Parallel Clustering Algorithm for the k-medoids Problem in High-dimensional Space for Large-scale Datasets

Sergey Vandanov[1], Aleksandr Plyasunov[2], Anton Ushakov[3]

[1]Department of Physics Novosibirsk State University, Novosibirsk, Russia
[2]Lab. of Mathematical Models of Decision Making Sobolev Institute of Mathematics of SB RAS, Novosibirsk, Russia
[3]Matrosov Institute for System Dynamics and Control Theory of SB RAS, Irkutsk, Russia

Problem Statement

Clustering in high-dimensional spaces is a fundamental task in machine learning and data mining, but traditional clustering algorithms, like **k-medoids**, struggle with scalability when applied to large datasets. **The computation of pairwise distances** and the nearest neighbor search is particularly **expensive**, making these algorithms impractical for large-scale and high-dimensional data. **The goal** of this research **is to develop a parallelized algorithm that can handle large datasets efficiently**, **while maintaining high clustering accuracy**, overcoming the limitations of classical k-medoids clustering methods.

Proposed Solution

We introduce **a parallel primal-dual heuristic algorithm** for solving **the k-medoids clustering problem** in high-dimensional space. Our algorithm utilizes **GPU parallelization** to:

- Compute the distance matrix and nearest neighbors in a fraction of the time compared to CPU implementations.
- Perform subgradient optimization on the Lagrangian dual function directly on the GPU, significantly improving computational speed without compromising solution accuracy.

This parallel approach efficiently addresses challenges associated with large-scale datasets, offering improvements in both execution time and solution quality over existing methods like PAM and FasterPAM.

Methodology and Algorithm Overview

Initial Problem Formulation:

$$\min_{C \subseteq P} \left\{ \sum_{j=1}^{m} \min_{i=1,\dots,k} d(p_j, c_i) \,\middle|\, |C| = k \right\}$$

In Terms of Integer Linear Programming:

$$\min_{(x,y)} \sum_{(i,j)\in A} d_{ij} x_{ij},$$

$$\sum_{i \in \delta^-(j)} x_{ij} + y_j = 1, j \in I, \quad (1)$$

$+$

$$x_{ij} \leqslant y_i, i \in I, j \in \delta^+(i),$$

$$\sum_{i \in I} y_i = k,$$

$$y_i,\ x_{ij} \in \{0,1\}, i \in I,\ (i,j) \in A.$$

d - pairwise distance (weight) matrix
y_i**=1**, if i - medoid, 0 otherwise
x_{ij}**=1**, 1 if i is the nearest medoid to point j
k - the number of clusters.
m - the number of points.
λ_i - Lagrange multipliers.
L - Lagrangian dual function (LDF)
ρ_i - reduced cost for y_i

Using Relaxation of the Constraints (1):

$$L(\lambda) = \min_{(x,y)} \{ \sum_{(i,j)\in A} d_{ij} x_{ij} - \sum_{j \in I} \lambda_j (\sum_{i \in \delta^-(j)} x_{ij} + y_j - 1) \}$$

$$\mathcal{L}(\lambda) = \sum_{l=1}^{k} \rho_{i_l}(\lambda) + \sum_{i \in I} \lambda_i, \longrightarrow \max_{\lambda \in R^m} L(\lambda)$$

Well Known Approach

Algorithm 1 Subgradient algorithm for maximization of the Lagrangian dual function

1: Initialization: set UB, $LB \leftarrow -\infty$, γ_0, $\beta_{\max}$, λ_j^0, $s \leftarrow 0$, and $\beta \leftarrow 0$;
2: Compute reduced costs $\rho(\lambda^s)$ and the Lagrangian dual function value $\mathcal{L}(\lambda^s)$;
3: **if** $\mathcal{L}(\lambda^s) > LB$, then $LB \leftarrow \mathcal{L}(\lambda^s)$ and $\beta \leftarrow 0$;
4: **if** $LB/UB \geqslant 1 - 10^{-5}$, then stop;
5: Compute $y(\lambda^s)$ and subgradient $g(\lambda^s)$;
6: **if** $\|g(\lambda^s)\|_2^2 < 10^{-5}$, then stop;
7: *(Optional.)* Compute $Z(y(\lambda^s))$. **If** $UB > Z(y(\lambda^s))$, then $UB \leftarrow Z(y(\lambda^s))$.
8: **if** $\beta \geqslant \beta_{\max}$, then $\gamma_s \leftarrow \frac{\gamma_s}{1.01}$ and $\beta \leftarrow 0$, **else** $\beta \leftarrow \beta + 1$;
9: **if** $\gamma_s < 10^{-3}$, stop;
10: Compute $\alpha_s \leftarrow \frac{\gamma_s(1.05 \cdot UB - \mathcal{L}(\lambda^s))}{\|g(\lambda^s)\|_2^2}$;
11: Set $\lambda^{s+1} \leftarrow \lambda^s + \alpha_k g(\lambda^s)$, $k \leftarrow k+1$ and go to step 2.
12: **return** found λ, dual bound LB, (optional) upper bound UB.

Proposed Modification

Calculation of reduced costs and Lagrangian dual function using column generation method

1: Initialization: set $\rho(\lambda^s) \leftarrow -\lambda^s$, $\mathcal{L}(\lambda^s) \leftarrow 0$ and $j \leftarrow 1$;
2: Compute $\mathcal{L}(\lambda^s) \leftarrow \mathcal{L}(\lambda^s) + \lambda_j^s$ and set $h \leftarrow 1$;
3: **if** $d_{l_h(j)j} \geqslant \lambda_j^s$, **then** go to 6;
4: Compute $\rho_{l_h(j)}(\lambda^s) \leftarrow \rho_{l_h(j)}(\lambda^s) + d_{l_h(j)j} - \lambda_j^s$;
5: **if** $h < m$, **then** set $h \leftarrow h+1$ and go to 3;
6: **if** $j < m$, **then** set $j \leftarrow j+1$ and go to 2;
7: Find $T(\lambda^s)$ and compute $\mathcal{L}(\lambda^s) \leftarrow \mathcal{L}(\lambda^s) + \sum_{i \in T(\lambda^s)} \rho_i(\lambda^s)$.
8: **return** $\rho(\lambda^s)$ and $\mathcal{L}(\lambda^s)$.

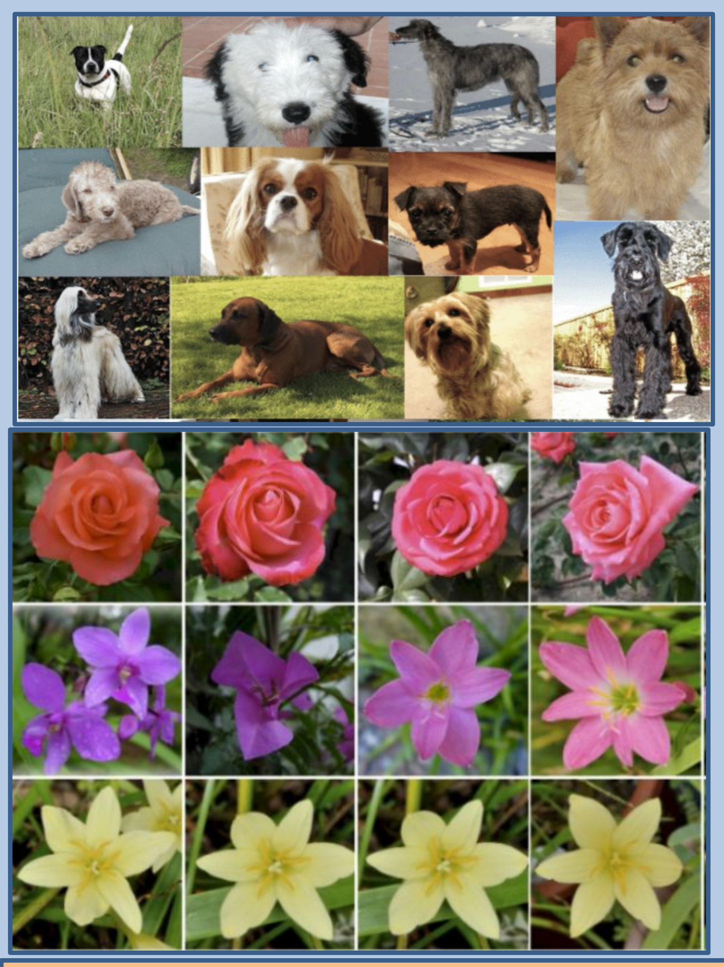
Compute reduced costs with CUDA

```
Initialize 1D distance matrix d_ij.
Initialize nearest-neighbor vector for every column l_h.
Initialize reduced costs ρ.
Initialize Lagrange multipliers (λ_v).
m ← amount of points
block_size ← 256
grid_size ← (m + block_size − 1)/block_size
// Since that moment we run code on the CUDA-kernel
// with block_size and grid_size.
j ← blockIdx.x × blockDim.x + threadIdx.x
// j - is a thread ID in terms of CUDA. Note,
// that blockIdx, blockDim, and threadIdx is
// CUDA-kernel variables.
if j < m then
    for h ← 0 to m − 1 do
        idx ← l_h[h × m + j]
        diff ← d_ij[idx × m + j] − λ_v[j]
        if diff ≤ 0 then
            atomicAdd(ρ_idx, diff)
    end if
  end for
end if
```

Results

NVIDIA cuBLAS

CUDA C/C++



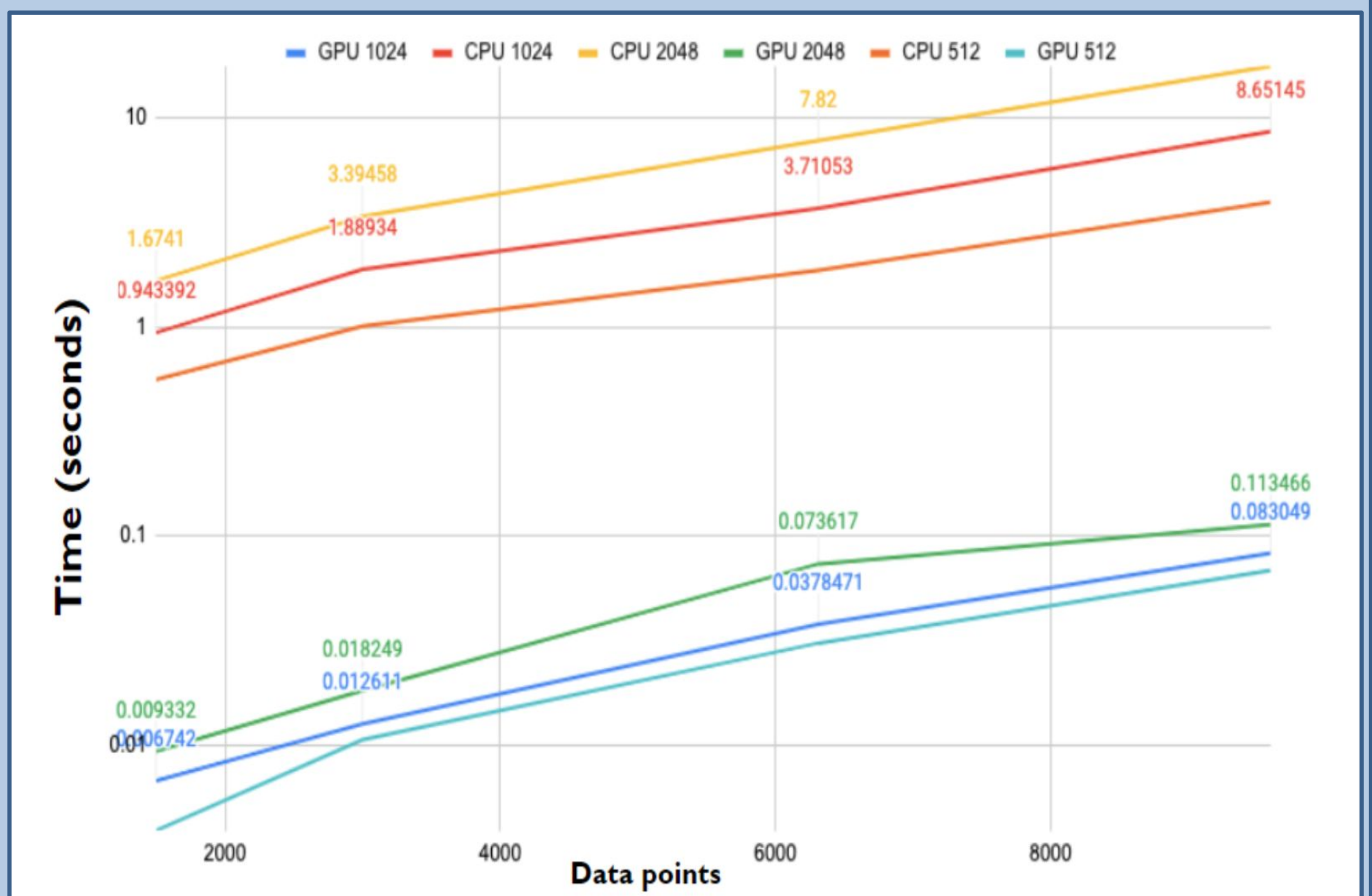


Fig. 2. Distance matrix calculation time depending on the number of points for the 512, 1024, 2048 embeddings dimension for a Stanford Dogs dataset. Logarithmic scale on the y-axis.

Comparison on 20000 1024-dim vectorized images (CLIP) with 120 clusters on Stanford Dogs dataset

Algorithm type	Obj. Val.	Time(sec.)	GAP(%)
k-medoids	365593.9	793	0.84%
PLH CPU	362551.3	398	0.01%
PLH GPU (Our)	**362551.3**	**33**	0.01%
PAM	362754.2	415	0.06%
FasterPAM	362754.2	112	0.06%

Заключение

- **Tested 12 clustering algorithms** for the k-medoids problem. Testing was conducted in two phases: 6 algorithms were excluded in the first phase for not considering problem-specific features.
- In the second phase, **evaluated the performance, stability, and scalability** of the remaining algorithms on various data volumes. The most promising algorithm was selected.
- Optimized and implemented the PLH algorithm in both parallel and standard versions using C++ with CUDA. The **parallel version** achieved a **40x speedup** on test datasets without loss of accuracy.
- A **new data preprocessing method** based on vectorization was proposed. The algorithm can **handle diverse data types** (images, text, audio) in a unified vector space.
- The developed version demonstrated the best performance across different datasets. **The integration** of the parallel algorithm into the software **is completed**, and future directions for optimization have been identified.