Fast hierarchical clustering of multispectral images and its implementation on NVIDIA GPU

S A Rylov and I A Pestunov

1Institute of Computational Technologies SB RAS, acad. Lavrentiev ave. 6, Novosibirsk, Russia, 630090

e-mail: rylovs@mail.ru

Abstract. The present work presents a parallel implementation of the hierarchical grid-based clustering algorithm HCA on GPU using CUDA technology that substantially reduced the processing time of multispectral images. Provided experimental results on model datasets and images confirm the efficiency of the HCA clustering algorithm and its parallel implementation. The computation speedup was shown to be 10–15x compared to 4 core CPU.

1. Introduction

Clustering of large datasets is urgent in many applied problems, while a priori information about the probabilistic characteristics of classes is usually absent. Segmentation of multispectral satellite images, which often contain tens millions of pixels, is one of them [1]. In these conditions, it is desirable to use algorithms that are nonparametric and at the same time computationally efficient. Grid-based clustering methods [2] possess both of these properties. They process the elements of the grid structure, which is formed by dividing feature space into a finite number of cells. Previously proposed by the authors hierarchical grid-based clustering algorithm HCA [3] at low computational costs can distinguish clusters of complex shape and obtains hierarchical clustering structure. Moreover, in contrast to the well-known hierarchical algorithms [4], it allows to separate clusters intersecting in the feature space.

Today, most personal computers are equipped with graphics cards (GPU), the performance of which has grown dramatically in the last years. Consequently, general-purpose computing on graphics processing units (GPGPU) technologies are actively developing for solving time-consuming non graphics tasks [5, 6].

The present work presents a parallel implementation of the hierarchical grid-based clustering algorithm HCA on GPU using CUDA technology that substantially reduced the processing time of multispectral images. Provided experimental studies on model datasets and images confirm the efficiency of the considered algorithm and its parallel implementation. The achieved performance allows to cluster high-resolution video stream in real time mode.
2. Grid-based hierarchical clustering algorithm HCA

This section briefly describes the nonparametric clustering algorithm HCA [3].

Let the set of objects X consist of d-dimensional vectors lying in the feature space $\mathbb{R}^d$: $X = \{ x_i = (x_{i1}, ..., x_{id}) \in \mathbb{R}^d, i = 1, ..., N \}$, and bounded by a hyper-rectangle $\Omega = [l^1, r^1] \times \cdots \times [l^d, r^d]$, where $l^i = \min x_{i1}$, $r^i = \max x_{i1}$, $x_i \in X$. Grid structure is formed by dividing $\Omega$ with hyperplanes $x' = (r' - l') \cdot i / m + l'$, $i = 0, ..., m$, where $m$ is the number of partitions in each dimension. The minimum element of this structure is a cell (closed rectangular hyper-rectangle bounded by hyperplanes). We introduce a common numbering of the cells (sequentially from one layer of cells to another). Cells $B_i$ and $B_j$ (i $\neq$ j) are adjacent if their intersection is not empty. The set of cells adjacent to $B$ will be denoted by $A_B$. The density $D_B$ of the cell $B$ is defined as the number of elements from the set $X$ belonging to the cell $B$. We assume that the cell $B$ is nonempty if $D_B > 0$.

The nonempty cell $B_i$ is directly connected to the nonempty cell $B_j$ if $B_i \rightarrow B_j$ is the cell with the maximum number that satisfies the conditions $B_i \in A_{B_j}$ and $\forall B \in A_{B_j} \cap B_i \geq D_{B_i}$. The nonempty adjacent cells $B_i$ and $B_j$ are directly connected ($B_i \leftrightarrow B_j$) if $B_i \rightarrow B_j$ or $B_j \rightarrow B_i$. The nonempty cells $B_i$ and $B_j$ are connected ($B_i \sim B_j$) if there exist $k_1, ..., k_l$ such that $k_i = i$, $k_l = j$ and for all $p = 1, ..., l-1$ we have $B_{k_p} \rightarrow B_{k_{p+1}}$. The introduced connectedness relation leads to the natural partition of the set of nonempty cells into the connected components $\{G_1, ..., G_3\}$. The connected component is defined as the maximum set of pairwise connected cells. Representative cell $Y(G)$ of the component $G$ is defined as a cell with the maximum number that satisfies the condition $Y(G) = \arg \max_{B \in G} D_B$.

In the proposed algorithm, we take the connected components as the basic elements of the hierarchy, and not the elements of initial data. Due to the use of a grid structure, the number of obtained connected components is relatively small in comparison with the number of data elements $N$, therefore the construction of a hierarchy does not require high computational efforts.

The components $G_{i1}$ and $G_{i2}$ are considered adjacent if there are adjacent cells $B_i$ and $B_j$ such that $B_i \in G_{i1}$ and $B_j \in G_{i2}$. To construct a hierarchy between components, we define the distance between adjacent components $G_{i1}$ and $G_{i2}$ by the formula
\[ h_{ij} = \min_{P_{ij} \in R_{ij}} \left[ 1 - \min_{B_{k_1} \in P_{ij}} D_{B_{k_1}} / \min(D_{Y_{i1}}, D_{Y_{i2}}) \right], \]
where $R_{ij} = \{ P_{ij} \}$ is a set of all possible paths between representative cells $Y_{i1} = Y(G_{i1})$ and $Y_{i2} = Y(G_{i2})$, $P_{ij} = \{ Y_{i1} = B_{kp}, ..., B_{ki}, B_{ki+1}, ..., B_{ij} = Y_{i2} \}$ such that for all $t = 1, ..., l-1: 1) B_{k_t} \in G_{i1} \cup G_{i2}; 2) B_{ki}, B_{ki+1}$ are adjacent cells.

We construct the matrix of distances between the connected components $\{h_{ij}\}$ on the distances between adjacent connected components $\{h_{ij}\}$ as follows. Let $\Theta_{ij} = \{ Q_{ij} \}$ be the set of all chains of the connected components $Q_{ij} = \{ G_{i1} = G_{k_1}, ..., G_{k_l}, G_{k_l+1}, ..., G_{ij} = G_{i2} \}$ such that for all $t = 1, ..., l-1$, the components $G_{k_t}, G_{k_t+1}$ are adjacent. Then the distance between arbitrary connected components $G_{i1}$ and $G_{i2}$ is defined as
\[ \hat{h}_{ij} = \min_{Q_{ij} \in \Theta_{ij}} \left[ \max_{1 \leq k \leq l} h_{k,k+1} \right]. \]
If the set $\Theta_{ij}$ is empty, we assume that $\hat{h}_{ij} = 1$.

The introduced distance based on the estimation of the data distribution density eliminates the problem of overlapping classes, which is inherent to hierarchical methods [4]. The advantage of the introduced distance $\hat{h}_{ij}$ is that it has ultrametric property [7], i.e. it is a metric that satisfies the strong triangle inequality: $\hat{h}_{ij} \leq \min (\hat{h}_{ik}, \hat{h}_{jk}) \forall i, j, k$. It is known that there exists one-to-one correspondence between distance matrices with ultrametric property and dendrograms [8]. Therefore, such matrix can be used as a descriptor for the hierarchical result.

The calculation of the ultrametric $\{\hat{h}_{ij}\}$ from a distance matrix like $\{h_{ij}\}$ is usually referred to as the minimum transitive closure operation [9]. Its implementation often involves the use of time consuming algorithms with computational complexity $O(n^3)$ [8-10] (for the distance matrix of size $n \times n$). But it
was shown by the authors [3] that the minimum transitive closure operation can be performed by applying single linkage algorithm (SLINK) for dendrogram construction, which has computational complexity $O(n^2)$.

The hierarchical grid-based algorithm HCA($m$) can be written as a sequence of the following major steps.

1. Grid structure formation. In this step, for each data element $x_i \in X$, a cell containing this element is determined, and the densities of all cells are calculated.
2. Finding the connected components $\{G_1, \ldots, G_S\}$ and their representative cells $Y(G_1), \ldots, Y(G_S)$.
3. Calculation of the distance matrix $\{h_{ij}\}$ between adjacent connected components in accordance to the above definition.
4. Constructing a dendrogram by applying single linkage method to the distance matrix $\{h_{ij}\}$.

The result of the algorithm is a hierarchical structure built on the set of connected components.

HCA clustering algorithm has a linear computational complexity from the number of data elements $N$ (due to the use of a grid structure). It allows to distinguish clusters of complex shape and to obtain hierarchical clustering structure. Furthermore, it can effectively separate clusters that are intersecting in the feature space. Despite the exponential computational complexity from the data dimension $d$, the algorithm is effective for practical use for the dimensions $d \leq 6$.

3. Parallel implementation of the clustering algorithm HCA on GPU with CUDA

CUDA is a powerful parallel computing platform and API model that allows to use NVIDIA GPUs for general purpose processing. Modern GPUs contain thousands of cores, grouped into blocks under the control of multiprocessors. The cores of one block perform the same set of instructions, but on different pieces of data. Each core contains a small number of registers and has quick access to a limited amount of shared memory within its block (managed cache). In addition, all the threads (executed on separate cores) can access a large amount of global memory, but random access time to it is very slow and can take hundreds of cycles. Synchronization of the threads during the execution is possible only within a block.

In our CUDA implementation of the clustering algorithm, we focused on the NVIDIA Kepler GPU architecture and utilized its features, which are not available in the previous GPU architectures, like fast atomic operations with global memory, shuffle instructions and others. To achieve maximum efficiency, we made separate implementations of the algorithm on GPU for different data dimensions (3 and 4). The initial implementation on CPU was designed to handle an arbitrary dimension (up to 8).

Parallel implementation of the HCA algorithm consists of six steps that are executed sequentially. For each step one GPU compute kernel is launched.

At the first stage, densities of the cells are calculated. This process can be considered as a computation of multidimensional histogram. In our implementation, the elements of the histogram array are filled in the GPU global memory using the atomic increment operation. Whereas common approaches to the histogram calculation on GPU use shared memory, which is not suitable for processing a multidimensional histogram because of the limited size of the shared memory (48 KB).

At the second stage, for each cell an adjacent cell with maximum density is found. In CUDA implementation, different cells are processed by different threads that are looking through adjacent cells. To improve the performance of this step, a number of optimizations were used.

- The use of shared memory. The threads of each block are aligned in a line along Z axis. Each thread looks through the adjacent cells with a fixed coordinate $z$. After that, these intermediate results are exchanged between neighboring threads through shared memory.
- Avoiding re-readings. Each thread sequentially processes a group of elements, moving along $Y$ axis. In this process, already calculated intermediate maxima for the layers with a fixed $y$ coordinate are used, which allows to check only one third of the elements required to determine the maximum value in the new position.

The list of representative cells of the connected components is formed at the second stage.
At the third stage, the connected components are extracted. Each thread processes one cell, successively following the links to the adjacent cells with maximum density until it reaches one of the representative cells. The passed chain of cells is then assigned to this representative cell (corresponding connected component).

At the fourth stage, the distance matrix between adjacent connected components is formed according to the definition. By construction, the lowest density between adjacent components is achieved at their border cells. Therefore, when all the pairs of adjacent cells are processed in parallel if the cells belong to the different components, then the density difference between these components is calculated using the densities of these two cells and corresponding representative cells. After that, the distance between the components is updated in global memory using atomic minimum operation.

At the fifth stage of the algorithm, a single linkage method (SLINK) is applied to the distance matrix between adjacent connected components to construct a dendrogram. For a matrix of the size \(n \times n\), SLINK algorithm makes \((n-1)\) iterations, where two closest elements are united at each iteration. The distance to the new joint element is defined as the minimum of the distances to the currently united elements. The algorithm based on the use of an array of partial minima (containing the values and indices of the minimal elements in each row of the distance matrix) is described in [9]. It can find the minimum matrix element in \(O(n)\) operations. In case of single linkage, the SANN property (same agglomerative nearest neighbor) allows to update a distance matrix and an array of partial minima in \(O(n)\) operations. As a result, \(O(n^2)\) computational complexity of the SLINK algorithm is achieved.

Each subsequent iteration of the SLINK algorithm (the level of the dendrogram) directly depends on the previous one, which makes it difficult to parallelize. The known methods of SLINK parallel implementation divide data into a number of intersecting parts, construct a dendrogram for each part independently, and then combine partial results [11-13]. However, this approach is difficult to implement on GPU, designed for mass similar calculations.

In our CUDA implementation, parallelization was made in the recalculation of distances to the united elements and in the search of the minimum element in the array, which was efficiently implemented using shuffle instructions that allow to exchange register values directly between neighboring threads without explicit synchronization [14]. To avoid problems with synchronization of the thread blocks and information exchange between them, the SLINK algorithm is performed by one block only, and the array of partial minima is stored in the shared memory of the block. In this case, the limited size of the shared memory (48 KB) limits the maximum size of the distance matrix to 6100×6100, which is sufficient, since the number of connected components rarely exceeds 2000 when processing multispectral satellite images.

In [15] there is described a similar approach to a parallel SLINK implementation on GPU based on the use of an array of partial minima. In their work, the authors processed matrices of the size from 4096×4096 to 16384×16384 and obtained the speedup on GPU from 0.8 to 2.1. It should be mentioned that they used an old video card NVIDIA Tesla C870.

The proposed SLINK implementation on GPU outruns the performance of the similar implementation on CPU at \(n\geq500\). The speedup factor for \(n=1000\) achieves 3x and for \(n>3000\) it exceeds 7x. Despite the better performance of CPU implementation for distance matrices of smaller sizes, their processing time on GPU does not exceed 1 millisecond. Therefore, it is reasonable to perform this computation stage on GPU to avoid additional data transfer between CPU and GPU, which would take more time.

At the final stage, data elements are assigned to the clusters, to which the cells containing these elements belong.

4. Experimental results
This section represents the results of experimental studies of the HCA algorithm on model data and images. To evaluate the performance of the considered algorithm, it was compared with other well-known clustering algorithms included in the open software package ELKI [16], like K-means, EM, DBSCAN, OPTICS, DeLiClu, SLINK and Mean shift. Clustering accuracy was determined as
the percentage of correctly classified elements. To calculate it, each class from the reference partition is associated with a cluster (one or none) containing the largest number of elements from this class.

Figure 1a represents the two-dimensional model dataset consisting of eight classes with a normal distribution [17]. Classes are grouped into three isolated groups, one of which has strong intersections between classes, which makes it particularly difficult to separate them. The HCA algorithm managed to extract all eight classes (figure 1b) with 98.0% accuracy. Moreover, the hierarchical data structure was revealed (figure 1b-d).

K-means and EM clustering algorithms, which were designed to separate classes with normal distribution, were able to extract all 8 classes, but only if the cluster centers were initialized successfully. The best result (98.3% accuracy) was achieved by the nonparametric Mean shift density-based algorithm, which is computationally expensive. At the same time, density-based algorithms (DBSCAN, OPTICS, DeLiClu), as well as hierarchical methods (single linkage, complete linkage, weighted average linkage), failed to separate the intersecting group of three classes.

Figure 1. Model dataset (a) and the clustering result of the HCA algorithm at different hierarchy levels (b-d).

The second model dataset (figure 2a) contains five classes that are different in shape, size and density, including rings and normally distributed classes [17]. HCA algorithm successfully separated all classes with 99.4% accuracy (figure 2b). On the other hand, none of the clustering algorithms from ELKI package could correctly extract all five classes. Mean shift algorithm cannot extract multimode clusters in the form of rings in principle. The best results were obtained by the OPTICS algorithm that reached 79.7% accuracy (figure 2c), and SLINK algorithm with 72.7% accuracy (figure 2d).

Computational efficiency of the HCA algorithm allows to process multispectral satellite images. Figure 3 shows the results of WorldView-2 satellite image clustering at different hierarchy levels. Four spectral bands were used: 1, 2, 4, 7. Image size is 2048×2048 pixels.

The given table shows the time performance of the proposed CUDA implementation of the HCA algorithm on GPU (GeForce GXT 770) compared with execution on one and four cores of the CPU (Intel Core i7 960, 3.2 GHz) on the images of different sizes. The parallel version of the algorithm for
CPU was implemented with OpenMP. Grid parameter $m$, which affects the segmentation detail level, was set to 32 (common value for image clustering). The results showed that the obtained speedup on GPU is approximately 15x for RGB images and 10x for four-band satellite images.

![Figure 2. Model dataset (a) and the clustering results obtained by HCA (b), OPTICS (c) and SLINK (d) algorithms.](image1)

![Figure 3. WorldView-2 satellite image (RGB-composite, bands 5, 3, 2) and the clustering result of the HCA algorithm at three different hierarchy levels.](image2)
Table 1. HCA algorithm time performance on the images (in milliseconds).

| Image size (megapixels) | Color images (RGB) | Satellite images (4 bands) |
|-------------------------|--------------------|----------------------------|
|                         | 5                  | 12.4                       |
|                         | 115                | 24                         |
|                         | 100                |                            |
| CPU, 1 core             | 47                 | 95                         |
|                         | 722                | 82                         |
|                         | 290                | 954                        |
| CPU, 4 cores            | 36                 | 70                         |
|                         | 718                | 62                         |
|                         | 240                | 767                        |
| GPU                     | 3                  | 5                          |
|                         | 37                 | 7                          |
|                         | 23                 | 64                         |
| Speedup                 | 12.4               | 14.3                       |
|                         | 19.4               | 8.5                        |
|                         | 10.4               | 12                         |

5. Conclusion
The parallel implementation of the hierarchical grid-based clustering algorithm HCA on NVIDIA graphics processors using CUDA technology is presented. Experimental results on model datasets and images have shown that the algorithm provides qualitative clustering results, and the proposed GPU implementation processes multispectral images 10–15 times faster than on CPU. The achieved performance allows to cluster high-resolution video stream in real time mode.
In future, we are planning to make CUDA implementation of the new clustering algorithm HCA-MS [18], which is based on the HCA algorithm and adapts computationally time-consuming mean shift procedure to refine clusters borders. We expect that the HCA-MS processing time of satellite images will be reduced from several minutes to several seconds.

6. References
[1] Myasnikov E V 2017 Hyperspectral image segmentation using dimensionality reduction and classical segmentation approaches Computer Optics 41(4) 564-572 DOI: 10.18287/2412-6179-2017-41-4-564-572
[2] Ilango M R and Mohan V 2010 A survey of grid based clustering algorithms Intern. J. Eng. Sci. and Technology 2(8) 3441-3446
[3] Pestunov I A, Rylov S A and Berikov V B 2015 Hierarchical clustering algorithms for segmentation of multispectral images Optoelectronics, Instrumentation and Data Processing 51(4) 329-338
[4] Lu Y and Wan Y 2013 Pha: a fast potential-based hierarchical agglomerative clustering method Pattern Recognition 46(5) 1227-1239
[5] Choquette J, Giroux O and Foley D 2018 Volta: performance and programmability IEEE Micro 38(2) 42-52
[6] Fursov V A, Goshin Y V and Kotov A P 2016 The hybrid cpu/gpu implementation of the computational procedure for digital terrain models generation from satellite images Computer Optics 40(5) 721-728 DOI: 10.18287/2412-6179-2016-40-5-721-728
[7] Leclerc B 1981 Description combinatoire des ultramétriques Math. Sci. Humaines 127(73) 5-37
[8] Mirzaei A and Rahmati M 2010 A novel hierarchical-clustering-combination scheme based on fuzzy-similarity relations IEEE Tr. Fuzzy Syst 18(1) 27-39
[9] Zheng L, Li T and Ding C 2010 Hierarchical ensemble clustering Proc. IEEE Int. Conf. on Data Mining 1199-1204
[10] Skiena S 2008 The Algorithm Design Manual (London: Springer)
[11] Olson CI F 1995 Parallel algorithms for hierarchical clustering Parallel Computing 21(8) 1313-1325
[12] Jin C, Patwary M A, Agrawal A, Hendrix W, Liao W and Choudhary A 2013 Disc: a distributed single-linkage hierarchical clustering algorithm using mapreduce Proc. 4th Int. SC Workshop on Data Intensive Computing in the Clouds 23 27-42
[13] Hendrix W, Patwary M A, Agrawal A, Liao W and Choudhary A 2012 Parallel hierarchical clustering on shared memory platforms 19th Annual Int. Conf. on High Performance Computing
Acknowledgements

This work was supported by SB RAS complex program of fundamental research «Interdisciplinary integration research» in 2018-2020 (project 37).