin and a number of objects in the data set.

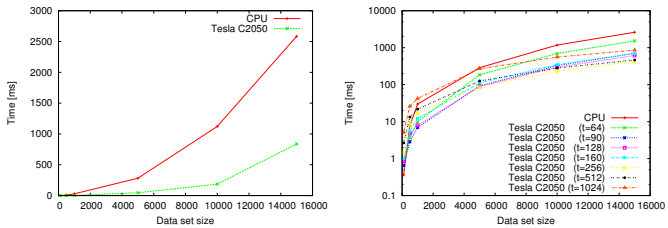
The *cudaFormClusters* kernel implements the formation of the density-based clusters. Each cluster is expanded using a stack breadth-first search (BFS). The expansion starts with the seeds found in the previous step and it iteratively appends to the cluster all objects that are directly density connected to the cluster. There are several BFS implementations for the GPU (e.g. [8]), however, they do expect a single BFS instance running at the same time while we run k BFS instances in parallel. Similar approach was recently presented in [2]. This implementation uses only a simple collision detection and avoids locking and atomic operations to improve the performance. A point that is density reachable from more forming clusters is assigned to one of them. Such a situation is a sign of poor clustering with clusters too close to each other that will be awarded with a low fitness and it will not survive in the evolution.

The *cudaDunnIndex* kernel implements the DI evaluation. It finds the minimum distance between every two clusters and maximum distance between any two points in the same cluster at the same time by a single parallel scan of the distance matrix. The kernel was implemented with minimum branching to optimize the performance.

3 EXPERIMENTAL EVALUATION

We have tested the performance and correctness of the GA for density based clustering on the GPU. The experiments were conducted on a PC with a 2.6 GHz CPU and an nVidia Tesla C2050 card with 448 cores at 1.15 GHz. Several data sets containing 100 to 15000 objects were generated to test the performance of the kernels. They were based on the *data_3_2* data set from [1] which was extended by generating additional points within the shape of its original clusters. We have measured the time needed to compute the DI and to form clusters. The kernel cudaDunnIndex is not data bound and it was executed with the largest block size (1024). The kernel cudaFormClusters was executed with different number of threads per block because it is data bound. The execution time of the DI computation and cluster formation on the CPU and GPU is shown in fig. 1(a) and fig. 1(b). Clearly, the DI computation on the GPU is faster than the sequential implementation. The GPU is 2.64 to 15.20 times faster than the CPU when computing the DI. The speedup in cluster formation achieved

[1] VŠB - Technical University of Ostrava & IT4 Innovations, Ostrava, Czech Republic, email: {pavel.kromer,jan.platos,vaclav.snasel}vsb@cz



(a) DI computation time. (b) Cluster formation time.

Figure 1. Performance of the GPU implementation

by the GPU is shown in table 1. As expected, the CPU was faster

Table 1. Cluster formation speedup on the GPU.

	Block size						
N	64	90	128	160	256	512	1024
100	0.56	0.56	0.43	0.37	0.25	0.14	0.07
500	2.09	**2.72**	1.61	1.35	0.93	0.58	0.30
1000	3.03	**4.29**	3.77	2.47	1.77	1.38	0.71
5000	1.55	3.11	3.25	2.56	**3.35**	2.28	1.07
10000	1.69	3.59	3.98	3.36	**5.1**	4.19	2.12
15000	1.71	3.71	4.25	3.75	**6.1**	5.64	3.04

for the smallest data set because it can benefit from its architecture. The GPU was able to speed up the cluster formation 2.7 to 6.1 times. However, the performance evaluation is rather illustrative because the performance of the kernel is data bound, i.e. the speedup factor will be different for other data sets.

The ability of the GPU accelerated GA to find good partitioning of different data sets with irregular clusters was tested on the modified Chameleon data [6]. Four data sets were created from Chameleon by noise reduction: rt4 with 4231 objects, rt5 with 4407 objects, rt7 with 5305, and rt8 with 4877 objects. The GA used a population of 100 candidate solutions, neighborhood size $\epsilon = 10$, crossover probability 0.8, and mutation probability 0.6. The GA was executed for different number of generations with thread block sizes 90 and 256.

The performance comparison of the CPU and GPU implementation for different data sets is shown in table 2(a) and the speedup for different number of generations is shown in table 2(b). The speedup is for all four data sets and both block sizes almost the same. The total execution time of the algorithm on the GPU is approximately 5-6 times shorter. Moreover, the average speedup is consistent at different generations. The GA was in most cases able to identify correct

Table 2. Average speedup on the GPU for different block sizes (BS).

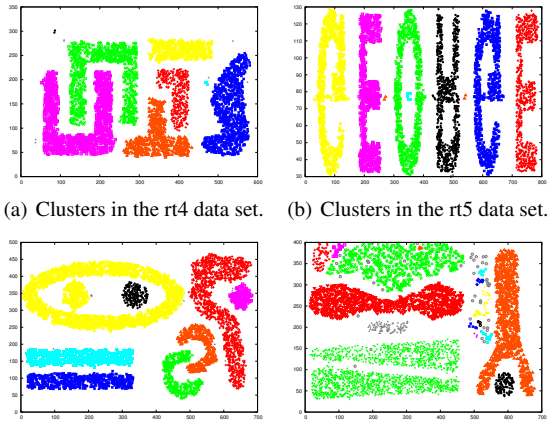
(a) For different data sets

dataset	BS 90	BS 256
t4	5.28	5.34
t5	5.74	5.76
t7	6.14	6.21
t8	5.76	5.77

(b) For different generations

gen.	BS 90	BS 256
200	5.72	5.71
500	5.75	5.80
1000	5.72	5.79

partitioning of the data before reaching 500 generations. Both, the CPU and GPU implementations delivered correct results. The largest clusters were identified and the remaining outlying points were gathered in the remainder cluster. The clusters found by the GA accelerated by the GPU are shown in fig. 2. Let us note that the left circle inside the ellipse in rt7 is density connected to the ellipse, the two triangles in rt8 really are density connected, and the upper left cluster and the sparse vertical clusters in rt8 are composed of multiple clusters.



(a) Clusters in the rt4 data set. (b) Clusters in the rt5 data set.

(c) Clusters in the rt7 data set. (d) Clusters in the rt8 data set.

Figure 2. Clusters in the chameleon dataset.

4 CONCLUSIONS AND FUTURE WORK

This study presents a design and initial implementation of a GA for clustering accelerated by the GPU. A simple density based clustering and the DI were used as GPU powered building blocks of the algorithm that was shown to outperform its sequential counterpart more than 5 times.

The cudaComputeDunn kernel can be improved to store the collision matrix to shared or texture memory. A better v of the clustering algorithm less sensitive to noise will be implemented and the performance of the CPU and GPU based GA for clustering without the pre-computed distance matrix will be compared.

ACKNOWLEDGEMENTS

This paper has been elaborated in the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 supported by Operational Programme 'Research and Development for Innovations' funded by Structural Funds of the European Union and state budget of the Czech Republic and supported by SGS, VŠB – Technical University of Ostrava, under the grant no. SP2012/58.

REFERENCES

[1] S. Bandyopadhyay and U. Maulik, 'Genetic clustering for automatic evolution of clusters and application to image classification', *Pattern Recognition*, **35**(6), 1197 – 1208, (2002).

[2] C. Böhm, R. Noll, C. Plant, and B. Wackersreuther, 'Density-based clustering using graphics processors', in *Proc. of the 18th ACM conference on Information and knowledge management*, CIKM '09, pp. 661–670, New York, NY, USA, ACM, (2009).

[3] S. Das, A. Abraham, and A. Konar, 'Metaheuristic pattern clustering an overview', in *Metaheuristic Clustering*, vol. 178 of *Studies in Comp. Intelligence*, pp. 1–62, Springer, (2009).

[4] J. C. Dunn, 'Well separated clusters and optimal fuzzy-partitions', *Journal of Cybernetics*, **4**, pp. 95–104, (1974).

[5] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. Ponce Leon F. De Carvalho, 'A survey of evolutionary algorithms for clustering', *Trans. Sys. Man Cyber Part C*, **39**, pp. 133–155, (2009).

[6] G. Karypis, E.-H. Han, and V. Kumar, 'Chameleon: hierarchical clustering using dynamic modeling', *Computer*, **32**(8), pp. 68 –75, (1999).

[7] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, 'Density-based clustering', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, **1**(3), pp. 231–240, (2011).

[8] L. Luo, M. Wong, and W.-M. Hwu, 'An effective gpu implementation of breadth-first search', in *Proc. of the 47th Design Automation Conf.*, DAC '10, pp. 52–55, New York, NY, USA, ACM, (2010).