TiAcc: Triangle-inequality based Hardware Accelerator for K-means on FPGAs

Yuke Wang[1], Boyuan Feng[1], Gushu Li[2], Georgios Tzimpragos[1], Lei Deng[2], Yuan Xie[2], Yufei Ding[1]
[1]Department of Computer Science,
[2]Department of Electrical and Computer Engineering,
University of California, Santa Barbara.
[1]{yuke_wang,boyuan,gtzimpragos,yufeiding}@cs.ucsb.edu
[2]{gushuli,leideng,yuanxie}@ece.ucsb.edu

***Abstract*—K-means is one of the most important unsupervised learning algorithms. In this paper, we present TiAcc, a triangle-inequality based K-means hardware accelerator on FPGAs. TiAcc highlights itself with an algorithm-hardware co-design strategy tailored for K-means clustering. Specifically, TiAcc leverages a novel triangle-inequality based filtering to eliminate unnecessary distance computations without changing the final clustering results. Meanwhile, it employs a pipeline decoupling approach to mitigate the irregularity of the remaining computations, and an efficient hardware architecture design to fully exploit the pipeline and parallel processing capability of FPGAs. Moreover, TiAcc provides parameterized configuration knobs that can minimize the manual efforts in the arduous hardware design process and provides flexibility to optimize hardware designs for a variety of datasets with different sizes and dimensionalities. Intensive experiments show that TiAcc achieves an average $4.94\times$ speedup and significant energy efficiency (average $74.22\times$) compared with an optimized K-means running on a server-grade Xeon CPU.**

I. Introduction

K-means clustering, one of the most important unsupervised learning algorithms, has been widely used in many machine learning applications, such as unlabeled data clustering [1], [2], image segmentation [3]–[5], and feature learning [6], [7]. Despite its popularity, K-means usually has unsatisfactory performance due to its high computation complexity. For an input dataset with n points and k clusters in d dimensions, the complexity of the standard K-means algorithm proposed by Lloyd [8] linearly depends on n, k, and d, making it hard to handle large datasets with high dimensionalities [9]–[11].

To improve the performance of K-means, two major types of methods have been explored. The first one focuses on *algorithm optimization* (*e.g.*, triangle-inequality based filtering [12]–[14] and KD-tree based methods [9]–[11]) to eliminate redundant computations. These methods demonstrate a striking power of avoiding distance computations from every pair of data points and cluster centers, and thus reducing the complexity of n, k, and d. Yet, they introduce tremendous data dependency across different data points and cluster centers, which prevents the optimized algorithms being efficiently deployed on the modern, massively parallel hardware.

The second type of methods concentrates on hardware-level optimization, which leverages the parallel computation capability of the modern hardware (*e.g.*, multi-core CPUs, GPUs, FPGAs, and ASICs) to boost the performance [15]–[24]. Among these hardware optimizations, accelerating K-means on the FPGA is one of the most promising directions due to its considerable performance, and a good balance between energy efficiency (compared with the CPU/GPU) and developing flexibility (compared with ASIC). Nevertheless, most existing FPGA designs [16], [17], [21], [24] fail to incorporate algorithm innovations, resulting in limited overall performance improvement. Besides, these FPGA designs are often optimized and hard-coded for specific datasets, lacking adaptability for a variety of datasets with different sizes and dimensionalities.

To address the above issues, we present TiAcc, a triangle-inequality based K-means hardware accelerator on FPGAs. TiAcc highlights its algorithm-hardware co-design tailored for K-means clustering. At the algorithm level, TiAcc generalizes the triangle-inequality based optimization by introducing *cluster grouping* and *multi-level filtering* to reduce unnecessary distance computations while largely maintaining the computation regularity. At the hardware level, TiAcc leverages *data batch streaming* and *pipeline decoupling* to accelerate the remaining distance computations efficiently.

Besides, TiAcc offers template-based parameterized designs and modularized optimizations to tackle the input sensitivity. K-means is an instance-based learning algorithm and its inputs generally consist of datasets across a wide range of dimensionalities, sizes and points distributions, which would easily challenge algorithm optimizations (*e.g.*, triangle-inequality filtering), and the hardware performance (*e.g.*, memory access and processing pipeline). To this end, as shown in Figure 1, TiAcc takes the specifications of the devices and datasets to customize a unified design template. Adjustable design parameters (*e.g.*, the number of clusters groups, and data batch size) of algorithm optimizations and hardware designs are introduced to benefit the design flexibility.

In addition, TiAcc provides an end-to-end workflow to reduce manual efforts in the tedious design and implementation process. It applies algorithm and hardware optimizations at beginning stages and integrates the modern hardware design toolchains (Xilinx Vivado [25]) as its backend to generate the hardware implementations that can be directly deployed on FPGAs.

We implement TiAcc-generated K-means designs on Xil-

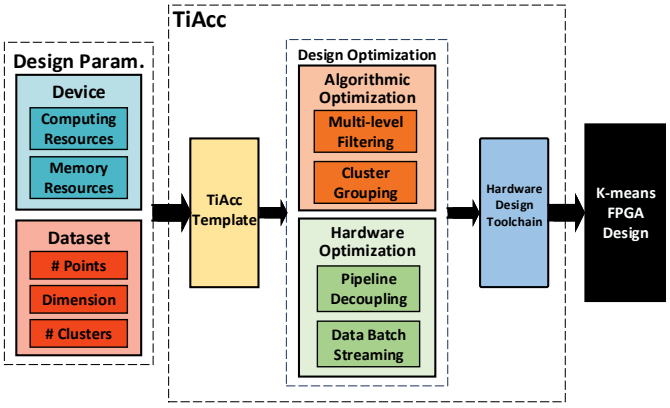


Fig. 1: TiAcc Design Workflow.

inx Zynq UltraScale+ ZCU102 [26], an off-the-shelf FPGA board, and compare its performance with the CPU and GPU based K-means implementations across various datasets. The remarkable energy-efficiency, satisfying speedup, as well as its simplified design and implementation process shape TiAcc a promising solution for K-means acceleration on FPGAs.

Overall, we have made the following contributions:

- We introduce a novel algorithm-hardware co-design strategy to remove redundant distance computations while minimizing the extra memory and computation overhead.
- We harness a highly parameterized design methodology to efficiently support various datasets by offering adjustable parameters.
- We offer an end-to-end workflow to reduce users' efforts during the arduous process of hardware design and implementation.
- Experiments on a wide range of problem settings show that TiAcc-generated designs are often superior to highly-optimized K-means implementations on the CPU and GPU. For instance, TiAcc consistently excels the highly-optimized K-means on a server-grade Xeon CPU with an average $4.94\times$ speedup, and $74.22\times$ energy-efficiency.

II. Background and Related Work

K-means is probably one of the most intuitive and popular unsupervised learning methods. The standard K-means algorithm [8] is first proposed by Stuart Lloyd in 1957 as a technique for pulse-coded modulation [8] and remains the dominant choice in practice, exemplified by the implementations in popular libraries (*e.g.*, Sklearn [27], and OpenCV [28]).

The standard K-means clusters *n* unlabeled data of *d* dimensions into *k* groups in an iterative manner, where each iteration consists of a *point-assignment* step and a *center-update* step. In the point-assignment step, the algorithm first calculates the distances between each point and all cluster centers, and then allocates every data point to the nearest cluster. In the following center-update step, the algorithm optimizes the positions of the centroids by calculating the new mean positions of the data points assigned to each cluster. The updated cluster centers will be used in the next point-assignment step. K-means algorithm will keep iterating through the above two steps until all cluster centers get stabilized. In the standard K-means, the point-assignment step dominates the overall computation time due to its high complexity, *i.e.*, $O(n \times k \times d)$ for computing the pairwise distances between each point and centroid, which leads to a major performance bottleneck when scaling up to high-dimensional large datasets [9]–[11].

To accelerate K-means, algorithm innovations, including KD-tree based [13], [29]–[31] and triangle-inequality based approaches [14], [22], [32], have been proposed to reduce the number of distance computations in the point-assignment step. Specifically, KD-tree based optimizations [29]–[31] rely on storing points in special data structures to enable a fast closest cluster search. These methods usually bring $3\times \sim 6\times$ performance improvement [29]–[31] compared with the unoptimized versions in low-dimension settings. But when handling large datasets with high dimension ($d \geq 20$), their performance would heavily suffer because of the exponentially-increased memory and computation overhead. Annoy [33] and Barnes–Hut algorithm [34], on the other hand, show better scalability compared with the standard KD-tree approach, but raising severe precision issues since they **cannot** guarantee the same output results as the original algorithm, and the quality of such approximation highly relies on the number of trees and specific application settings.

The second type is triangle-inequality optimization [12], [14], [32]. Its core idea is to utilize cheaper triangle-inequality based bound computations to replace the computation-expensive distance computations, while **guaranteeing the output correctness** as the original algorithm. Previous triangle-inequality based optimizations in K-means, such as Elkans' K-means [12] and Yinyang K-means [14], have proofed that their optimizations can always guarantee the same clustering results (*e.g.*, both point assignments and final cluster centers) as the standard version (with no distance computation removed). Such a quality guarantee is based on the idea that we only remove distance computations if and only if we are 100% sure that they will not change the final results. In addition, triangle-inequality based solutions [14], [32] are more robust, scalable, and flexible towards diverse settings compared to KD-tree based methods.

In TiAcc, we adopt the triangle-inequality based methods but amortize its computation irregularity and memory overhead through *multi-level filtering* and *cluster grouping*. It thus offers a much effective design that could be efficiently deployed on hardware accelerators like FPGAs. Before introducing the details of TiAcc, we will first briefly review *triangle inequality*, and a state-of-the-art K-means algorithm design based on it, *Elkan's K-means*. Finally, we discuss the challenges when directly applying these algorithm innovations to FPGAs.

A. Triangle Inequality

Triangle-inequality states that the sum of any two sides of a triangle must always be greater than the length of the remaining side. By making use of this property, the clustering algorithm computes distance bounds to get a "sense" of the actual distances, rather than computing the precise distance. Assume that $d(a, b)$ represents the exact distance between point a and b. Point c is an aside reference point, and $d(a, c)$ and $d(b, c)$ are known distances. By applying triangle

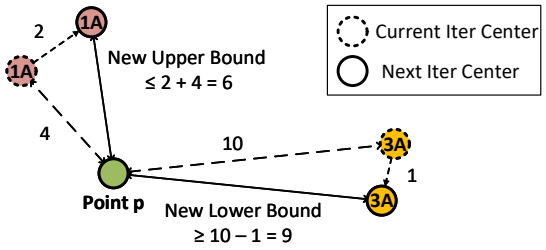


Fig. 2: Elkan's K-means approach.

inequality, the algorithm can estimate the lower and upper bounds (denoted by lb and ub, respectively) of $d(a, b)$ as follows:

$$\begin{aligned} lb(a,b) &\geq |d(a,c) - d(b,c)| \\ ub(a,b) &\leq d(a,c) + d(b,c) \end{aligned} \tag{1}$$

B. Elkan's K-means

Elkan [12] proposes Elkan's K-means for accelerating the K-means algorithm with the triangle inequality. Elkan's K-means avoids the exact point-to-cluster distance computation by using the upper and lower distance bounds, while still guaranteeing the same final results as the standard K-means. For clarity, an example illustrating its main steps and functionality is provided in Figure 2. Assume $1A$ is the closest center that point p is assigned to in the current iteration. In the next iteration, we can apply triangle inequality to compute the upper bound between p and the new $1A$, and the lower bounds between p and other centers, such as $3A$. These pairs of upper bounds and lower bounds compose an array of distance filters, which could filter out unnecessary distance computations. As this example indicates, as long as the lower bound of $d(p, 3A)$ is still larger than the upper bound of $d(p, 1A)$, *i.e.*, $9 > 6$, it is impossible that $3A$ becomes the new closest cluster center of p in the next K-means iteration. Therefore, the exact distance computation between p and $3A$ can be safely eliminated. Despite its simplicity, Elkan's K-means is still the best known triangle-inequality based K-means on common sequential CPUs for its full power of removing redundant distance computations.

C. K-means on FPGAs

Accelerating K-means on FPGAs has been widely studied [15]–[22], [24] but most of the previous K-means FPGA designs only count on hardware-level implementations, resulting in limited overall performance improvement. In addition, previous designs are often built for specific datasets, leading to limited flexibility. For example, Hussain et al. [17] propose an FPGA design for K-means clustering on biological microarray data. While their design outperforms the CPU-based implementation, their design can only handle datasets with a small number of points ($n = 2,905$) and clusters ($k = 8$), making it hard to deal with large datasets. Lin et al. [22] apply K-means on FPGAs with the distance filtering optimization, and achieve speedup over the CPU K-means on a small dataset (1/6 of the MNIST dataset with $n = 10,000$). And their designs also suffer from scalability issue on FPGAs, because 1) it requires lots of extra memory space to maintain the distance bounds (*i.e.*, $k - 1$ lower bounds per point), which limits its applicability on FPGAs with small on-chip memory; 2) it would introduce non-trivial distance bound computations under large-dataset settings; 3) it could increase the computation irregularity due to the fine-grained distance computation filtering that may largely degrade its performance on FPGAs.

In contrast, TiAcc highlights itself in reducing the computation cost, memory overhead, and computation irregularity. And it overcomes the aforementioned issues through integrating the algorithm and hardware design optimizations effectively.

III. Algorithm-level Design

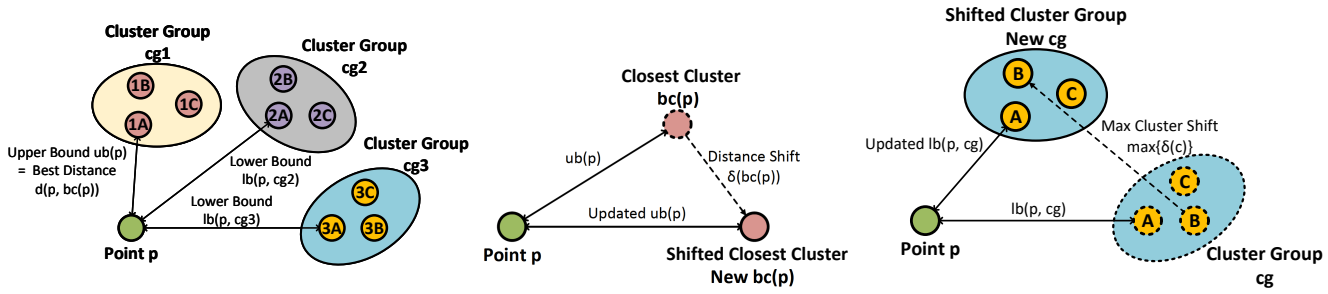
TiAcc K-means highlights its algorithm-level design by taking the hardware limitations (*e.g.*, on-chip memory, and pipeline processing paradigm) into careful consideration. Specifically, TiAcc K-means reduces the on-chip memory overhead by *cluster grouping* and maintains computation regularity by *multi-level filtering* (point- and group- level filtering).

Algorithm 1: TiAcc K-means

input : Point set, P = $\{p_1, p_2, \ \ldots, \ p_n\}$
output: Point label, PL = $\{pl_1, pl_2, \ \ldots, \ pl_n\}$

1 Initialize C = $\{c_1, c_2, \ \ldots, \ c_k\}$;
2 CG = $\{cg_1, cg_2, \ \ldots, \ cg_{k_c}\}$ = $clusterGroup(C)$;
3 $disBound = disComp(P, C, CG)$;
4 **while** *TiAcc K-means not converge* **do**
5 $\quad P_{pass} = pointFilter(P, disBound)$;
6 $\quad CG_{pass} = groupFilter(P_{pass}, disBound, CG)$;
7 $\quad$ **foreach** p *in* P_{pass} **do**
8 $\quad\quad assignedLabel[p] = argmin(p, CG_{pass}[p])$;
9 $\quad$ **end**
10 $\quad updateCenter(C, assignedLabel)$;
11 **end**

As illustrated in Algorithm 1, TiAcc K-means consists of several major steps. In the beginning, we create the clusters' centers at $line$ 1, by using initialization options such as "K-means++" [35] and "random" [36]. At $line$ 2, we group $initial\ clusters$ to prepare cluster groups for multi-level filtering in distance computation reduction. After that, we run the standard K-means with $clusters$ and $points$ for only one iteration to build the initial distance bounds for each data point at $line$ 3. The *multi-level filtering* optimization starts from the second iteration, which covers $line\ 4 - 11$. It includes point-level filtering and group-level filtering at different levels of granularity. Specifically, at $line$ 5, point-level filtering removes points that will avoid all of their following distance computations in the current iteration. At $line$ 6, group-level filtering further prunes the cluster groups of the points which pass the point-level filtering. At $line$ 8, we calculate the exact distance between the points and clusters in the cluster groups that have passed the aforementioned multi-level filtering. Each point will be assigned to its closest cluster. Last, we update the clusters' centers at $line$ 10. If any center has been changed, we will keep executing the loop until the algorithm converges. The multi-level filtering optimization accelerates the point-assignment step and will **NOT** change the final result.



(a) Point-cluster Relationship. (b) $ub(p)$=$ub(p)$+$\delta(bc(p))$. (c) $lb(p, cg)$=$lb(p, cg)$-$\underset{c \in cg}{max}\ \delta(c)$.

Fig. 3: Point-cluster Relationship; Update of distance upper bound (ub) and lower bound (lb).

A. Cluster Grouping

Cluster grouping gathers a set of clusters that are close to each other based on their initial centers (an example is depicted in Figure 3a). Cluster grouping brings two key benefits. First, TiAcc K-means only needs to store $n \times k_c$ ($k_c << k$) distance bounds between each of the n point and all k_c cluster groups for filtering computations, which largely reduces the memory and computation overhead compared with Elkan's K-means ($n \times (k-1)$ distance bounds). Second, TiAcc increases memory efficiency by accessing a group of clusters instead of an individual cluster each time.

Based on the cluster grouping, TiAcc K-means maintains two types of distance bounds for each point: distance upper bound ub and distance lower bound lb, as shown in Figure 3b and Figure 3c, respectively. The update of the distance upper bound follows the same process as Elkan's K-means, where the new distance upper bound of p for the current iteration is calculated by applying triangle inequality on previous distance upper bound ub and the distance shift $\delta(bc(p))$ of p's closest cluster $bc(p)$. The distance lower bound $lb(p, cg)$ generalizes the lower bound of Elkan's K-means in the sense that it computes the lower bound from a point p and a set of centers denoted by cg. By conservatively updating the distance lower bound ($lb(p, cg)$) with the maximum distance shift of the cluster group ($\underset{c \in cg}{max}\ \delta(c)$), TiAcc extends the usage of triangle inequality from the regular point-to-point bound computation to a more efficient point-to-group bound computation.

However, there are still three challenging questions that must be addressed before we can exploit the efficiency of cluster grouping. 1) How to build these cluster groups efficiently? 2) Will the quality of cluster groups affect the final output results? 3) How many cluster groups should be created for efficient filtering?

First, to build the cluster groups, we run standard K-means on initial cluster centers on the CPU through a few iterations (five iterations is used throughout our evaluations for balancing filtering performance and runtime overhead).

Second, despite that cluster centers will be updated iteratively, the relative adjacency of these centers does not change much across iterations. Thus TiAcc groups the clusters only once at the beginning. Cluster regrouping across iterations, while feasible, does not help in our empirical study. Moreover, cluster grouping will **NOT** affect the correctness of the final results but could impact the efficiency of triangle-inequality based filtering. Because if groups are not representative, distance bounds could be relatively loose and fewer distance computations can be eliminated.

Third, the number of cluster groups (k_c) provides a design knob for our algorithm optimization that controls the trade-off between the computation elimination and the regularity. With more cluster groups, TiAcc has the potential to exploit more redundant distance computations. But it also introduces more irregularity to the remaining distance computation, along with extra distance-bound computation and memory overhead. The best choice of k_c could actually differ across different datasets.

B. Multi-level Filtering

As the major algorithm optimization of TiAcc K-means, multi-level filtering reduces the distance computations at different levels of granularity. Based on the distance bounds from the cluster grouping, a group-level filtering condition can be applied to remove cluster groups. Previous algorithm optimizations (*e.g.*, Elkan's K-means [12]) uses fine-grained filtering to directly remove distance computations between points and clusters. Such a fine-grained approach may eliminate more redundant distance computations, but it loses flexibility and efficiency while increasing the memory overhead.

To further improve the filtering performance, we design point-level filtering placed before the group-level filtering. Point-level filtering allows us to identify points that would not change their closest cluster center in the new iteration, and safely remove all distance computations of such points. Our empirical study also shows that just applying point-level filtering can reduce more than 40% distance computations, which demonstrates its importance.

Point-level Filtering Point-level filtering labels points for distance computation reduction. If a point does not satisfy the point-level filtering condition, it will be labeled as PASS. Then this point will be forwarded to the next stage of the processing pipeline, where the group-level filtering will be activated to check all cluster groups of this point. On the other hand, if the point satisfies the point-level filtering condition, it will be labeled as NOPASS, then all the following processing stages of this point will be avoided.

The input of point-level filtering relies on several distance bounds of an incoming point p: 1) $ub(p)$: the upper bound distance between p and its closest cluster in the last iteration; 2) $lb(p, cg)$, the lower bound distance between p and cluster group cg in the last iteration; 3) $\delta(bc(p))$, the distance shift of the p's closest cluster in the last center update; 4) $\delta(c)$, the

distance shift of the clusters other than p's closest cluster in the last center update.

At the end of each K-means iteration, the cluster centers will be updated, and the distance upper bound between p and the closest cluster of p in the current iteration will be changed accordingly. As described in Figure 3b, the distance upper bound of p now becomes

$$ub(p) + \delta(bc(p)). \tag{2}$$

Similarly, the distance lower bound between p and a cluster group cg can be updated as

$$lb(p, cg) - \underset{c \in cg}{max}\ \delta(c). \tag{3}$$

where $\underset{c \in cg}{max}\ \delta(c)$ selects a cluster c from all clusters inside cg which maintains the maximum value of distance shift in the last center update. The processing of the cluster group shifting and the lower bound updating is depicted in Figure 3c.

Based on these distance bounds, the difference between the minimum of updated lower bounds and the updated upper bound can be calculated by a simple distance-bound comparison. Therefore, if we can have

$$\underset{cg \in CG}{min}\{lb(p, cg) - \underset{c \in cg}{max}\ \delta(c)\} \geq ub(p) + \delta(bc(p)) \tag{4}$$

where CG is the set of all cluster groups, all following distance calculations of p will be avoided, and the closest cluster of p will not change in the current iteration. Otherwise, if the result of distance bounds comparison satisfies

$$\underset{cg \in CG}{min}\{lb(p, cg) - \underset{c \in cg}{max}\ \delta(c)\} < ub(p) + \delta(bc(p)) \tag{5}$$

potentially redundant distance computations of p will be further eliminated in group-level filtering based on the distance lower bounds of p and all cluster groups.

Group-level Filtering Group-level filtering reduces distance computations in a more fine-grained way at the cluster-group level. In group-level filtering, the updated distance lower bounds of a point p will filter cluster groups. In the distance bounds comparison, if we can have

$$lb(p, cg) - \underset{c \in cg}{max}\ \delta(c) < ub(p) + \delta(bc(p)). \tag{6}$$

then cluster group cg will be kept as the candidate group. Then all distances between the clusters inside cg and p are going to be calculated in the next stage of the processing pipeline. Otherwise, if the updated lower bound is greater than or equal to the updated upper bound for p and cg, then distance calculations between p and all clusters inside cg can be skipped.

IV. Accelerator Design

In this section, we introduce the hardware accelerator design of TiAcc. We first detail the hardware components of TiAcc architecture, followed by its relation with the algorithm optimization in the last section. Then we introduce two key hardware-level optimizations (*data batch streaming* and *pipeline decoupling*), which can largely simplify the FPGA implementations of the aforementioned algorithm optimizations while reducing the resource consumption of FPGAs.

A. TiAcc Architecture

As shown in Figure 4, TiAcc K-means accelerator includes five major components: *block RAM (BRAM)*, *point-level filter*, *group-level filter*, *distance calculator*, and *center updater*. These hardware components can effectively support the aforementioned upper-level algorithm design. Specifically, BRAM holds the necessary data (*e.g.*, distance bounds) for a fast and low-cost memory access; Point-level and group-level filters implement the functionalities of the multi-level filtering; Distance calculator provides the high-performance fully-vectorized point-to-point distance computation; Center updater updates the clusters' center positions in each iteration. More importantly, TiAcc incorporates an efficient *data batch streaming* strategy for a high-performance data communication and a *pipeline decoupling* approach to reduce the computation irregularity and workload imbalance. TiAcc also introduces a tunable parameter – data batch size, to enable hardware design reconfiguration.

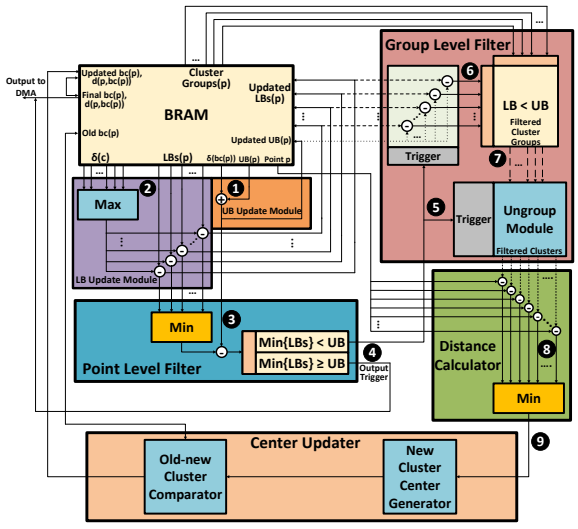


Fig. 4: TiAcc Architecture Design.

Point-level Filter. To implement functionalities of point-level filtering, TiAcc architecture design includes a **UB Update Module** in hardware design for managing the point-to-cluster distance upper bound, as shown in Figure 4b (❶). Correspondingly, an **LB Update Module** is integrated to process the cluster group shifting and the lower bound updating, as shown in Figure 4b (❷). From the distance bounds (Figure 4b (❸)) generated by the previous two modules, the difference between the minimum of updated lower bounds and the updated upper bound can be calculated by using our specified **Distance Bounds Comparison Module** (Figure 4b (❹)). Based on the distance-bound comparison, the group-level filter can be triggered to explore the potential redundant distance computation of a point at cluster-group level (Figure 4b (❺)).

Group-level Filter. Group-level filter (Figure 4b (❻)) will be invoked only if it gets the trigger signal sent by the point-level filter. Group-level filter is used to reduce the distance computations in a more fine-grained way at the cluster-group level. Cluster groups that have not been filtered out will be

sent to an **Ungroup Module**, as shown in Figure 4b (❼) for further computation.

Distance Calculator. Distance calculator, as shown in Figure 4b (❽), computes the precise distance between points and clusters' centers. Distance calculator will only be triggered if the incoming points and cluster groups have passed the multi-level filtering at the previous steps. Then the minimum of these calculated distances becomes the new closest cluster distance by passing a **Minimum Distance Selection Module** (Figure 4b (❾)). The new closest cluster distance becomes a new distance upper bound of this point. Meanwhile, the closest cluster which corresponds to this new distance upper bound is updated as well.

Center Updater. Center updater takes the closest cluster distance and the closest cluster label of each data point from the distance calculator to generate the new cluster centers. The old cluster centers will be fetched from the on-chip BRAM. After the new cluster centers have been generated, the distance shift between the old cluster centers and the corresponding new cluster centers will be calculated by an **Old-new Cluster Comparator**. Finally, the old clusters will be replaced by the new clusters. All clusters will be held in BRAM until TiAcc K-means converges. Compared with updating cluster centers off-chip, updating cluster centers on the FPGA largely reduces the time and cost of data transferring between different kinds of memories.

B. Data Batch Streaming

To efficiently transfer data, we propose a data batch streaming strategy to transfer data between the external DRAM and the on-chip BRAM. There are several benefits of this strategy: 1) The on-chip memory of FPGAs is limited (KB to MB), whereas datasets are large (MB to GB). Compared with caching all data at on-chip memory, data batch streaming can minimize the memory overhead by keeping only a small part of the entire dataset. Most of the previous K-means FPGA-based designs have to store the whole dataset at on-chip memory before running K-means. Therefore, their designs largely depend on the size of FPGA on-chip memory and datasets; 2) Compared with storing all data in the external memory, data batch streaming is superior in its minimized data access latency. The on-chip memory offers high access speed (sub-nanoseconds to nanoseconds per data access), whereas the external memory is slow (around 50 nanoseconds per data access). In addition, the most frequently used data are cached in the on-chip memory, such as clusters and distance bounds, thus the frequency and cost of fetching data from the external memory are reduced as well.

C. Pipeline Decoupling

While the "point-centered" processing paradigm gives a clear guideline to handle individual point effectively, K-means clustering with filtering optimization would cause severe computation irregularity and workload imbalance when the number of points scales up. As shown in Figure 5a, after the point and group level filtering, points with fewer distance computations (*i.e.*, higher filtering ratio) would have lower processing latency, while points with more distance computations (*i.e.*, lower filtering ratio) would have longer processing latency. The overall latency is usually determined by the longest one. Therefore, distance computation saving from filtering cannot be fully capitalized as performance gains because of workload imbalance from the highly-coupled filtering and distance computation process. Moreover, such a one-point-single-pipeline hardware design scheme limits the number of points being processed each time due to on-chip resource constraints.

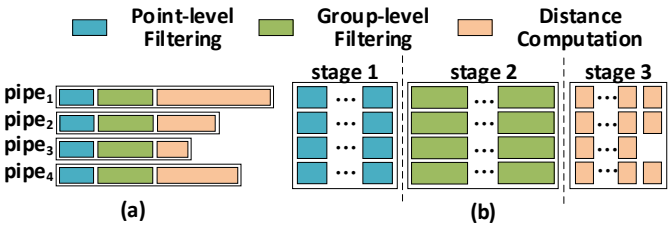


Fig. 5: (a) Conventional Pipeline; (b) Decoupled Stages.

Fortunately, TiAcc transforms the above "point-centered" processing pipeline to "task-centered" stages, which have decoupled filtering and distance computation steps to mitigate above issues. As shown in Figure 5b, the original lengthy pipeline is decomposed into three stages, each of which manages a specific task. There are several benefits of applying such a transformation: 1) computations of different points can be well abstracted by using unified processing stages, each of which manages one processing task for all points and thus eliminates the computation irregularity in the one-point-single-pipeline setting; 2) highly-modularized processing stages allow for further optimizations on each processing stage to maximize its performance gains (*e.g.*, increasing the number of computation units, and optimizing the on-chip memory architecture); 3) workload imbalance can be reduced through more evenly distributed computations. Because distance computation units can share workloads from all points instead of just sticking to distance computations from only one point.

V. Evaluation

This section evaluates and compares the efficiency of TiAcc K-means FPGA designs against CPU, GPU, and other FPGA K-means implementations on real-world datasets.

TABLE I: Datasets for Experiments.

Dataset	# Points (n)	# Dimension (d)	# Clusters (k)
Parkinsons Updrs	5,875	21	38
Letter Recognition	20,000	16	70
Electronic Board Read	45,781	5	53
KEGG Meta Net	65,554	28	64
Skin NonSkin	245,057	4	62
3D Spatial Network	434,874	3	82

A. Experiment Settings

1) ***Datasets****:* We use six real-world datasets obtained from UCI Machine Learning Repository [37], covering a wide range of data point size (5,000 $\sim$ 430,000) and dimensionality (3 $\sim$ 28). These datasets are also used in related work [14], [32], [38]. Details of datasets are listed in Table I.

TABLE II: Performance and Energy Efficiency Comparison.

Dataset.	Time. TiAcc (s)	Power. TiAcc (W)	Speedup. *vs.* ARM	En.Eff. *vs.* ARM	Speedup. *vs.* Xeon	En.Eff. *vs.* Xeon	Speedup. *vs.* Titan Xp	En.Eff. *vs.* Titan Xp
Parkinsons Updrs	0.001	5.26	28.34×	**8.57**×	6.90×	**65.95**×	3.00×	**42.43**×
Letter Recognition	0.006	4.64	13.17×	**5.42**×	1.92×	**25.10**×	0.67×	**13.33**×
Electr Board Read	0.010	4.06	9.89×	**4.99**×	1.79×	**21.53**×	0.70×	**13.88**×
KEGG Meta Net	0.016	5.52	16.53×	**4.91**×	2.27×	**27.53**×	0.31×	**6.97**×
Skin NonSkin	0.061	3.95	8.91×	**2.98**×	8.83×	**160.06**×	0.31×	**6.82**×
3D Spatial Network	0.143	4.16	6.42×	**2.28**×	7.93×	**145.14**×	0.19×	**4.31**×

*2) **Baselines**: **(a) CPU-based Implementations.*** We use Sklearn [27] K-means for the CPU comparison. Sklearn [27] is one of the most popular machine learning libraries developed and maintained by lots of domain experts and software engineers, and its highly-optimized K-means implementation can achieve the state-of-the-art performance with guaranteed correctness; ***(b) GPU-based Implementations.*** We use CUDA K-means from the NVIDIA official repository [39] for the GPU comparison. K-means implementation released by NVIDIA is a highly-efficient CUDA-based implementation that leverages the power of cuBLAS [40] library for acceleration, which maximizes its performance gains on CUDA-capable GPUs and outperforms other publicly available GPU implementations based on our experimental studies; ***(c) FPGA-based Implementations***. It is challenging to give a fair comparison with other FPGA-based implementations since existing FPGA designs are mostly **NOT** publicly available and are **ONLY** crafted for specific algorithms/inputs/FPGAs. Instead we manually implement Elkan's K-means on FPGAs as a reference design to demonstrate the algorithm-hardware co-design benefits of TiAcc (Section V-D).

*3) **Platforms**:* We implement and evaluate TiAcc K-means designs (operating at 300MHz) on ZCU102 (Xilinx Zynq UltraScale+ MPSoC ZCU102 [26]) by using Xilinx Vivado Design Suite (v2018.3) [25]. ZCU102 is a general-purpose evaluation FPGA board with -2L speed grade for rapid-prototyping based on the Zynq UltraScale+ MPSoC (multiprocessor system-on-chip). ZCU102 has a quad-core ARM Cortex-A53, dual-core Cortex-R5 real-time processors based on Xilinx's 16nm FinFET+ programmable logic fabrics (Logic Slices: 600,000, DSP: 2,520, LUT: 274,080, Flip-Flops: 548,160, Block RAM: 4,012.5 KB, 512MB off-chip DDR4 memory connected with a 16-bit bus running at 2.4Gbps). Note that all FPGA designs are evaluated (*e.g.* power and performance) based on post-place-and-route results. We also test our TiAcc framework on PYNQ-Z1 FPGA [41]. The overall trend of performance improvements is very similar and only differ in net speedup due to different on-chip resources and lower clock frequency. We thus only show the results of ZCU102 considering the page limit.

*4) **Metrics**:* To evaluate the performance of the TiAcc K-means design, the time of each K-means iteration has been measured, and we define the speedup as:

$$\textbf{Speedup} = \frac{Time(Other)}{Time(TiAcc)}. \tag{7}$$

To measure the ratio of performance versus the power consumption of TiAcc K-means design, we define the energy efficiency as:

$$\textbf{Energy\_Efficiency} = Speedup \times \frac{Power(Other)}{Power(TiAcc)}. \tag{8}$$

where *Other* can be one of the implementations (*e.g.*, CPU or GPU) introduced above.

B. Compared with CPU-based Implementations

Sklearn [27] K-means is running on an 8-core 16-thread **Intel Xeon Silver 4110 Processor** [42] (Clock Frequency: 2.1GHz, Memory: 64GB DDR4) with **all 8 cores fully utilized**. We use the software timer $perf\_counter$ for measuring the running time and an off-the-shelf Poniie PN2000 power meter [43] to evaluate the actual runtime power consumption. The result in Table II shows TiAcc K-means on ZCU102 FPGA can achieve up to 8.83× speedup on Skin NonSkin dataset and has an average 4.94× speedup across all six experimental datasets. We also observe that TiAcc K-means showcases its speedup advantage on the datasets with the large point size such as Skin NonSkin ($n = 245,057$) and 3D Spatial Network ($n = 434,874$), which demonstrates its algorithm-hardware co-optimization effectiveness in handling relatively large datasets. This considerable performance comes from the group-level filtering to remove most of the unnecessary distance computations and accelerating remaining distance computations through efficient parallelization and pipelining. Moreover, the significant energy efficiency (up to 160.06×, 74.22× on average) compared with Xeon CPU shows the advantage of TiAcc K-means designs in energy-saving and indicates its potential applications in most of the power-constrained embedded settings, such as IoT devices.

In the comparison with the embedded processor, we run Sklearn K-means on Jetson Nano Board [44] (**a Quad-Core ARM Cortex-A57**, Clock Frequency: 1.9GHz, Memory: 4GB LPDDR4) with **all 4 cores fully in use**. The result in Table II shows TiAcc K-means on the FPGA is a good alternative with much better performance (6.42× $\sim$ 28.34×, average 13.87×) and energy efficiency (2.28× $\sim$ 8.57×, average 4.84×) compared to the popular ARM-based solution.

C. Compared with GPU-based Implementations

GPU-based computation highlights itself with massive parallelization, significant performance, and high power consumption. In this experiment, CUDA K-means is running on **NVIDIA Quadro P6000 GPU** [45] (3840 CUDA cores, Memory: 24GB GDDR5X, Peak Memory Bandwidth: 432GB/s, Peak Single Precision Performance: 12 TFLOPs). Quadro P6000 is a high-end GPU in high-performance machine learning and deep learning domain. We use the NVProf [46] tool

from NVIDIA for measuring both GPU runtime power and the CUDA kernel execution time. As shown in Table II, TiAcc K-means shows its strength in energy saving, which demonstrates $4.31\times \sim 43.23\times$ better energy efficiency compared with Quadro P6000. We also notice that the performance superiority of GPU is more pronounced on large datasets due to its massive thread-level parallelism. However, such benefit could fade or even disappear on small datasets, such as Parkinson Updrs ($n = 5,875$, $d = 21$, $k = 38$), where TiAcc K-means is $3\times$ faster. Because in this case most of the data during K-means iterations can be directly accessed from the FPGA on-chip memory instead of the external DDR memory, which can largely reduce memory access latency and energy consumption. On 3D Spatial Network dataset, however, GPU-based implementation shows higher performance. Because this dataset comes with an extreme setting (*i.e.*, the smallest feature dimension $d = 3$), which would lower the distance filtering benefits and under-utilize FPGA resources and lead to inferior performance. While GPU implementation can still maximize its performance gains through more active threads in such a setting, we want to emphasize that even under such extreme settings, TiAcc can still beat the GPU in terms of performance/power ratio, as the GPU always comes with much higher power consumption.

D. Compared with Elkan's K-means

In this section, we further demonstrate the effectiveness of TiAcc designs from both algorithm and hardware perspectives through carefully designed experiments. We choose Elkan's K-means for comparison because it is the most representative optimized K-means algorithm with the state-of-the-art CPU performance by using the fine-grained distance filtering optimization, while our TiAcc K-means centers around a more coarse-grained distance filtering strategy. Comparison with Elkan's K-means can break down TiAcc K-means benefits in terms of its algorithm optimization, hardware acceleration, and the effective co-design of combining these two.

Specifically, we implement four designs, **I)** Elkan's K-means on CPU; **II)** TiAcc K-means on CPU; **III)** Elkan's K-means on FPGA; **IV)** TiAcc K-means on FPGA. Considering that there is no publicly available Elkan's K-means FPGA-based implementation ([22] also does not open-source its design), we implement Elkan's K-means on ZCU102 FPGA by following the design principles from [22] and also apply optimizations for improving the memory and computation efficiency to maximize its performance. And we evaluate above designs on three datasets (Letter Recognition, KEGG Meta Net dataset, and Skin NonSkin) for demonstration.

As shown in Figure 6, we observe that in the comparison of CPU-based implementations (Design I and II), TiAcc K-means is slower compared with the Elkan's K-means on higher dimension datasets (KEGG Meta Net: $d = 28$). The reason is that the cost of each point-to-point distance computation is expensive in the high-dimension dataset setting, and Elkan's K-means applies fine-grained filtering that can remove more distance computations , whereas TiAcc K-means uses coarse-grained group-level filtering that removes fewer distance computations. However, on low-dimension datasets (Skin NonSKin: $d = 4$), the cost of each point-to-point distance computation is not very expensive, and the extra bound computation overhead of Elkan's K-means would offset the performance benefits from reducing distance computations.

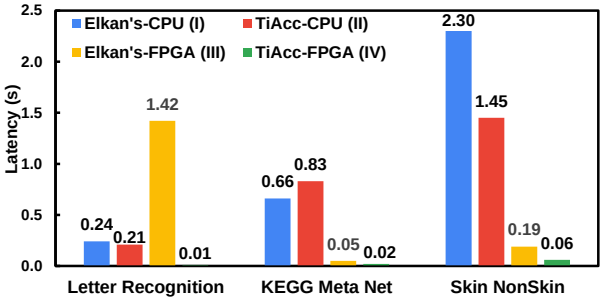


Fig. 6: Comparison with Elkan's K-means.

In the evaluation of FPGA-based implementations (Design III and IV), we observe that TiAcc K-means always maintains the lowest latency among these four implementations. The major reason behind its considerable performance is that TiAcc leverages an effective algorithm-hardware co-design, which can well combine and balance the benefits from the hardware-aware algorithm optimization and the FPGA acceleration. This conclusion can be strengthened through comparing Design II and IV, where the FPGA acceleration of TiAcc K-means can add an additional $23.88\times \sim 51.85\times$ speedup on top of its algorithm optimization. In contrast, Elkan's K-means heavily counts on the aggressive hardware-agnostic algorithm optimization (fine-grained filtering), even though it gets performance gains on the CPU ($1.25\times$ faster than TiAcc K-means CPU implementation) on higher dimension datasets, it achieves inferior performance on the FPGA due to excessive computation irregularity and memory overhead.

In addition to performance comparison, we also compare the resource consumption of TiAcc design with Elkan's K-means on Skin NonSkin dataset.

TABLE III: Resource Comparison with Elkan's K-means.

Design	BRAM _18K	DSP48E	FF	LUT
Elkans	19	3	39453	41627
TiAcc	17	3	32488	30641

As shown in Table III, we can see that Elkan's K-means consistently takes more computation and memory resources because more distance bounds are maintained (stored and updated) during the fine-grained filtering optimization. This also help to demonstrate the a better resource efficiency of TiAcc K-means designs.

E. Distance Computation Reduction

In this section, we decompose the optimizations in TiAcc K-means to show their benefits in detail. We implement two algorithms (**Design I**: a standard K-means (Lloyd's Algorithm); **Design II**: an optimized K-means with TiAcc algorithmic optimizations (applying cluster grouping and multi-level triangle-inequality optimization). Design I and Design II have been evaluated in their corresponding distance computation reduction performance across all six datasets used in the aforementioned experiments. We record the total number of

TABLE IV: Data Type Comparison.

Distance Type	Data Type	# DSP	# FF	# LUT	Percent Err.	Latency (Clock Cycles)
Euclidean Distance	I	51	281	1512	22.29%	35
	II	5	758	1370	1.62%	64
	III	16	1785	2455	0%	70
Manhattan Distance	I	0	190	981	21.67%	7
	II	2	427	2072	0.36%	56
	III	6	1231	4503	0%	69

Note: I: Fixed-point; II: Single-precision Floating-point; III: Double-precision Floating-point.

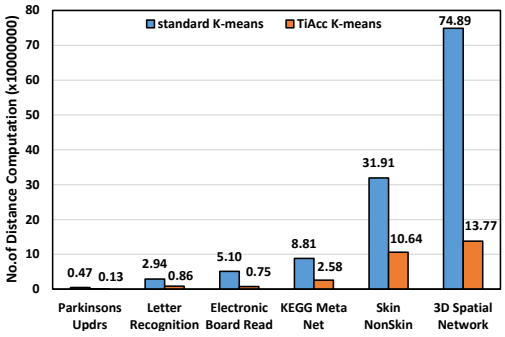


Fig. 7: Distance Computation Reduction.

distance computations within the first twenty iterations of both TiAcc K-means and standard K-means, as shown in Figure 7. To measure the distance computation reduction, we define the reduction ratio as $\frac{nc_{std}-nc_{TiAcc}}{nc_{std}}$, where the nc stands for getting the number of distance calculations in each design. We observe that TiAcc reduces 74.5% distance computations of standard K-means on average, which indicates that our novel multi-level triangle-inequality based filtering is able to capture and remove most unnecessary expensive distance calculations.

F. Study on the Data-Type Selection

TiAcc uses the single-precision floating-point as its major data type. There are several benefits of this data type: 1) It maintains accuracy without compromising the final result of K-means clustering; 2) It is resource-saving and energy-efficient when implemented on FPGAs; 3) It can cover a wide range of data value and avoid the data overflow issue during the computation. To show the performance and accuracy of the single-precision floating-point data type, we implement two types of distance calculators: Euclidean distance and Manhattan distance calculators. For each type of distance calculator, we apply three different data types: **I** (Fixed-point: 32 bits in total, 13 bits in fractional part), **II** (Single-precision floating point: 32 bits), **III** (Double-precision floating point: 64 bits). In the evaluation, we compare data types in each distance calculator in terms of their accuracy, resource consumption, and performance. All synthesized modules are set to run at 100 MHz. The fractional part of the fixed-point is set to 13 bits, which can accommodate 4 decimal digits for maintaining enough precision. The Within-Cluster Sum of Square (WCSS) is used as a metric for evaluating the clustering result of K-means clustering, which is defined as

$$\textbf{WCSS} = \sum_{i=1}^{n} d(p_i, bc(p_i)). \tag{9}$$

We measure the percent error of the fixed-point and single-precision floating-point solution with respect to the double-precision floating-point solution in terms of their WCSS. The result is based on the average resource consumption, percent error, and latency of the six experimental settings (d). From Table IV, we can see that in the same type of distance calculator, II (Single-precision floating point) always maintains a much lower precision loss (I: 1.62%, II: 0.36%) compared with I (Fixed-point) (I: 22.29%, II: 21.67%). The high precision loss of I in distance calculation leads to a sub-optimal result, which compromises the clustering performance. We also see that II outperforms III because II uses only 50% hardware elements of III on average. This helps reduce the overall power consumption and design complexity. The result supports our choice of using single-precision floating-point as the major data type in TiAcc design for the trade-off between accuracy and performance.

G. Study on the Resource Breakdown

To better understand the resource utilized by TiAcc, we decompose the resource of each module in TiAcc FPGA-design on the Skin NonSkin dataset.

TABLE V: Resource Decomposition for Skin NonSkin.

Module Name	BRAM _18K	DSP48E	FF	LUT
Point-level Filter	1	0	1147	851
Group-level Filter	1	3	2758	1893
Distance Calculator	0	3	13764	15429
Center Updater	7	3	14509	6544

From Table V, we observe that group-level filter takes more compute resources than point-level filter because group-level bound computations are more intensive than point-level bound computations, and both of them occupy fewer memory resources because of just caching a small number of distance bounds. Center updater takes more memory resources due to caching more intermediate results during updating. Distance calculators and center updaters consume most of the FFs and LUTs, since they demand such computing resources to handle regular but intensive distance computations for sufficient parallelism and overall throughput.

VI. Conclusion

In this paper, we present TiAcc that accelerates K-means on FPGAs with triangle inequality. We highlight TiAcc K-means in its novel multi-level triangle-inequality based filtering for distance computation reduction, pipeline decoupling for minimizing irregularity and workload imbalance, data streaming for efficient large dataset handling, and its unique way of combining algorithmic optimization and hardware design in a collaborative manner to fully stretch the design potentials. Extensive experiments show that TiAcc hardware

accelerator design is superior to previous research explorations and the state-of-the-art machine learning tools in terms of its considerable speedup, energy-efficiency, configurability, and adaptability across various datasets. Overall, TiAcc paves a promising way for efficient K-means acceleration on resource-constrained systems, such as FPGAs and embedded systems.

REFERENCES

[1] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[2] L. Portnoy, "Intrusion detection with unlabeled data using clustering," Ph.D. dissertation, Columbia University, 2000.

[3] S. Ray and R. H. Turi, "Determination of number of clusters in k-means clustering and application in colour image segmentation," in *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*. Calcutta, India, 1999, pp. 137–143.

[4] H. Ng, S. Ong, K. Foong, P. Goh, and W. Nowinski, "Medical image segmentation using k-means clustering and improved watershed algorithm," in *2006 IEEE Southwest Symposium on Image Analysis and Interpretation*. IEEE, 2006, pp. 61–65.

[5] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image segmentation using k-means clustering algorithm and subtractive clustering algorithm," *Procedia Computer Science*, vol. 54, pp. 764–771, 2015.

[6] A. Coates and A. Y. Ng, "Learning feature representations with k-means," in *Neural networks: Tricks of the trade*. Springer, 2012.

[7] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[8] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[9] M. Shindler, A. Wong, and A. W. Meyerson, "Fast and accurate k-means for large datasets," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2375–2383.

[10] P. Nair and K. N. Chaudhury, "Fast high-dimensional bilateral and nonlocal means filtering," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1470–1481, March 2019.

[11] L. Jing, M. K. Ng, and J. Z. Huang, "An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data," *IEEE Transactions on Knowledge & Data Engineering*, 2007.

[12] C. Elkan, "Using the triangle inequality to accelerate k-means," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*, ser. ICML'03, 2003.

[13] G. Hamerly, "Making k-means even faster," in *Proceedings of the 2010 SIAM international conference on data mining (ICDM)*. SIAM, 2010.

[14] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz, "Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, ser. ICML'15, 2015.

[15] J. Canilho, M. Véstias, and H. Neto, "Multi-core for k-means clustering on fpga," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[16] A. G. S. Filho, A. C. Frery, C. C. de Araujo, H. Alice, J. Cerqueira, J. A. Loureiro, M. E. de Lima, M. G. S. Oliveira, and M. M. Horta, "Hyperspectral images clustering on reconfigurable hardware using the k-means algorithm," in *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, Sep. 2003.

[17] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, June 2011.

[18] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, "Highly parameterized k-means clustering on fpgas: Comparative results with gpps and gpus," in *2011 International Conference on Reconfigurable Computing and FPGAs*, Nov 2011, pp. 475–480.

[19] C. Chung and Y. Wang, "Hadoop cluster with fpga-based hardware accelerators for k-means clustering algorithm," in *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, June 2017, pp. 143–144.

[20] T. Saegusa and T. Maruyama, "An fpga implementation of k-means clustering for color images based on kd-tree," in *Field Programmable Logic and Applications, 2006. (FPL)*. IEEE, 2006, pp. 1–6.

[21] Z. He, D. Sidler, Z. István, and G. Alonso, "A flexible k-means operator for hybrid databases," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 368–3683.

[22] Z. Lin, C. Lo, and P. Chow, "K-means implementation on fpga for high-dimensional data using triangle inequality," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 437–442.

[23] Q. Y. Tang and M. A. S. Khalid, "Acceleration of k-means algorithm using altera sdk for opencl," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 1, pp. 6:1–6:19, Sep. 2016.

[24] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable fpga architecture for k-mean clustering," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2013.

[25] Xilinx, "Vivado design suite," www.xilinx.com/products/design-tools/vivado.html.

[26] ——, "Zynq ultrascale+ mpsoc zcu102 evaluation kit," xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[29] G. Di Fatta and D. Pettinger, "Dynamic load balancing in parallel kd-tree k-means," in *2010 10th IEEE International Conference on Computer and Information Technology (ICCIT)*, 2010.

[30] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis & Machine Intelligence (PAMI)*, 2002.

[31] P. Sirait and A. M. Arymurthy, "Cluster centres determination based on kd tree in k-means clustering for area change detection," in *2010 International Conference on Distributed Frameworks for Multimedia Applications*, 2010.

[32] Y. Ding, X. Shen, M. Musuvathi, and T. Mytkowicz, "Top: A framework for enabling algorithmic optimizations for distance-related problems," 2015.

[33] Annoy, "Approximate nearest neighbors in c++/python optimized for memory usage and loading/saving to disk," https://github.com/spotify/annoy.git.

[34] J. Barnes and P. Hut, "A hierarchical o (n log n) force-calculation algorithm," *nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[35] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[36] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[37] D. Dheeru and E. K. Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[38] G. Chen, Y. Ding, and X. Shen, "Sweet knn: An efficient knn on gpu through reconciliation between redundancy removal and regularity," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 621–632.

[39] Nvidia, "kmeans clustering with multi-gpu capabilities," git@github.com:NVIDIA/kmeans.git.

[40] Nvidia, "Dense linear algebra on gpus," developer.nvidia.com/cublas. [Online]. Available: developer.nvidia.com/cublas

[41] Xilinx, "Pynq-z1 (zynq-7000-arm-fpga-soc)," https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/.

[42] Intel, "Xeon sliver 4110," ark.intel.com/content/www/us/en/ark/products/123547/intel-xeon-silver-4110-processor-11m-cache-2-10-ghz.html.

[43] Poniie, "Pn2000 watt meter," poniie.com/products/6. [Online]. Available: poniie.com/products/6

[44] Nvidia, "Jetson nano development kit," www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/.

[45] Nvidia, "Nvidia quardo p6000," www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/productspage/quadro/quadro-desktop/quadro-pascal-p6000-data-sheet-us-nv-704590-r1.pdf.

[46] Nvidia, "Nvprof," docs.nvidia.com/cuda/profiler-users-guide/index.html.