Parallel implementation of the greedy heuristic clustering algorithms

Parallel implementation of the greedy heuristic clustering algorithms

L A Kazakovtsev[1,2], I P Rozhnov[1], E A Popov[1], M V Karaseva[1,2] and A A Stupina[1,2]

[1] Reshetnev Siberian State University of Science and Technology, 31, Krasnoyarskiy Rabochiy avenue, Krasnoyarsk, 660031, Russia
[2] Siberian Federal University, 79, Svobodny avenue, Krasnoyarsk 660041, Russia

E-mail: levk@bk.ru

Abstract. Authors propose parallel greedy heuristic k-means clustering algorithms for implementation on the graphical processing units (GPU) for solving large-scale problems. The computational experiments illustrate high performance of the GPUs in comparison with running the greedy heuristic algorithms on a central processor unit which is especially significant in the case of big datasets and bug numbers of clusters. The efficiency of the greedy heuristic algorithms in comparison with the standard k-means algorithm remains.

1. Introduction

Automatic grouping (clustering) systems become increasingly widespread due to the expansion of the application area of data analysis problems such as image recognition, solution of diagnostic problems in medicine, marketing research, Internet traffic research, etc. [1-3].

The k-means problem, along with a very similar p-median problem, is one of the classical problems of location theory [4]. The k-means problem is to find such k cluster centers $X_1...X_k$ in a d-dimensional space that the sum of squares of distances from them to given points A_i reaches its minimum.

$$F(X_1,...,X_k) = arg \min_{X \in \Re^d} \sum_{i-1}^{N} \min_{X \in \{X_1,...,X_k\}} \left\| A_i - X \right\|^2 , \quad (1)$$

The most popular method for solving the k-means problem is the algorithm of the same name, also known as the Lloyd's algorithm or the ALA procedure (Alternating Location-Allocation). The k-means algorithm sequentially improves the known solution, allowing us to find a local minimum. In the strict sense, this algorithm is not a local search algorithm, since the search for a new solution is not necessarily carried out in an ε-neighborhood of the existing solution. This is a simple and fast algorithm applicable to the widest class of problems. The algorithm has some limitations, in particular, the number of groups k must be known in advance. This algorithm can be described as follows.

```
Algorithm 1  k-means
Required: data vectors $A_1...A_N$ and $k$ initial cluster centers $X_1...X_k$
    do
        1: For each center $X_i$, build a cluster $C_i$ of data vectors so that for each data vector of this
           cluster, center $X_i$ is the nearest of all centers.
        2: For each cluster, calculate new value of center $X_i$.
    until steps 1-2 alter at least one cluster.
```

The aim of our study is to improve the accuracy of the result of solving the k-means problem to obtain the most accurate (by the value of the objective function) and a stable result, for a fixed, limited time, with the use of modern parallel GPU (Graphical Processor Unit) systems.

2. Clustering Algorithms of the Greedy Heuristic Method

In [3, 5], authors consider the application of genetic algorithms with a greedy agglomerative heuristic procedure, as well as modifications of the EM algorithm for the separation of homogeneous batches of industrial products and show the advantage of new algorithms over classical clustering algorithms for multidimensional data.

The greedy agglomerative heuristic procedure for the problem of k-means and similar problems [6] consists of two steps. Suppose that there are two well-known (parental) solutions to the problem (the first of which, for example, is the best of known solutions), which are represented by the sets of cluster centers *S*. First, the sets of parent decisions are merged (unified). We obtain an intermediate invalid solution with an excessive number of clusters. Then, the number of centers is gradually reduced. In each iteration, algorithm eliminates such a center, that its removal results in the least significant deterioration in the value of the objective function (1).

Algorithm 2 is the basic greedy heuristic algorithm, which sequentially reduces the number of clusters (given by centers):

Algorithm 2 Basic Greedy Agglomerative Heuristic Procedure for Large Clustering Problems
Required: initial number of clusters *K*, required number of clusters $k<K$, $k>50$, initial solution *S*, $|S|=K$.

```
    1: Improve the solution $S$ with Algorithm 1 (if possible).
    while $K \neq k$ do
        for each $i' \in \{\overline{1,K}\}$ do
        2: $S' = S \setminus \{X_{i'}\}$. Caclulate $F'_{i'} = F(S')$ where $F(.)$ is the value of the objective function (1).
        end do
        3. Form set $S_{elim}$ of $n_{elim}$ of centroids, $S_{elim} \subset S$, $|S_{elim}| = n_{elim}$, with minimum values of $F'_{i'}$. Here,
            $n_{elim} = \max\{1, 0.2 \cdot (|S| - k)\}$.
        4: Compose new solution $S = S \setminus S_{elim}$, $K = K-1$, and improve it with Algorithm 1.
    end do
```

Ways of merging solutions may be different. One of such ways is elementwise merging [8]:

Algorithm 3 Greedy Procedure #1
Required: two "parent" sets (arrays) of cluster centers $S' = \{X'_1, \ldots, X'_k\}$ and $S'' = \{X''_1, \ldots, X''_k\}$

```
    Calculate the objective function (1): $F^* = F(S')$;
    Arrange the elements of $S''$ in ascending order of values $F(S' \cup \{X''_{i'}\})$.
    for each $i' \in \{\overline{1,K}\}$ do
        1: Attach an element of $S''$ to $S'$: $S = F(S' \cup \{X''_{i'}\})$.,
        2: Run Algorithm 2 with initial solution $S$. Save the obtained set of cluster centers $S_{i'}$ and
        corresponding value $F_i$ of the objective function (1). If $F_i < F^*$ then $S' = S$.
    end do
    3. Return the best solution obtained in Step 2.
```

In [3, 5], author propose simpler ways of merging.

Algorithm 4 Greedy Procedure#3:

```
1: Combine sets $S = S' \cup S''$.
2: Run Algorithm 2 with $S$.
```

Algorithm 5 Greedy Procedure#2: 1: Combine sets $S = S' \cup S''$. 2: Run Algorithm 2 with *S*. Generate randomly $r' \in [0;1)$. Calculate $r=[(k/2-2)\,r'2]+2$. Form a randomly chosen subset S''' of S'' of cardinality *r*. Combine sets $S = S' \cup S'''$. Run Algorithm 2 with *S*.

These greedy heuristic procedures formed the basis for a wide variety of efficient genetic algorithms [1, 2], where these procedures are used as crossover operators, as well as VNS algorithms (Variable Neighborhood Search) [7].

The idea of this work is to implement the algorithms of the Greedy Heuristics Method using GPU systems [8] and investigate their properties when solving problems of high dimensionality.

3. Compute Unified Device Architecture

CUDA (Compute Unified Device Architecture) is a software&hardware architecture for parallel computing, which can significantly increase computational performance through the use of graphics processors from Nvidia company[8]. Researchers use CUDA extensively in various fields, including video and image processing, computational biology and chemistry, fluid dynamics modeling, image recovery from computed tomography, seismic analysis, ray tracing etc.

Weak points of using previous GPU programming methods are that they do not use vertex shader execution blocks, data are stored in textures only, and multipass algorithms use pixel shader units [9]. Limitations of previous GPU programming methods can include: insufficient use of hardware capabilities, limited memory bandwidth, no scatter operation (only gather), mandatory use of the graphics API [9].

The main advantage of CUDA is that this architecture is designed to effectively use non-graphical computing on the GPU and uses the C programming language without requiring the transfer of algorithms to a convenient form for the concept of a graphics pipeline. CUDA does not use graphical APIs, offering random access to memory (scatter or gather) [8].

Performing calculations on the GPU shows excellent results in algorithms that use parallel data processing, in contrast to algorithms implemented on the CPU, if the same sequence of commands is applied to a large amount of data. The best results are achieved if the ratio of the number of arithmetic instructions to the number of memory accesses is large enough. This places less demands on flow control, and the high density of calculations and large amounts of data eliminate the need for large caches which are rather efficient on a CPU.

4. Parallel Implementation of Greedy Algorithms

Various parallel versions of the k-means algorithms are known [8, 10]. For the second part of the algorithm which realizes the 1st step of Algorithm 1, we used a single CUDA thread.

```
Algorithm 1.1a CUDA realization of Step 1 of  Algorithm 1, part 1.
$X'_j$=0 for all $j \in \{\overline{1,k}\}$. // Here, $X'_j$ are vectors used for calculation of new cluster centers.
$counter_j$=0 for all $j \in \{\overline{1,k}\}$. // object counters for each cluster.
```

For the second part of the algorithm which realizes the 1st step of Algorithm 1, we used $N_{trreads}$=512 threads for each CUDA block. Number of blocks is calculated as

$$N_{blocks}=(N+N_{threads}-1)/N_{threads}. \tag{2}$$

Thus, each thread processes only one data vector.

```
Algorithm 1.1b CUDA realization of Step 1 of  Algorithm 1, part 2
$i$ = $blockIdx.x * blockDim.x + threadIdx.x$ .
if $i>N$ then Return.
$j'$=arg $\min_j \left\|A_j - X_i\right\|^2$. // number of cluster
```

```
$X'_{j'}=X'_{j'}+A_i$.
$C_i=j'$. // Assign $A_i$ to cluster $j'$.
$counter_{j'}=counter_{j'}+1$.
```

Synchronize threads.

For the second part of the algorithm which realizes the 2nd step of Algorithm 1, we used $N_{trreads}$=512 threads for each CUDA block. Number of blocks is calculated as $N_{blocks2}=(k+N_{threads}-1)/N_{threads}$.

Algorithm 1.2a CUDA realization of Step 2 of Algorithm 1

```
$j$ = blockIdx.x * blockDim.x + threadIdx.x .
if $j>k$ then Return.
$X_{j'}= X'_i/counter_j$.
Synchronize threads.
```

In addition, we implemented Step 2 of Algorithm 2 on the GPU. At this step, Algorithm 2 calculates the total distance after removing one cluster: $F'_{i'}=F(S')$, where $S'=S\setminus\{X_{i'}\}$. Having calculated F(S), it we can calculate $F'_{i'}=F(S')= F(S) + \sum_{l=1}^{N}\Delta D_l\cdot$, where

$$\Delta D_l\cdot=\begin{cases}0, & C_{i'}\neq l,\\ \left(min_{j\in\{\overline{1,k}\},\, j\neq i'}\left\|A_j - X_j\right\|^2\right)-\left\|A_j - X_{C_{i'}}\right\|^2, & C_{i'}=l.\end{cases} \quad (3)$$

Here, we used 512 threads for each CUDA block, number of blocks is calculated in accordance with (2). First, variable *sumD* in initialized with 0. Then, the following algorithm runs for each data vector and calculates ΔD_l .

Algorithm 2.2a CUDA realization of Step 2 of Algorithm 2

```
$l$ = blockIdx.x * blockDim.x + threadIdx.x .
if $l>k$ then Return.
Calculate $\Delta D_l$ in accordance with (3).
If $\Delta D_l$ >0 then atomicAdd(sumD, $\Delta D_l$ ).
Synchronize threads.
```

All other algorithms are run on the central processor.

5. Experimental results

For our study, we used classical data sets from the UCI (Machine Learning Repository) [11] and Clustering basic benchmark [12] repositories. The system was as follows: of Intel Core 2 Duo E8400CPU, 4GBRAM. NVIDIA GeForce 9600 GT graphics processor, with 2048 MB of RAM.

For all data sets, we performed 30 attempts to run each of the 10 algorithms (k-means, k-VNS1, k-VNS2, k-VNS3, k-VNS1-RND, k-VNS2-RND, k-VNS3-RND, GA -FULL, GA-MIX, GA-ONE). Only the best results achieved in each attempt were recorded, then from these results for each algorithm the minimum and maximum values (Min, Max), mean value (Average) and standard deviation (Std.dev.) were calculated. The k-means algorithm was launched in multi-start mode. The best values of the objective function (minimum value, mean value and standard deviation) are shown in bold italics (Tables 1-3).

In our earlier research of the BIRCH-3 data set [8] without CUDA technology, the best value (3.72525E+13) for the minimum objective function was obtained subject to 6 hours for each attempt. When calculating using the GPU (Table 2), we obtained the minimum value of the objective function 3.71473E+13 with the same algorithm (its CUDA version), in 10 minutes, and little worse value (3.72082E+13) in 1 minute.

Table 1. Results of experiments with dataset Mopsi-Joensuu (180 seconds, 30 attempts).

Algorithm	Objective function value			
	Min	Max	Average	Std. Dev.
	100 clusters			
k-means	20.2234	25.1256	22.6732	1.9230
k-VNS1	1.8518	2.0704	1.9320	0.0996
k-VNS2	***1.6519***	1.7969	1.7335	0.0504
k-VNS3	1.6745	1.7950	1.7301	***0.0444***
k-VNS1-RND	1.9142	2.9365	2.2084	0.3680
k-VNS2-RND	1.7589	2.0456	1.8427	0.1026
k-VNS3-RND	1.6558	1.8107	1.7204	0.0646
GA-FULL	***1.6544***	1.7569	***1.6760***	***0.0398***
GA-MIX	1.6600	17.7807	5.4884	6.5581
GA-ONE	19.0837	33.0772	26.8381	4.5549
	300 clusters			
k-means	5.6141	8.9812	7.7135	1.1162
k-VNS1	2.0335	3.4027	2.6656	0.4973
k-VNS2	5.1070	11.1468	8.9344	2.2980
k-VNS3	***0.1432***	0.2974	***0.1836***	***0.0582***
k-VNS1-RND	2.2020	4.3911	2.7338	0.8446
k-VNS2-RND	6.7474	14.6131	10.9959	2.6691
k-VNS3-RND	***0.1533***	14.4612	9.1619	5.6364
GA-FULL	0.2073	3.6894	1.2855	1.5409
GA-MIX	0.7039	2.5733	1.4348	0.6968
GA-ONE	8.0874	15.9837	11.8232	3.1623

Table 2. Results of experiments with dataset BIRCH-3 (100 clusters, 30 attempts).

Algorithm	Objective function value			
	Min	Max	Average	Std. dev.
	60 seconds			
k-means	8.18676E+13	9.96542E+13	8.98255E+13	8.37212E+12
k-VNS1	3.71973E+13	3.76732E+13	3.73639E+13	0.18509E+12
k-VNS2	3.73240E+13	4.06161E+13	3.91485E+13	1.14305E+12
k-VNS3	3.72082E+13	3.72550E+13	***3.72422E+13***	***0.01998E+12***
k-VNS1-RND	3.71993E+13	3.76607E+13	3.73757E+13	0.18322E+12
k-VNS2-RND	3.98574E+13	5.17877E+13	4.47900E+13	4.74952E+12
k-VNS3-RND	***3.71558E+13***	3.73328E+13	***3.72362E+13***	0.06507E+12
GA-FULL	3.74076E+13	3.84774E+13	3.75950E+13	0.34167E+12
GA-MIX	3.76402E+13	4.13519E+13	3.84577E+13	1.44968E+12
GA-ONE	6.36816E+13	9.10870E+13	7.47659E+13	11.6766E+12
	600 seconds			
k-means	7.98405E+13	9.96542E+13	8.93187E+13	9.04845E+12
k-VNS1	***3.71474E+13***	3.71933E+13	***3.71778E+13***	***0.02348E+12***
k-VNS2	***3.71474E+13***	3.72261E+13	***3.71834E+13***	0.02595E+12
k-VNS3	***3.71473E+13***	3.72453E+13	***3.71817E+13***	0.03723E+12
k-VNS1-RND	***3.71474E+13***	3.71932E+13	***3.71775E+13***	***0.02326E+12***
k-VNS2-RND	***3.71474E+13***	3.72275E+13	***3.71853E+13***	0.03177E+12
k-VNS3-RND	***3.71474E+13***	3.72275E+13	***3.71857E+13***	0.03163E+12
GA-FULL	3.72332E+13	3.74141E+13	3.72741E+13	0.06510E+12
GA-MIX	3.71525E+13	3.72071E+13	3.71949E+13	***0.02097E+12***
GA-ONE	3.71495E+13	3.7233E+13	3.71906E+13	0.04180E+12

Table 3. Results of experiments with dataset KDDCUP04BioNormed (2000 clusters, 14 hours, 30 attempts).

Algorithm	Objective function value			
	Min	Max	Average	Std. dev.
k-means	4 424 475	4 426 251	4 425 137	***786.5***
k-VNS1	4 358 583	4 386 584	4 367 311	12 966.3
k-VNS2	4 338 584	4 419 181	4 378 916	42 724.9
k-VNS3	***4 311 992***	4 318 547	***4 315 658***	2 721.5
GA-FULL	***4 314 647***	4 319 851	***4 316 581***	2 847.4
GA-MIX	4 332 422	4 354 462	4 342 210	11 224.5
GA-ONE	4 426 306	4 431 211	4 428 233	2 615.5

6. Conclusions

Note the following: the clustering algorithms of the Greedy Heuristic Method, which show the best results of the objective function with a small number of clusters, are not always the best with the increase in the number of clusters. However, the advantage of the family of greedy heuristic algorithms over the k-means algorithm remains after transition to the CUDA architecture. The use of a GPU shows an advantage in the achieved speed in comparison with the calculations on the CPU, and the advantage increases for large data sets and a large number of clusters.

Acknowledgements
Results were obtained in the framework of the state task No. 2.5527.2017/8.9 of the Ministry of Education and Science of the Russian Federation.

References

[1] Vempala S and Wang G 2002 A spectral algorithm for learning mixtures of distributions *FOCS* 841-60
[2] Orlov V, Stashkov D, Kazakovtsev L, Rozhnov I, Nasyrov I and Kazakovtseva O 2018 Improved method of production batchs of electronic components with special quality requirements *Modern high technology* **1** 37-42
[3] Kazakovtsev L and Antamoshkin A 2014 Genetic Algorithm with Fast Greedy Heuristic for Clustering and Location Problems *Informatica* **38** 229-40
[4] Farahani R and Hekmatfar M 2009 *Facility location: Concepts, models, algorithms and case studies* (Heidelberg: Springer-Verlag)
[5] Kazakovtsev, L, Stupina A 2015 Fast genetic algorithm with greedy heuristic for p-median and k-means problems *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops* 35-9
[6] Kazakovtsev L 2016 *The greedy heuristics method for systems of automatic grouping of objects* (Krasnoyarsk)
[7] Orlov V, Kazakovtsev L, Rozhnov I, Popov N and Fedosov V 2018 Variable neighbourhood search algorithm for k-means clustering *IOP Conf. Series: Mater. Scie. Eng.* **450** 022035, DOI:10.1088/1757-899X/450/2/022035
[8] Zechner M, Granitzer M 2009 Accelerating K-Means on the Graphics Processor via CUDA DOI:10.1109/INTENSIVE.2009.19
[9] Luebke D and Humphreys G 2007 How gpus work *Computer* **40**(2) 96–100
[10] Lutz C, Breß S, Zeuch S, Markl V and Rabl T 2018 Efficient k-Means on GPUs *DaMoN'18* June 11 Houston, TX, USA
[11] UCI Machine Learning Repository Available from http://archive.ics.uci.edu/ml
[12] Clustering basic benchmark Available from http://cs.joensuu.fi/sipu/datasets