S©MR

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ
Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru
ISSN 1813-3304

Vol. 22, No. 2, pp. C104–C120 (2025)
https://doi.org/10.33048/semi.2025.22.C08

УДК 519.6
MSC 68T05

NPU-ACCELERATED K-MEANS AND K-MEDOIDS CLUSTERING ALGORITHMS

I.L. VASILYEV, T.V. GRUZDEVA, D.A. BOYARKIN, A.V. USHAKOV

*Communicated by the Programme Committee
of the XXIV International Conference
"Mathematical Optimization Theory and Operations Research" (MOTOR 2025)*

Abstract: Clustering is one of the basic tasks in unsupervised machine learning and data mining. The clustering process consists of partitioning data items from a dataset so that similar items are grouped into non-overlapping clusters. The so-called center-based clustering approach remains one of the most popular ones to formalize the clustering problem. In this case, partitioning data items into clusters performed by identifying the fixed number of so-called representatives (centers) of clusters. Then, clusters are formed by assigning each data item to the closest center.

The k-means (minimum sum-of-squares) and k-medoids (minimum sum-of-stars) problems are the most well-known center-based clustering problems and perhaps the most popular clustering models at all. Despite their simplicity and popularity, k-medoids and k-means algorithms are both suitable for clustering relatively small datasets due to their high-computational complexity. The most time-consuming operation in these algorithms is the calculation of distances (measures of dissimilarity) between data items. To make

Vasilyev,I.L., Gruzdeva, T.V., Boyarkin, D.A., Ushakov, A.V., NPU-accelerated k-means and k-medoids clustering algorithms.

The research is funded by the Ministry of Education and Science of the .
Russian Federation (project 121041300065-9).
Received May, 31, 2025, Published November, 24, 2025.

the complex and consuming mathematical computations of such type, recently the Neural Processing Unit (NPU) was designed and optimized as a highly specialized processor to carry out AI challengers.

We proposed and developed special variants of well-known k-medoids and k-means algorithms accelerated by NPU and parallel implementation. The algorithms realized outperform not only the CPU but also the GPU in terms of performance, effectiveness and efficiency, which is proved by computational simulations.

Keywords: clustering, neural processing unit, k-means, k-medoids, parallel computing, machine learning.

1 Introduction

Clustering is one of the basic tasks in unsupervised machine learning and data mining. The clustering process consists of partitioning data elements from a dataset so that similar elements are grouped into non-overlapping clusters. The so-called center-based clustering approach remains one of the most popular ones to formalize the clustering problem. In this case, partitioning data items into clusters is performed by identifying a fixed number of so-called cluster representatives or cluster centers. Then, clusters are obtained by assigning each data item to the closest (most similar) center. The number of clusters is equal to the number of centers, so it is supposed to be known in advance. Given a distance (or dissimilarity measure), any center-based clustering problem can be represented as a non-convex optimization problem, where the goal is to identify centers in such a way that the total sum of dissimilarities between data items and the closest centers is minimized.

The k-means (minimum sum-of-squares) and k-medoids (minimum sum-of-stars) problems are the most well-known center-based clustering problems and perhaps the most popular clustering models at all. Note that they can be considered as facility location problems, e.g., k-medoids problem is the well-known p-median facility location problem, and k-means model is a particular case of the multi-source Weber problem [1, 2].

Given a finite set of m data items, represented as vectors $a^j \in I\!R^n$, $j \in J = \{1, \dots, m\}$, the k-means problem is to find k cluster centers (centroids) $x_c \in I\!R^n$, such that the total sum of squared Euclidean distances between data items and their closest centers is minimized:

$$\min_{C \subset I\!R^n} \left\{ \sum_{j=1}^{m} \min_{x_c \in C} \|a^j - x_c\|^2, \ |C| = k \right\}, \tag{1}$$

The problem is NP-hard even in the plane for arbitrary number of clusters k [3]. It is also NP-hard both in general dimension even for $k = 2$ [4] and when the dimension n is a part of the input, whereas the number of clusters k is not [5].

The k-medoids problem is very similar to the aforementioned k-means problem. It is to select k medoids (or exemplars) x_m^1, x_m^2, $\dots, x_m^k$ from within the finite set $\mathcal{A}$ so as the overall sum of distances $d(\cdot, \cdot)$ (or other measures of dissimilarity) between data items and their closest medoids is minimized:

$$\min_{C \subset \mathcal{A}} \Big\{ \sum_{j=1}^{m} \min_{x_m \in C} d(a^j, x_m),\ |C| = k \Big\}. \tag{2}$$

In contrast to the k-means problem, cluster representatives are not cluster means but so-called medoids selected among data items. The geometric (Euclidean and Manhattan) and graph versions of the k-medoids problem are known to be NP-hard [6, 7].

Both clustering problems have widely been studied in the literature. One of the oldest and most popular clustering algorithms in practice, k-means (or Lloyd's algorithm [8]), is a local search (alternative) heuristic to the minimum-sum-of-squares clustering problem. Besides k-means there are a number of other well-known algorithms, e.g. McQueen's algorithm and the Hartigan-Wong algorithm. In spite of k-means, the latter is an exchange-type heuristic, i.e. it tries to improve the objective value by exchanging data items between clusters.

The k-medoids problem is also known as the discrete p-median problem is one of the classical facility location problems. The most well-known clustering algorithm based on the k-medoids problem is PAM, which is a two stage heuristic including a greedy step to finds initial medoids followed by the best-improvement exchange heuristic over the SWAP neighborhood. The second stage is actually similar to the Hartigan-Wong algorithm. There is also an alternative (location-allocation) heuristic for the k-medoids problem, known as k-medoids or Cooper's algorithm, that follows the exactly same steps as k-means but finds medoids rather than means as cluster representative. Note that the main computational difference between k-means and k-medoids is the step of refining cluster centers. In case of k-means, for each cluster, it can be done in linear time resulting in the $\mathcal{O}(mnk)$ time complexity of a single iteration. For k-medoids, this step is not so simple and requires quadratic time in general case.

Despite their simplicity and popularity, k-medoids and k-means are both suitable for clustering relatively small datasets due to their high-computational complexity. For example, for MINIST8M test dataset, where $m = 8.1M$, $n = 784$, $k = 1000$, one iteration of k-means runs several hours [9]. However, real-worlds data may consists of much larger number of data items and possible clusters. Each data item may be defined by thousands of features, especially in applications involving clustering images or videos. Thus, it is extremely challenging to cluster such huge datasets with the conventional clustering algorithms.

There are several strategies of how make k-means and k-medoids applicable for large-scale data, e.g. random sampling (e.g. CLARA), binary hashing [9, 10], and fast procedures of distance calculations [11, 12]. Note that as the most time consuming operations, finding the nearest cluster center and recomputing cluster centers can be performed concurrently for each data item, there is a large strand of literature devoted the development of shared-memory and distributed parallel implementations of clustering algorithms [13]. A huge number of computations per one iteration make GPUs especially suitable for speeding up the k-means and k-medoids algorithms making GPU-implementations the most effective and widespread. Effective GPU-accelerated implementations often requires the development of very efficient execution strategy. Thus, there are dozens of GPU-implementations for both clustering algorithms that differ in what operations are performed on device or CPU, how to effectively implement data transfer, how to optimize basic algebraic operations, etc. [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

The Neural Processing Unit (NPU) is highly specialized processor designed explicitly to carry out Artificial Intelligence (AI) challengers. It is optimized to make the complex and consuming mathematical computations required in machine learning algorithms and artificial neural networks. Before the development of the NPU, all AI-related problems are solved by either the CPU or GPU (graphics processing unit) which are still effective and efficient, but not in AI area. NPUs excel in parallel processing capabilities, allowing to perform multiple operations simultaneously. This is crucial for solving problems that involve large datasets and complex computations. Second, focusing on energy-efficient designs, NPUs consume less power while delivering high performance, which make them ideally for edge devices. Furthermore, NPUs are tailored to accelerate specific AI tasks, e.g. deep learning, resulting in faster training and inference times.

In this paper we propose the first NPU-accelerated implementations of the well-known k-means and k-medoids algorithms. As was noted above the most time-consuming operation is the calculation of distances (measures of dissimilarity) between data items and cluster centers. Fortunately, these distances can be computed using special constructed matrices (tensors) and operations on them, which are performed very efficiently on the NPUs.

2 Algorithms

The k-means algorithm consists in two principal steps: assigning data items to the current closest (by the squared Euclidean distance) cluster centers followed by recompute the centers as means of data items assigned to the same clusters. The algorithm terminates when there are no changes in cluster assignments of a sufficiently large number of data items (see Alg. 1).

Algorithm 1 k-means algorithm

```
1: Choose initial cluster centers $x_c^i \in \mathbb{R}^n$, $i \in I$ (often $x_c^i \in \{a^1 \dots, a^m\}$).
2: while clusters $C_i$ change do
3:     Assign data items $a^j$, $j \in J$, to the closest cluster center $i^*$ by computing $i^* \leftarrow \operatorname*{argmin}_{i \in I} \sum_{j=1}^{m} \|a^j - x_c^i\|^2$. Form clusters $C_i$ of data items assigned to the center $x_c^i$.
4:     For each cluster $C_i$, $i \in I$, compute new $x_c^i$ as the center of gravity (mean) of all the data items from $C_i$: $x_c^i \leftarrow \frac{1}{|C_i|} \sum_{j: a^j \in C_i} a^j$.
5: end while
```

The k-medoids algorithm [27] follows the same steps as k-means except for step 4. Instead of finding means, which can be done in linear time, k-meodis requires finding a medoid such that the distance between the data items in a cluster and their closet medoid is minimized. If on Step 4 of Alg. 1, for each cluster C_i, $i \in I$, we search for $j_i \leftarrow \operatorname*{argmin}_{j: a^j \in C_i} \sum_{\substack{j': \\ a^{j\prime} \in C_i}} \|a^j - a^{j\prime}\|^2$ and get $x_m^i \leftarrow a^{j_i}$, then we obtain a k-medoids algorithm. Obviously, it can be done in quadratic time, which makes this step much more computationally expensive in comparison with k-means.

As k-medoids operates on the most centrally located objects in clusters, it is less sensitive to outliers in comparison with the k-means clustering. Moreover, k-medoids is flexible in the definition of dissimilarities between data items, e.g. dissimilarity measure may be not a proper distance metric.

As k-means and k-medoids are local search heuristics, their performance are highly dependent on the strategy of selecting initial cluster centers.

3 NPU-based implementation

Here we present our implementations of the clustering algorithms. The main feature of NPUs that makes it highly effective in AI tasks is effective matrix computations. The idea of our NPU-based implementations of the clustering algorithms is to represent the location and allocation steps in the form of matrix-matrix computations. The latter allows us to achieve the full utilization of NPUs computing power.

3.1. Matrix representation and calculations. Let us first consider k-means. To form the matrix of distances between data items and cluster centers, $D \in \mathbb{R}^{m \times k}$, let us introduce the data matrix $A \in \mathbb{R}^{m \times n}$ where each row is the feature vector of a data item, and cluster center matrix $X \in \mathbb{R}^{k \times n}$ with rows corresponding to the coordinates of cluster centers:

$$A = (a^1, a^2, \dots, a^m)^\top, \quad X = (x^1, x^2, \dots, x^k)^\top.$$

Next, let us calculate squired Euclidean norm of each row and construct the following auxiliary matrices $A_{\|\cdot\|}, X_{\|\cdot\|} \in I\!R^{m\times k}$:

$$A_{\|\cdot\|} = \left(\underbrace{\begin{matrix} \| a^1 \|^2 & \dots & \| a^1 \|^2 \\ \| a^2 \|^2 & \dots & \| a^2 \|^2 \\ \dots & \dots & \dots \\ \| a^m \|^2 & \dots & \| a^m \|^2 \end{matrix}}_{k} \right), X_{\|\cdot\|} = \left(\left. \begin{matrix} \| x^1 \|^2 & \dots & \| x^k \|^2 \\ \| x^1 \|^2 & \dots & \| x^k \|^2 \\ \dots & \dots & \dots \\ \| x^1 \|^2 & \dots & \| x^k \|^2 \end{matrix} \right\} m \right),$$

Note that each row of the matrix $A_{\|\cdot\|}$ consists of k copies of norms of the corresponding feature vector. Each column of the matrix $X_{\|\cdot\|}$ consists of m copies of the corresponding cluster center.

Thus, we can calculate the distance matrix

$$D = A_{\|\cdot\|} + X_{\|\cdot\|} - 2AX^T,$$

with elements

$$d_{ij} = \| a^j - x^i \|^2 = \| a^j \|^2 + \| x^i \|^2 - 2\langle x^i, a^j \rangle,\ i = 1, \dots k,\ j = 1, \dots m.$$

As for the k-medoids clustering, instead of one matrix, we compute the matrices $A_i \in I\!R^{m_i \times n}$ and $\widehat{A}_{\|\cdot\|} \in I\!R^{m_i \times m_i}$ for each cluster i with m_j data items:

$$A_i = \begin{pmatrix} a^{i1} \\ a^{i2} \\ \dots \\ a^{m_i} \end{pmatrix}, \quad \widehat{A}_{\|\cdot\|} = \left(\underbrace{\begin{matrix} \| a^{i1} \|^2 & \dots & \| a^{i1} \|^2 \\ \| a^{i2} \|^2 & \dots & \| a^{i2} \|^2 \\ \dots & \dots & \dots \\ \| a^{m_i} \|^2 & \dots & \| a^{m_i} \|^2 \end{matrix}}_{m_i} \right),$$

Thus, for each cluster i, we can calculate the matrix of pairwise distances $D_i \in I\!R^{m_i \times m_i}$:

$$D_i = \widehat{A}_{\|\cdot\|} + \widehat{A}^T_{\|\cdot\|} - 2A_iA_i^T$$

with elements

$$d_{ij} = \| a^i - a^j \|^2 = \| a^i \|^2 + \| a^j \|^2 - 2\langle a^i, a^j \rangle, \quad i = 1, \dots m_i,\ j = 1, \dots m_i.$$

Obviously, the pairwise distances between data items may be pre-computed and given to k-medoids as input. However, the large amount of memory required for storing these may be a serious bottleneck for the k-medoids algorithm, as their number is the square of the number of data items.

3.2. Huawei Ascend. The implementation of k-means and k-medoids algorithms was carried out on the Ascend NPU devices using the Compute Architecture for Neural Networks (CANN). CANN is a device management framework that includes various APIs, particularly the AscendCL. AscendCL provides a collection of C language APIs to develop DNN (deep neural network) apps on the Ascend platform. AscendCL offers APIs for resource management, memory management, model loading and execution, operator loading and execution and other capabilities on the Ascend CANN platform with Ascend hardware. In a nutshell, AscendCL is a unified API framework used to invoke every resource.

The core of Ascend's hardware computing capabilities is the so-called AI Processor, which efficiently performs matrix computations required in training and inference of neural networks, general computations, perform execution control, operations between scalars and vectors, etc. While these advantages are critical to performing DNN-related calculations, the strength of matrix calculations can be used in other algorithms that involve intensive matrix operations. In our implementation of the k-means and k-medoids algorithms, we focus on the attempt to exploit these advantages, developing matrix representations of location-allocation steps in the algorithms.

3.3. Ascend memory management. The programs written using AscendCL are executed on Devices. A Device refers to an Ascend AI Processor-powered hardware that connects to the Host over the PCIe interface, providing the Host with the NN computing capability. A Host is a computer on which one or more Devices are installed. The Device has its own installed RAM, which conceptually makes it similar to the GPU. In the case of using several Devices, their memory is independent of each other. Memory sharing between devices is not supported. The same can be said about the interaction between Device and Host.

Thus, performing any operations on Device means copying any data from the Host memory to the NPU Device for subsequent processing and copying the result back. This can be a costly procedure. On the other hand, limited installed memory on NPU Devices may require batch transfer of data for processing. These factors must be taken into account when developing algorithms.

3.4. NPU-accelerated k-means algorithm. The implementation of k-means algorithm for NPU is shown in Figure 1, where the following notations are used:

- $data$ – $m \times n$ data points (host-side, C array);
- max_iter – manually implemented iterations limit;
- $means$ – $k \times n$ center matrix (host-side, C array);
- A – $m \times n$ data matrix (device-side, tensor);
- X – $n \times k$ center matrix (device-side, tensor);
- P_X – $n \times k$ center matrix from the previous iteration (host-side, tensor);
- PP_X–$n \times k$ center matrix from the iteration before previous (host-side, tensor);
- $A_{\|\cdot\|}$ – $m \times k$ matrix with Euclidean norms of data items (host-side, tensor);
- $X_{\|\cdot\|}$ – $m \times k$ matrix with Euclidean norms of cluster centers (host-side, tensor);
- L – $m \times 1$ point-to-cluster assignment labels (host-side, tensor).

It is also worth mentioning that in terms of AscendCL, the concept of tensor is used. It is viewed as a multidimensional matrix. The k-means and k-medoids algorithms use ordinary two-dimensional matrices, therefore, a tensor should be understood here as a matrix.

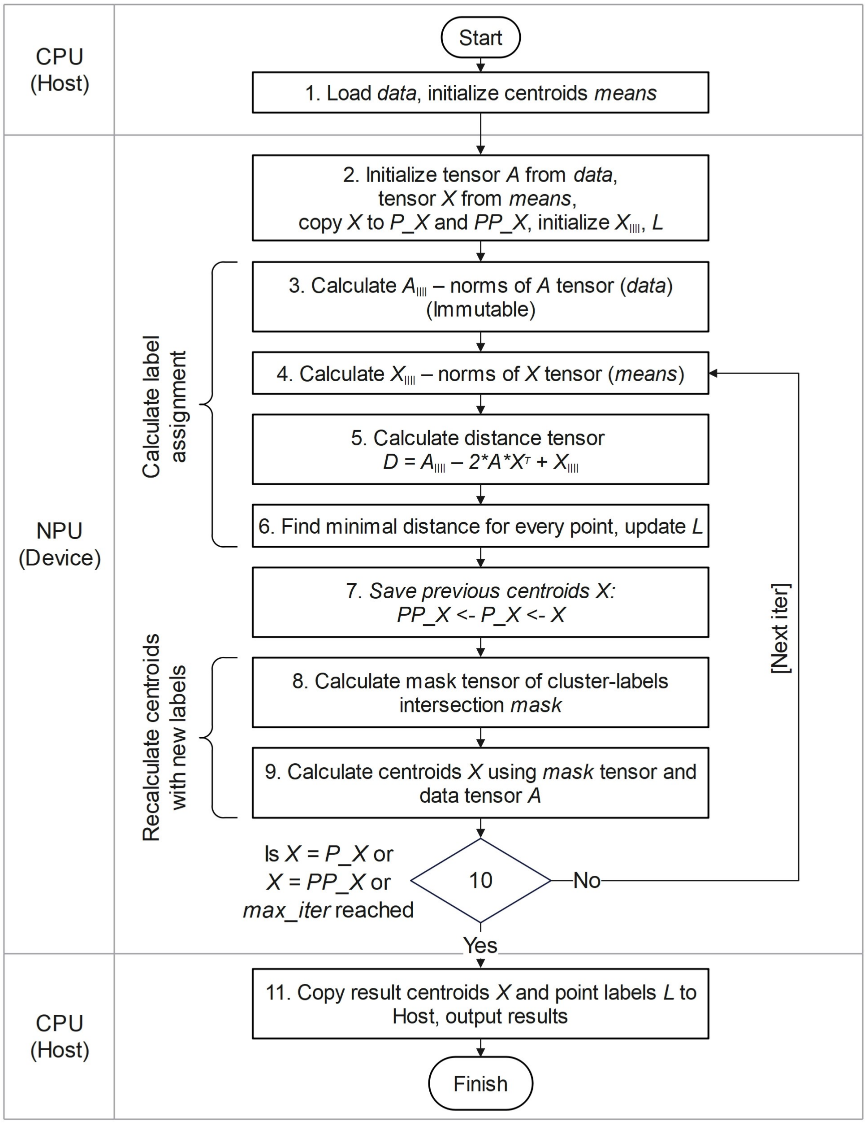
CPU (Host)
Start
1. Load *data*, initialize centroids *means*
NPU (Device)
2. Initialize tensor *A* from *data*, tensor *X* from *means*, copy *X* to *P_X* and *PP_X*, initialize $X_{\|\|}$, *L*
Calculate label assignment
3. Calculate $A_{\|\|}$ – norms of *A* tensor (*data*) (Immutable)
4. Calculate $X_{\|\|}$ – norms of *X* tensor (*means*)
5. Calculate distance tensor $D = A_{\|\|} – 2*A*X^T + X_{\|\|}$
6. Find minimal distance for every point, update *L*
7. *Save previous centroids X: PP_X <- P_X <- X*
Recalculate centroids with new labels
8. Calculate mask tensor of cluster-labels intersection *mask*
9. Calculate centroids *X* using *mask* tensor and data tensor *A*
[Next iter]
Is *X* = *P_X* or *X* = *PP_X* or *max_iter* reached
10
No
Yes
CPU (Host)
11. Copy result centroids *X* and point labels *L* to Host, output results
Finish

FIG. 1. NPU-accelerated k-means algorithm scheme

The algorithm uses AscendCL single operators and works as follows:

(1) Loading initial data matrix $data$ and initializing $means$ using the Lloyd algorithm;

(2) Transmitting the $data$ and $means$ to the memory of the Ascend NPU Device. The result is the tensors A and X. To reduce the number of

further transformations, X is initialized in transposed form:

$data(Carray) \to CreateAclTensor \to A(aclTensor)m \times n;$
$means(Carray) \to CreateAclTensor \to Y(aclTensor)n \times k.$

Memory is also reserved in the device for matrices: P_X, PP_X (centroid values at the previous iteration and the iteration before last), $X_{\|\cdot\|}$ (norms of the X matrix), L (pointer matrix). All further operations starting from this step are performed in the memory of the NPU Device and do not affect the computer's RAM, this saves resources on data transfer between Host and Device.

(3) The calculation of the tensor $A_{\|\cdot\|}$ is performed. Matrix $A_{\|\cdot\|}$ does not change during the calculation process, therefore it is calculated before starting the iterative process only once. AscendCL provides a ready-made aclnnNorm operator to calculate the norm. However, for our problem, the result of its execution also needs to be squared, and the matrix dimension has been changed using the repetition operator k times:
$A\{m \times n\} \to aclnnNorm \to out\_temp\{M \times 1\} \to$
$aclnnInplacePowTensorScalar(2) \to out\_temp\{M \times 1\} \to$
$aclnnRepeatInterleaveIntWithDim(k) \to A_{\|\cdot\|}\{m \times k\}.$

Changing the dimension by copying columns is necessary to perform further matrix operations.

(4) The calculation of the matrix $X_{\|\cdot\|}$ is identical to calculation of $A_{\|\cdot\|}$ (step 3), although it is performed for columns, not rows:
$X\{n \times k\} \to aclnnNorm \to out\_temp\{1 \times k\} \to$
$aclnnInplacePowTensorScalar(2) \to out\_temp\{1 \times k\} \to$
$aclnnRepeatInterleaveIntWithDim(m) \to X_{\|\cdot\|}\{m \times k\}.$

(5) To calculating the distance matrix D, AscendCL provides two single operators – *aclnnAdd* and *aclnnInplaceAddmm*. The first one is a regular summation, the second performs multiplication with a scalar and summation in one operation:
$A_{\|\cdot\|}\{m \times k\}, X_{\|\cdot\|}\{m \times k\} \to aclnnAdd \to res\_temp\{m \times k\};$
$res\_temp\{m \times k\}, A\{m \times n\}, X\{n \times k\} \to$
$aclnnInplaceAddmm(-2) \to D\{m \times k\}.$

(6) Calculating the minimum distance from cluster's centers to each point and obtain a vector L of pointers. To complete this step, one operator is enough:
$D\{m \times k\} \to aclnnArgMinGetWorkspaceSize \to L\{M \times 1\}.$

(7) Saving the values of previous centroids:
$P\_X\{n \times k\} \to aclnnInplaceCopy \to PP\_X\{n \times k\};$
$X\{n \times k\} \to aclnnInplaceCopy \to P\_X\{n \times k\}.$

(8) Obtaining a binary matrix-mask reflecting the belonging of the points of matrix A to the centroids X. To obtain it, the following operations are performed:

(a) Transpose matrix L and change its dimension using row repetition:
$L\{m \times 1\} \to aclnnPermute \to L^T\{1 \times m\} \to$
$aclnnRepeatInterleaveIntWithDim(k) \to L^T\_R\{k \times m\}.$

(b) Creating a matrix of centroid numbers from array of cluster indexes:
$Clusters(Carray) \to CreateAclTensor \to$
$clustersNum\_temp\{k \times 1\} \to$
$aclnnRepeatInterleaveIntWithDim(m) \to$
$clustersNumR\_temp\{k \times m\}.$

(c) Logical correlation of the contents of the pointer matrix and the matrix of centroid indexes allows us to obtain the required $mask$ matrix:
$L^T\_R\{k \times m\},\ clustersNumR\_temp\{k \times m\} \to$
$aclnnEqTensor \to mask\_temp\{k \times m\}.$

As a result we get a matrix of booleans. To use it in matrix operations below, conversion to float is needed:
$mask\_temp\{k \times m\} \to aclnnCast(float) \to$
$mask\_temp\{k \times m\}.$

(9) Calculating new centroids and updating the matrix X need performing the following steps:

(a) Coordinate sums are calculated for all points included in the cluster:
$mask\_temp\{k \times m\},\ A\{m \times n\} \to$
$aclnnMatmul \to sum\_temp\{k \times n\}.$

(b) Calculation of the number of points included in each cluster:
$mask\_temp\{k \times m\} \to aclnnReduceSum \to$
$count\_temp\{k \times 1\} \to$
$aclnnRepeatInterleaveIntWithDim(n) \to$
$count\_R\_temp\{k \times n\}.$

(c) Calculation of new centroids:
$sum\_temp\{k \times n\},\ count\_R\_temp\{k \times n\} \to$
$aclnnInplaceDiv \to X^T\{k \times n\} \to$
$aclnnPermute \to X\{n \times k\}.$

The resulting matrix X may contain NAN if one or more clusters are empty. For such clusters, the centroid is preserved from the previous iteration. In this case, search for NAN is performed and the coordinates of the corresponding centroids are replaced with the previous ones:

$X\{n \times k\} \to aclnnIsFinite \to condition\_temp\{n \times k\};$

$condition\_temp\{n \times k\},\ X\{n \times k\},\ P\_X\{n \times k\} \to$
$aclnnSWhere \to YN, K.$

(10) The stopping criteria are checking. If the matrix X is equal to P_X or PP_X matrices, then it means that the optimal centroids for the clusters have been found and the matrix X no longer changes, the cycle ends. Otherwise, repeat steps 4–9. Also, the terminate condition is to reach the maximum number of iterations.

(11) The matrix X and labels L are uploaded to the host and the result is output.

3.5. *k*-medoids NPU algorithm. The scheme of the k-medoids algorithm implementation on NPU is shown in Figure 2. The algorithm description uses the notations from Subsection 3.4 with the following additions and clarifications:

- $medoids$ – $k \times 1$ medoids' indexes (host-side, C array);
- $Xmed$ – $k \times 1$ center (medoids) indexes (device-side, tensor);
- P_Xmed – $k \times 1$ medoids' indexes from the previous iteration (host-side, tensor);
- PP_Xmed – $k \times 1$ medoids' indexes from the iteration before previous (host-side, tensor);

The algorithm uses AscendCL single operators and iterates as follows:

(1) Loading initial $data$ matrix and initializing $medoids$ (pick k random data items). The $medoids$ stores only indexes of vectors from $data$ accepted as medoids of clusters.

(2) Transmitting the $data$ and $medoids$ to the memory of the Ascend NPU Device. The result is the tensors A and $Xmed$

$data(Carray) \to CreateAclTensor \to A(aclTensor)\{m \times n\};$
$medoids(Carray) \to CreateAclTensor \to Xmed(aclTensor)\{k \times 1\}.$

In the device, memory is also reserved for matrices: P_Xmed, PP_Xmed (medoids indexes at the previous iteration and the iteration before last), X(medoids), $X_{\|\cdot\|}$(norms of the medoids from X matrix), L(pointer matrix).

(3) Calculating the tensor $A_{\|\cdot\|}$ (norms of the data items from matrix A). This step is completely identical to the step 3 of the k-means algorithm (see Subsection 3.4).

(4) Obtaining the matrix X of the medoids:

$$A\{m \times n\},\ Xmed\{k \times 1\} \to aclnnEmbedding \to$$
$$X^T\{k \times n\} \to aclnnPermute \to X\{n \times k\}.$$

(5) Calculating the matrix $X_{\|\cdot\|}$is completely identical to the step 4 of the k-means algorithm (see Subsection 3.4).

(6) Calculating the distance matrix D (see the AscendCL operators chain from the step 5 of the k-means algorithm).

(7) Computing the minimum distance for each data item and obtaining a vector of pointers L. For further operating, L is copied into Host's RAM:

$$D\{m \times k\} \to aclnnArgMinGetWorkspaceSize \to$$
$$L\{M \times 1\} \to aclrtMemcpy \to clusters(Carray).$$

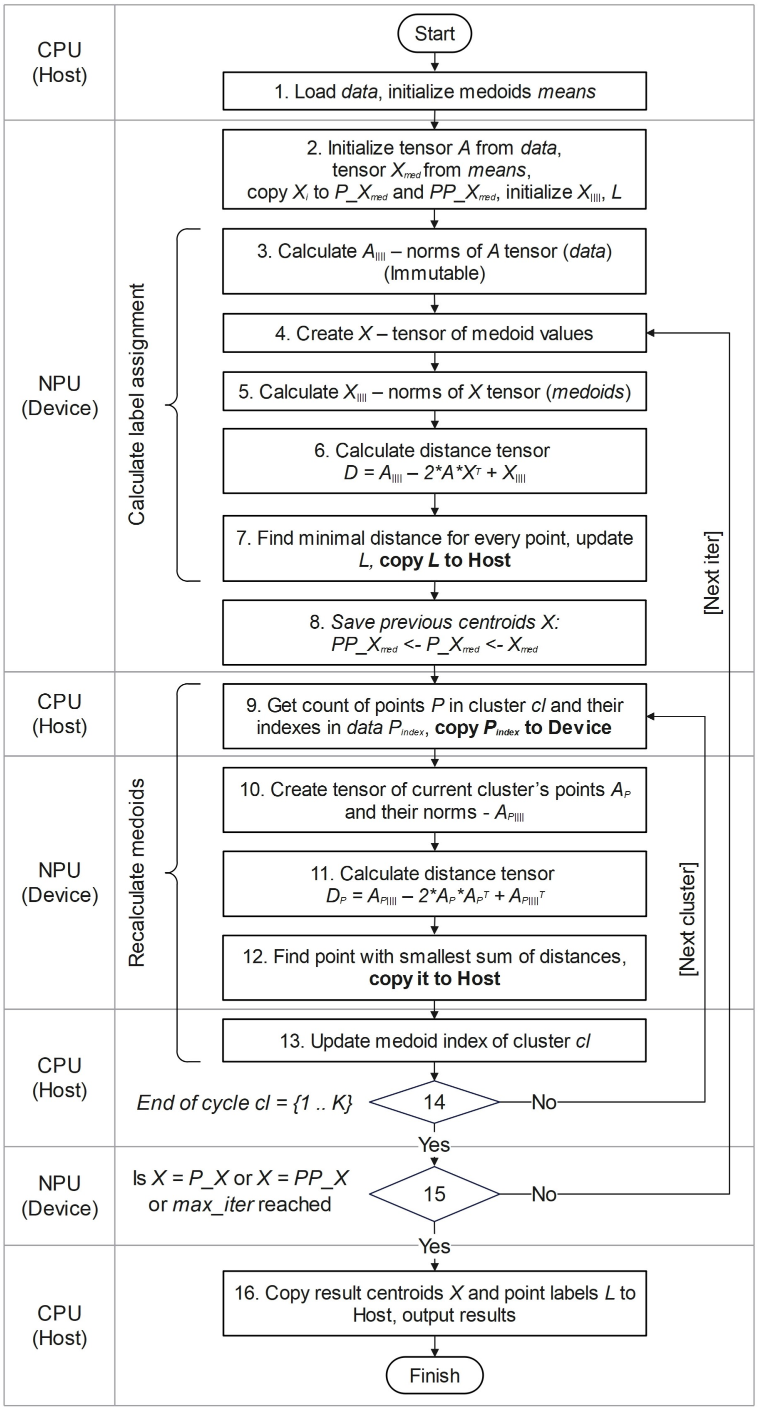
CPU (Host)
NPU (Device)
CPU (Host)
NPU (Device)
CPU (Host)
NPU (Device)
CPU (Host)
Start
1. Load data, initialize medoids means
2. Initialize tensor A from data, tensor X_{med} from means, copy X_i to P_X_{med} and PP_X_{med}, initialize $X_{||||}$, L
Calculate label assignment
3. Calculate $A_{||||}$ – norms of A tensor (data) (Immutable)
4. Create X – tensor of medoid values
5. Calculate $X_{||||}$ – norms of X tensor (medoids)
6. Calculate distance tensor $D = A_{||||} – 2*A*X^T + X_{||||}$
7. Find minimal distance for every point, update L, copy L to Host
8. Save previous centroids X: PP_X_{med} <- P_X_{med} <- X_{med}
[Next iter]
Recalculate medoids
9. Get count of points P in cluster cl and their indexes in data P_{index}, copy P_{index} to Device
10. Create tensor of current cluster's points A_P and their norms - $A_{P||||}$
11. Calculate distance tensor $D_P = A_{P||||} – 2*A_P*A_P^T + A_{P||||}^T$
12. Find point with smallest sum of distances, copy it to Host
13. Update medoid index of cluster cl
[Next cluster]
End of cycle cl = {1 .. K}
14
No
Yes
Is X = P_X or X = PP_X or max_iter reached
15
No
Yes
16. Copy result centroids X and point labels L to Host, output results
Finish

Fig. 2. *k*-medoids algorithm

(8) Saving the indexes of previous medoids:

$P\_Xmed\{k \times 1\} \to aclnnInplaceCopy \to PP\_Xmed\{k \times 1\}$;
$Xmed\{k \times 1\} \to aclnnInplaceCopy \to P\_Xmed\{k \times 1\}$.

(9) The number of points p in each cluster cl is calculated, as well as the indices of these points in $data$ – P_ind. For this step, the vector $clusters$ is used. Operations are performed in Host's RAM. The resulting indices are copied into the memory of the Ascend NPU device:
$p\_ind(Carray) \to CreateAclTensor \to P\_ind\{p \times 1\}$.

(10) A submatrix A_P is formed by only the points under the indices from the set P_ind:
$P\_ind\{p \times 1\},\ A\{m \times n\} \to aclnnEmbedding \to A_P\{p \times n\}$.

For the resulting matrix, the norms of data items are calculated:

$A_P\{p \times n\} \to aclnnNorm \to out\_temp\{p \times 1\} \to$
$aclnnInplacePowTensorScalar(2) \to out\_temp\{p \times 1\} \to$

For both matrices, a transpose operation is performed:

$A_P\{p \times n\} \to aclnnPermute \to A_P^T\{n \times p\}$;
$A_{P\|\cdot\|}\{p \times p\} \to aclnnPermute \to A_{P\|\cdot\|}^T\{p \times p\}$.

These matrices are needed to calculate the distance matrix in the next step.

(11) Calculating the distance submatrix D_P by usage the matrices $A_{P\|\cdot\|}$ and $A_{P\|\cdot\|}^T$. This step is identical to the step 5 of the k-means algorithm and the step 6 of this algorithm with precision to the notations.

(12) Finding the point with the smallest sum of distances to other points in the cluster:
$D\{p \times p\} \to aclnnReduceSum \to sum\_dist\_temp\{k \times p\} \to$
$aclnnArgMin \to min\_index\_temp\{1\}$.

(13) Updating the cluster's medoid index to the new one (executed on Host):
$min\_ind\_temp\{1\} \to aclrtMemcpy \to min\_ind(Cint)$;
$medoid[cl] = p\_ind[min\_ind]$, where cl is a current cluster.

(14) Steps 9–12 are performed for each cluster $cl = 1, \dots, K$.
At the end of the cycle, the tensor $Xmed$ is updated:
$medoid(Carray) \to aclrtMemcpy \to Xmed(aclTensor)\{k \times 1\}$.

(15) The conditions for exiting the loop are checked. If the $Xmed$ matrix is equal to the P_Xmed or PP_Xmed matrix, then this means that the optimal medoids for the clusters have been found and the $Xmed$ matrix no longer changes, the cycle ends. Otherwise, the steps 4–13 are repeated. Also, the algorithm terminates when the maximum number of iterations has reached.

(16) The matrix $Xmed$ and point labels L is uploaded to Host and the result is output.

4 Computational experiments

The computing environment used to execute and test our algorithm is equipped with a CPU Kunpeng-920 (2.6 GHz, aarch64), 1 TB of RAM and 8 NPU devices of Ascend 910B with 64GB of memory size. The operating system is EulerOS with the CANN version 8.0 installed.

The synthetic dataset is created by using a random data generator producing a uniform data distribution, so that no natural clustering structure exists.

In Table 1, we report the results of running the NPU-accelerated k-means algorithm developed and its comparison against the two competing algorithms: classical k-means [8] running on CPU and Yinyang K-means [11, 26] implementing on GPU, which is one of the fastest k-means implementations. The column m_n_k contains the number of data points in each instance (m), the number of data features (n), the number of clusters to be found (k). The column *obj.val.* shows the found objective value of the k-means problem. Next, for the three algorithms, the number of iterations (it) and the run time in seconds are specified.

Table 1. k-means algorithm

m_n_k	**obj.val**	**CPU**		**GPU**		**NPU**	
		it	**time, s**	**it**	**time, s**	**it**	**time, s**
100_8_10	3218.7	8	0.0001	8	0.0012	8	0.7961
1000_16_500	29366.9	5	0.0579	5	0.0085	5	0.7970
10000_64_600	4060770.0	14	10.4625	14	0.2834	13	0.8509
50000_128_800	4.73E+07	33	363.6061	33	36.3916	26	1.3819
100000_256_1000	2.00E+08	55	3509.7585	55	292.8231	35	5.0792
500000_1024_1000	4.20E+09	65	88672.9600	Out of memory		75	29.1492

On the small-scaled tests the NPU-accelerated k-means algorithm runs slower than both of its competitors. This behavior is not a surprise and comes from memory latency and data transfer overheads that makes algorithm slower. However, our NPU-accelerated implementation turns out to be 3000 times faster than CPU k-means on the large-scale instances including 500 000 elements, 1024 features and 1000 clusters. In this test case, the Yinyang K-means on GPU fails due to the memory error. In the test with $m = 100\,000$, the NPU-accelerated k-means is about 58 times faster than the Yinyang K-means on GPU which was announced to be the fastest implementation of k-means.

Table 2 shows the results of testing the NPU-accelerated k-medoids algorithm and the CPU-implemented conventional k-medoids. As well as for k-means, the NPU-accelerated k-medoids algorithm is slower on the small-scale tests up to 10000 items, 64 features, and 600 clusters. However, when clustering largest-scaled datasets, NPU-k-medoids algorithm turns out to be 445 times faster than its CPU counterpart.

TABLE 2. k-medoids algorithm

m_n_k	**obj.val**	**CPU**		**NPU**	
		it	**time, s**	**it**	**time, s**
10000_64_600	6054170.0	2	1.3	2	3.33
50000_128_800	7.42E+07	2	22.2	2	3.41
100000_256_1000	3.32E+08	2	124.8	2	4.11
500000_1024_1000	7.59E+09	2	2882.38	2	6.47

5 Conclusion

In the paper, we proposed and developed NPU-accelerated implementations of the k-medoids and k-means algorithms. The computational study has demonstrated that the NPU-accelerated algorithms give a significant speedup compared to their sequential implementations as well as the known fastest alternative, Yinyang K-means [11, 26].

References

[1] I. Vasil'yev, A. Ushakov, *Discrete facility location in machine learning*, J. Appl. Ind.Math., **15** (2021), 686–710. Zbl 1496.68282

[2] A.V. Ushakov, I.L. Vasilyev, T.V. Gruzdeva, *A computational comparison of the p-median clustering and k-means*, Int. J. Artif. Intell., **13**:1 (2015), 229–242.

[3] M. Mahajan, P. Nimbhorkar, K. Varadarajan, *The planar k-means problem is NP-hard*, Theor. Comput. Sci., **442** (2012), 13–21. Zbl 1260.68158

[4] D. Aloise, A. Deshpande, P. Hansen, P. Popat, *NP-hardness of Euclidean sum-of-squares clustering*, Mach. Learn., **75**:2 (2009), 245–248. Zbl 1378.68047

[5] A.V. Dolgushev, A.V. Kel'manov, *On the algorithmic complexity of a problem in cluster analysis*, J. Appl. Ind. Math., **5**:2 (2011), 191–194. Zbl 1248.62101

[6] O. Kariv, S. Hakimi, *An algorithmic approach to network location problems. I: The p-centers*, SIAM J. Appl. Math., **37**:3 (1979), 513–538. Zbl 0432.90074

[7] N. Megiddo, *Combinatorial optimization with rational objective functions*, Math. Oper. Res., **4**:4 (1979), 414–424. Zbl 0425.90076

[8] S. Lloyd, *Least squares quantization in PCM*, IEEE Trans. Inf. Theory, **28**:2 (1982), 129–137. Zbl 0504.94015

[9] X. Shen, W. Liu, I.W. Tsang, F. Shen, Q.S. Sun, *Compressed k-means for large scale clustering*, Proc. of the Thirty-First AAAI Conference on Artificial Intelligence, **31**:1, 2017, 2527–2533.

[10] Y. Gong, M. Pawlowski, F. Yang, L. Brandy, L. Boundev, R. Fergus, *Web scale photo hash clustering on a single machine*, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, New York, 2015, 19–27.

[11] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, T. Mytkowicz, *Yinyang k-means: a drop-in replacement of the classic k-means with consistent speedup*, Proc. of the 32nd International Conference on Machine Learning, **37**, ICML15, Lille, 2015, 579–587.

[12] Q. Yu, K.-H. Chen, J.-J. Chen, *Using a set of triangle inequalities to accelerate k-means clustering*, in Shin'ichi Satoh (ed.) et al., *Similarity search and applications*, LNCS, **12440**, Springer, Cham, 2020, 297–311.

[13] A.V. Ushakov, I. Vasilyev, *Near-optimal large-scale k-medoids clustering*, Inf. Sci., **545** (2021), 344–362. Zbl 1475.62196

[14] M. Baydoun, H. Ghaziri, M. Al-Husseini, *CPU and GPU parallelized kernel k-means*, J. Supercomput., **74** (2018), 3975–3998.

[15] J. Bhimani, M. Leeser, N. Mi, *Accelerating k-means clustering with parallel implementations and GPU computing*, 2015 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, New York, 2015, 1–6.

[16] S. Cuomo, V. De Angelis, G. Farina, L. Marcellino, G. Toraldo, *A GPU-accelerated parallel k-means algorithm*, Comput. Electr. Eng., **75** (2019), 262–274.

[17] S. Daoudi, C.M.A. Zouaoui, M.C. El-Mezouar, N. Taleb, *A comparative study of parallel CPU/GPU implementations of the k-means algorithm*, 2019 International Conference on Advanced Electrical Engineering (ICAEE), IEEE, New York, 2019, 1–5.

[18] J. Gao, M.Wu, J. Liao, F. Meng, C. Chen, *Clustering one million molecular structures on GPU within seconds*, J. Comput. Chem., **45**:32 (2024), 2710–2718.

[19] M. Krulivs, M. Kratochvil, *Detailed analysis and optimization of CUDA k-means algorithm*, Proc. 49th International Conference on Parallel Processing, ACM, New York, 2020, Article No. 69.

[20] M. Li, E. Frank, B. Pfahringer, *Large scale k-means clustering using GPUs*, Data Min. Knowl. Disc., **37**:1 (2023), 67–109. Zbl 1514.62117

[21] Y. Li, K. Zhao, X. Chu, J. Liu, *Speeding up k-means algorithm by GPUs*, J. Comput. Syst. Sci., **79**:2 (2013), 216–229.

[22] C. Lutz, S. BreГц, T. Rabl, S. Zeuch, V. Markl, *Efficient k-means on GPUs*, Proc. 14th International Workshop on Data Management on New Hardware, DAMON'18, ACM, New York, 2018, Article No. 3.

[23] G.J. Lim, L. Ma, *GPU-based parallel vertex substitution algorithm for the p-median problem*, Computers & Industrial Engineering, **64**:1 (2013), 381–388.

[24] Z. Mo, Y.Wang, Q. Zhang, G. Zhang, M. Guo, Y. Zhang, C. Shen, *The parallelization and optimization of k-means algorithm based on MGPUSim*, In Pimenidis, E., Angelov, P., Jayne, C., Papaleonidas, A., Aydin, M. (eds), *Artificial neural networks and machine learning – ICANN 2022*, LNCS, **13532**, Springer, Cham, 2022, 309–320.

[25] J. Nelson, R. Palmieri, *Don't forget about synchronization! A case study of k-means on GPU*, Proc. 10th InternationalWorkshop on Programming Models and Applications for Multicores and Manycores, PMAM'19, ACM, New York, 2019, 11–20.

[26] C. Taylor, M. Gowanlock, *Accelerating the Yinyang k-means algorithm using the GPU*, 2021 IEEE 37th International Conference on Data Engineering (ICDE), IEEE, New York, 2021, 1835–1840.

[27] H.-S. Park, C.-H. Jun, *A simple and fast algorithm for k-medoids clustering*, Expert. Syst. Appl., **36**:2 (Part 2) (2009), 3336–3341.

Igor Leonidovich Vasilyev
Matrosov Institute for System Dynamics and Control Theory of SB RAS,
Lermontov str., 134,
664033 Irkutsk, Russia
Email address: vil@icc.ru

Tatiana Vladimirovna Gruzdeva
Matrosov Institute for System Dynamics and Control Theory of SB RAS,
Lermontov str., 134,
664033 Irkutsk, Russia
Email address: gruzdeva@icc.ru

Denis Alexandrovich Boyarkin
Melentiev Energy Systems Research Institute of SB RAS,
Lermontov str., 130,
664033 Irkutsk, Russia
Email address: denisboyarkin@isem.irk.ru

Anton Vladimirovich Ushakov
Matrosov Institute for System Dynamics and Control Theory of SB RAS,
Lermontov str., 134,
664033 Irkutsk, Russia
Email address: aushakov@icc.ru