TiAcc: An Efficient Triangle-Inequality based K-means Hardware Accelerator Design on FPGA

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

***Abstract*—K-means is an important but computation-intensive algorithm for unsupervised learning. In this paper, we present TiAcc, an efficient Triangle-Inequality based K-means hardware accelerator design on FPGA. TiAcc highlights itself with an algorithm-hardware co-optimization strategy. Specifically, TiAcc leverages a novel Triangle-Inequality based filtering to reduce the costly exact distance computation, a data labeling approach to minimize computation irregularity, and an efficient hardware architecture design to fully exploit the pipeline and parallel processing capability of the FPGA. Moreover, TiAcc design parameterization minimizes the manual efforts in the arduous hardware design process and provides flexibility for users to optimize hardware design for balancing performance, energy and resource consumption.**

In the experiments, TiAcc K-means achieves considerable speedup (up to $9.81\times$) and significant energy efficiency (up to $151.10\times$) compared with Tensorflow K-means running on an 8-core server-grade CPU, and outperforms the state-of-the-art FPGA-based K-means with $5\times \sim 12.27\times$ faster speed and $24\% \sim 45\%$ higher throughput.

I. Introduction

K-means clustering is one of the most important and widely applied unsupervised learning algorithms, finding its strength in many machine learning application scenarios, 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 n points dataset with d dimension and k clusters, the complexity of standard K-means proposed by Lloyd [8] linearly depends on n, k, and d, making it hard to handle large-scale dataset with high dimensionality [9–11].

Two major directions have been explored to improve the performance of K-means. One category of methods focus on algorithmic optimization to reduce redundant computations, such as triangle-inequality based filtering [12–14] and KD-tree based methods [9–11]. While these methods demonstrate a striking power of computation elimination, they often introduce tremendous computation irregularities–diverging execution paths and operations among different data points, making the optimized algorithms hard to be efficiently deployed on modern, massively parallel hardware.

The second category concentrates on hardware-level optimization, which leverages the parallel computation capacity of modern hardware (e.g., multi-core CPUs, GPUs, FPGAs, and ASICs) to achieve performance boost [15–25].

Among these hardware optimizations, accelerating K-means on FPGA is one of the most promising directions due to its good balance between energy efficiency compared with CPU/GPU and developing flexibility compared with ASIC. Nevertheless, most of the existing FPGA designs [16, 17, 21, 24] fail to incorporate algorithmic innovations, resulting in limited overall performance improvement. In addition, these FPGA designs are often built and optimized for specific datasets, which lack of configurability and adaptability for a variety of datasets with different size and dimensionality [16, 17].

In this paper, we present TiAcc, an efficient Triangle-Inequality based K-means hardware accelerator design on FPGA. TiAcc leverages an algorithm-hardware co-optimization strategy, which combines the benefits from both the algorithmic optimization and hardware architecture properties to improve the overall design comprehensively. At the algorithm level, we achieve distance computation reduction by using a novel triangle-inequality based filtering without introducing much memory and computation overhead. At the hardware level, we fully exploit the parallel and pipeline processing capability of FPGAs by mitigating the computation irregularity introduced by the algorithmic optimization.

In addition, TiAcc minimizes the manual efforts in hardware design and implementation process, while maintaining design configurability and adaptability for different datasets and hardware devices. TiAcc takes the device and dataset specifications as the design inputs and applies algorithmic and hardware optimization. Moreover, TiAcc integrates the modern hardware design toolchains as its backend to generate the hardware implementation that can be directly deployed on the FPGA.

We implement TiAcc K-means on Xilinx Zynq UltraScale+ MPSoC ZCU102, an off-the-shelf FPGA board, and compare its performance with the state-of-the-art CPU, GPU, and FPGA K-means implementations over a set of datasets. The re-

markable energy-efficiency improvement, satisfying speedup, plus its simplified design and implementation process shape TiAcc a promising solution for K-means acceleration on FPGA.

Overall, we have made the following contributions in this paper:

- A novel multi-level triangle-inequality based filtering algorithmic optimization is introduced to remove redundant and expensive distance computations while minimizing the extra memory and computation overhead.
- A data-labeling approach is developed to mitigate the computation irregularity and simplify the hardware implementation of the advanced algorithmic design.
- A data batch streaming approach is proposed to manipulate large datasets efficiently, which reduces resources and energy consumption.
- A highly parameterized design is generated to efficiently support various hardware devices and datasets with different n, k and d.
- Experiments on a wide range of problem settings show that TiAcc is often superior to the state-of-the-art K-means implementations on ARM, multi-core CPU, GPU and FPGA in many aspects. For instance, TiAcc consistently excels Tensorflow K-means on an 8-core Xeon CPU with an averaged 5× speedup, and 88.6× better energy-efficiency.

The rest of paper is organized as follows: In Part II, we illustrate the background and the related work of the K-means algorithm optimization; In Part III, we present the overview of TiAcc; In Part IV, we detail our algorithmic design and optimization; In Part V, we discuss TiAcc accelerator hardware design; In Part VI, we show the experimental evaluation and analysis of TiAcc; In Part VII, we draw a conclusion for this paper.

II. Background and Related Work

K-means is probably one of the most intuitive, still extremely popular, unsupervised machine learning methods and is often used for the analysis of large datasets. The standard K-means algorithm was 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, such as GraphLab [26], OpenCV [27], Tensorflow [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 means 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-described two steps until all cluster centers get stabilized. In standard K-means, the point-assignment step takes the dominant part of computation time due to its higher complexity, i.e., $O(n \times k \times d)$ for computing the pairwise distances between each points and centroids, which leads to a major performance bottleneck when scaling up to handle large high-dimensional datasets [9–11].

To accelerate K-means, algorithmic innovations including KD-tree based [13, 29] and triangle-inequality based approaches [14, 25], have been proposed to reduce the number of distance computations in the point-assignment step. They have demonstrated significant performance improvement on software. However, KD-tree based method shows its weakness in high-dimension datasets ($d > 20$) [29], while the traditional triangle-inequality based approach reduces distance computation at the expense of introducing computation irregularity and memory overhead.

In TiAcc, we adopt triangle-inequality based methods but reduce its computation irregularity and memory overhead through *multi-level filtering* and *data labeling*. Before giving details of our designs, we will first give a brief introduction to *Triangle Inequality*, a simple *optimized K-means* based on it, and point out the challenges to directly apply these algorithmic innovations to FPGA.

A. Triangle Inequality

Triangle inequality states 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 given distances. By applying triangle inequality, the algorithm estimates the lower and upper bounds of $d(a, b)$ in the following way:

$$lb(a, b) \geq d(a, c) - d(b, c)$$

$$ub(a, b) \leq d(a, c) + d(b, c)$$

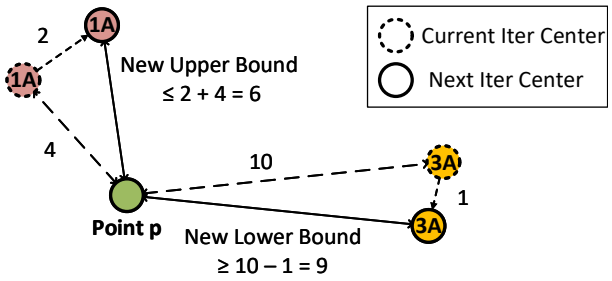


Fig. 1: Elkan's K-means approach.

B. Elkan's K-means

One work that demonstrates the advantages of this approach in K-means clustering is presented by Elkan et al. [12]. Elkan's K-means algorithm avoids exact point-to-cluster distance computation by using 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 Fig. 1. 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 lower bound of $d(q, 3A)$ is still larger than the upper bound of $d(q, 1A)$, i.e., $9 > 6$, it is impossible that $3A$ becomes the new closest cluster of p in the next K-means iteration. Therefore, the exact distance computation between p and $3A$ can be safely eliminated.

C. K-means on FPGA

Implementing K-means on FPGA has been widely studied by previous research [15–21, 24, 25]. However, most of the previous K-means FPGA designs only count on hardware-level implementation, resulting in limited overall performance improvement. In addition, previous designs are often built for specific datasets, which leads to limited applicability. For example, Hussain et al. [17] propose a 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 small number of points ($n = 2905$) and clusters ($k = 8$), making it hard to deal with large datasets with high dimensionality.

One of the prior work [25] closest to ours is proposed by Z. Lin et al. They apply a simplified Elkan's K-means on FPGA. And they achieve great speedup over the software K-means. However, the problems of their design are obvious: 1) Extra computations are required, such as the computation of center distance shift and distance bounds. It introduces a non-trivial cost when n, k and d are large; 2) Calculation irregularity is increased. Since the operations required for processing different points are largely diverged due to the distance computation filtering; 3) Extra memory space is required to maintain the distance bounds (e.g., $k - 1$ lower bounds per point). It prevents deployment on devices with limited on-chip memory.

In contrast, TiAcc gracefully addresses the above issues by reducing computation and memory overhead, and computation irregularity. To do that, TiAcc integrates the algorithmic and hardware design optimization effectively. More details are discussed in the following sections.

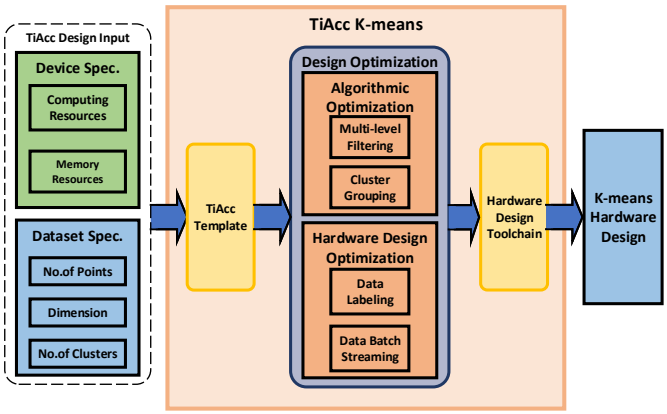


Fig. 2: TiAcc Design Workflow.

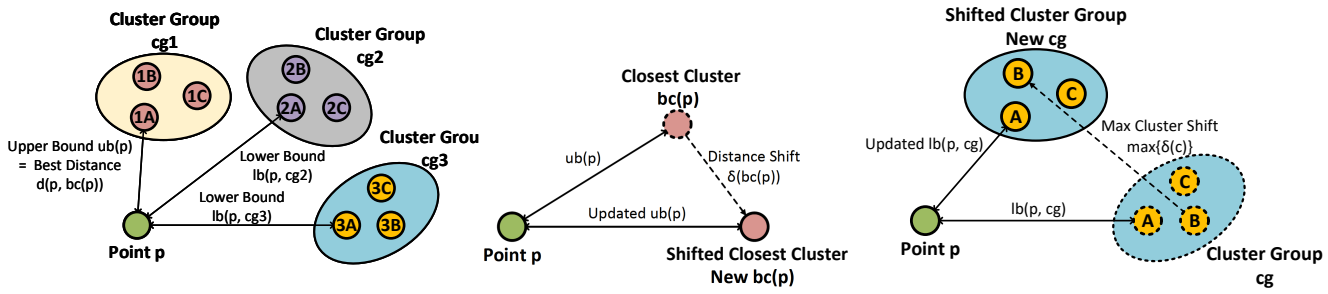
III. TiAcc Overview

As described in Fig 2, TiAcc takes the dataset specification and hardware device specification as the input. Then the key parameters from these input specifications are used to configure TiAcc built-in template. This template is fully parameterized and composed of a set of basic building parts (e.g., point-to-point distance computation, minimum distance selection) of K-means clustering. Then TiAcc applies two types of optimizations on customized hardware design templates from the last step. In the optimization phase, parameters such as the number of cluster groups k_c, data batch size B can be tuned for the trade-off between design performance and resource consumption. The output of design optimization becomes the input of hardware design toolchain (e.g., Xilinx Design Suite), which has been seamlessly integrated in the TiAcc design flow. Through the general procedure of the hardware design, the final output of TiAcc is a complete hardware design file, which can be directly delivered to users to deploy on the existing hardware accelerators.

IV. Algorithm-level Design

The overall TiAcc K-means algorithm-level design, as described in Algorithm 1, consists of several major steps. At the beginning of TiAcc, we initialize the clusters center at $line$ 1, by using initialization options such as "K-means++" [30] and "random" [31]. At $line$ 2, we group $initial\ clusters$ to prepare cluster groups for multi-level filtering in distance computation reduction. After that, we run standard K-means with $initial\ clusters$ and $points$ for only one iteration to build the initial distance bounds for each data point at $line$ 3.

TiAcc *multi-level filtering* optimization starts from second iteration, which covers $line\ 4 - 11$. It includes point-level filtering and group-level filtering at different levels of granularity. At the $line$ 5, point-level filtering removes points that will avoid all of their following distance computations in the current iteration. At the $line$ 6, group-level filtering further prunes the cluster groups of the points which pass the point-level filtering. At the $line$ 8, exact distances are calculated between the points and clusters in the cluster groups that pass



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

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

the above two-level filtering. Each point will be assigned to its closest cluster. Last, we update the clusters' center at $line$ 10. And 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 clustering result. More details of cluster grouping and multi-level filtering will be discussed in the following part of this section.

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 $P_{pass} = globalFilter(P, disBound)$;
6 $CG_{pass} = groupFilter(P_{pass}, disBound, CG)$;
7 **foreach** *p in* P_{pass} **do**
8 $assignedLabel[p] = argmin(p, CG_{pass}[p])$;
9 **end**
10 $updateCenter(C, assignedLabel)$;
11 **end**

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 Fig. 3a). There are several benefits of cluster grouping. First, TiAcc K-means only needs to maintain ($n \times k_c$, where $k_c << k$) distance bounds between each point and cluster groups for filtering computations, which largely reduces the memory and bounds computation overhead compared with Elkan's K-means ($n \times (k-1)$ distance bounds). Second, TiAcc reduces memory access overhead by accessing a group of clusters instead of an individual cluster each time.

Based on clustering 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 Fig. 3b and Fig. 3c, respectively. The update of 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 $lb(p, cg)$ generalize the lower bound of Elkan's K-means in the sense that it compute 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 $max\{\delta(c)\}(c \in cg)$, TiAcc extends the usage of Triangle Inequality from regular point-to-point to efficient point-to-group bound computations.

The idea sounds promising, but there are still two challenging questions to be addressed in cluster grouping: 1) How to build these cluster groups efficiently? 2) How many cluster groups to create for efficient filtering?

We use standard K-means to build the cluster groups out of the initial cluster centers through a few iterations (5 iterations is used throughout our evaluation). 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.

The number of cluster groups (k_c) provides a design knob for our algorithmic optimization that controls the trade-off between computation elimination and 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 bounds computation and memory overhead. The best choice of k_c could actually differ across different datasets. Details about choosing k_c for a dataset are presented in Section V-H and Section VI-F.

B. Multi-level Filtering

As the major algorithmic optimization of TiAcc K-means, multi-level filtering reduces the distance computations at different levels of granularity. Based on the distance bounds

from cluster grouping, a group-level filtering condition can be applied to remove cluster groups. Giving a point p, we remove the cluster groups, whose distances lower bounds is larger than distance upper bound of p. No distance calculation is needed between the p and all the clusters inside these cluster groups. In contrast, previous algorithmic optimizations (e.g., Elkan's K-means [12]) are based on fine-grained filtering to directly remove distance computations between each point and clusters. Such fine-grained approaches may eliminate more redundant distance computations, but they lose flexibility and efficiency while increasing the memory overhead.

To further optimize the filtering performance, we design a point-level filtering before the group-level filtering. Point-level filtering allows us to safely remove all distance computations of a certain point without further checking. For a data point p, TiAcc first gets the minimal value of its distances lower bounds. If this value is still larger than the current upper bound distance of p, which suggests there is no cluster that is closer, then all distance calculations p can be avoided in the current iteration. Otherwise, p will be forwarded to the next-stage group-level filtering. Our empirical study also shows that only applying point-level filtering can reduce more than 40% distance computations, which demonstrate the importance of point-level filtering optimization.

V. Accelerator Design

In this section, we present a detailed TiAcc K-means architecture design at the hardware-component level. As described in Fig. 5b, TiAcc K-means accelerator includes five major components: Block RAM, Point-level Filter, Group-level Filter, Distance Calculator, and Center Updater. In addition, TiAcc leverages an efficient *data batch streaming* strategy for high-performance data communication, a *data labeling approach* to simplify the processing pipeline, and a highly parameterized hardware design for parameter tuning and design space exploration.

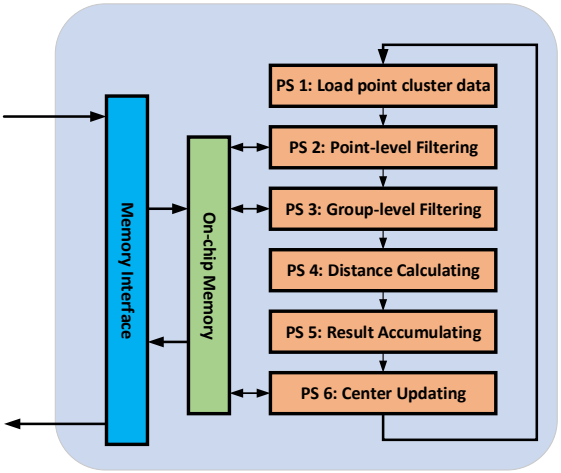


Fig. 4: TiAcc Processing Pipeline.

A. Processing Pipeline

FPGA-based hardware accelerator features its computation paradigm with massive pipeline and parallel processing. TiAcc finds its unique way of realizing triangle-inequality based K-means optimization on hardware through hardware-aware algorithm adaptation.

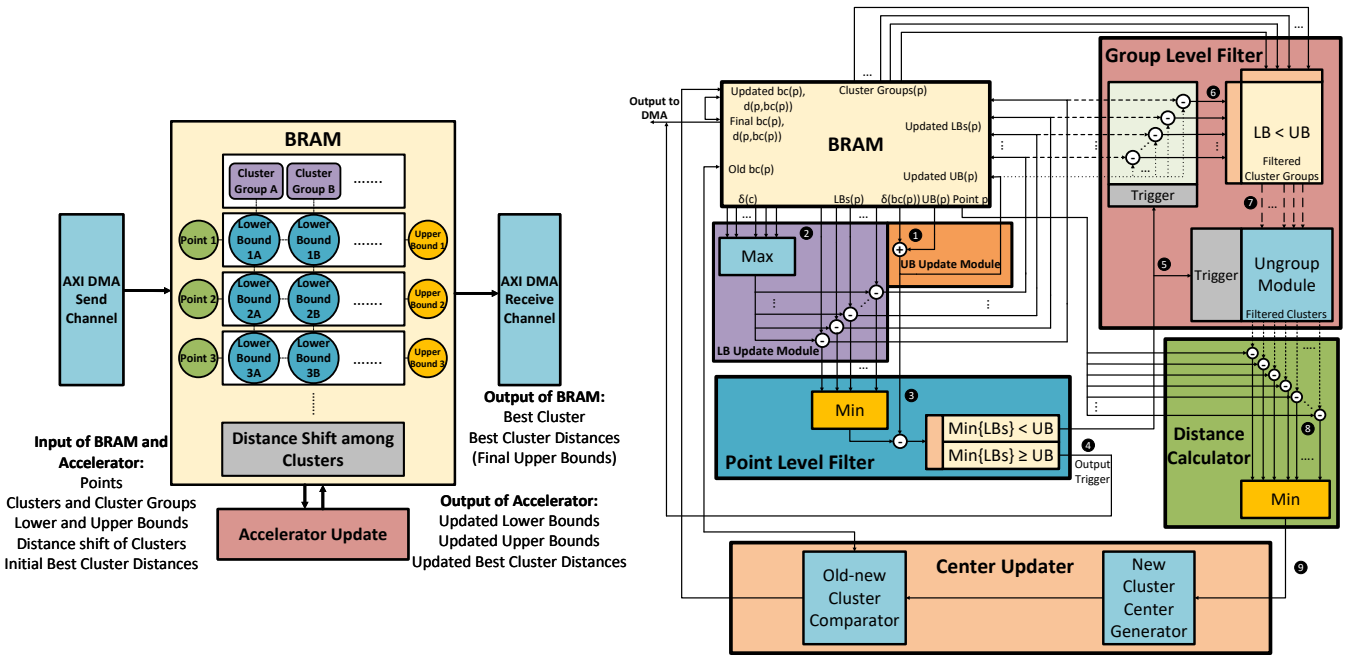
As shown in Fig. 4, the TiAcc K-means accelerator is organized as several continuous pipeline stages. The first stage (**PS 1**) is data loading. It loads a batch of data points and corresponding distance bounds from off-chip DRAM to on-chip BRAM. The second stage (**PS 2**) is the point-level filtering, which filters out the data points and avoids all of their following distance computations in the current iteration. After the data points pass the point-level filtering, the group-level filtering at the third pipeline stage (**PS 3**) will be activated to filter out more redundant distance computations of these points. The fourth stage (**PS 4**) is distance calculating, which only computes the distance between data points and clusters cluster groups which are not filtered out in the above step. The fifth pipeline stage (**PS 5**) is the result accumulating, which temporarily caches the clustering results from the last stage output and builds the new cluster centers. The last pipeline stage (**PS 6**) is center updating, which updates the old clusters' center. Then, we update the clusters' center distance shifting and the maximum distance shifting of the cluster-group center respectively.

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) 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. Since the on-chip memory of FPGAs is limited (KB $\sim$ MB), whereas datasets are large (GB). However, 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 on external memory, data batch streaming is superior in its minimized data access latency. Since the on-chip memory offers high access speed (sub-nanoseconds $\sim$ nanoseconds per data access), whereas external memory is slow ($\sim$50 nanoseconds per data access). In addition, the most frequently used data are cached at on-chip memory, such as the clusters and distance bounds, the frequency and cost of fetching data from external memory are reduced as well.

C. Point-level Filter

Point-level Filter labels points for distance computation reduction. If a point does not satisfy the point-level filtering condition, it will be labeled as PASS by Point-level Filter. Then this point will be forward to the next stage of the processing



(a) Data Communication.

(b) TiAcc Accelerator Architecture.

Fig. 5: Overview of TiAcc.

pipeline, where the Group-level Filter 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 marked as NOPASS, and all the following processing stages of this point will be avoided. Point-level Filter is the key component located at the gateway of the multi-level filtering processing pipeline.

The input of Point-level Filter relies on several distance bounds of the incoming point p: 1) $ub(p)$: the distance between p and its closest cluster in the last iteration; 2) $lb(p, cg)$: the distances between point p and each cluster group 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 point p and the closest cluster of p in the current iteration will be changed accordingly, as described in the Fig. 3b. We implement an **UB Update Module** in hardware design to realize the equivalent functionality in the algorithm, as shown in Fig. 5b (❶). After the upper bound update, the distance upper bound of p now becomes:

$$ub(p) + \delta(bc(p))$$

where $\delta(bc(p))$ is the distance shift of the closest cluster of p. Similarly, the distance lower bound between point p and a cluster group cg can be updated as follows:

$$lb(p, cg) - \max_{\forall c, c \in cg} \{\delta(c)\}$$

where $lb(p, cg)$ is the distance lower bound between p and cg. $\delta(c)$ is the distance shift of clusters inside cluster group cg, and we need to select a cluster c from all clusters inside cg which has the maximum value of distance shift in the last center update. The processing of cluster group shifting and lower bound updating are depicted in Fig. 3c and we implement a **LB Update Module** in hardware design, which is presented in Fig. 5b (❷).

Based on the above distance bounds, as implemented in Fig. 5b (❸), 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**, as shown in Fig. 5b (❹). In the distance bounds comparison, if we can have:

$$\min_{\forall cg, cg \in CG} \{lb(p, cg) - \max_{\forall c, c \in cg} \delta(c)\} \geq ub(p) + \delta(bc(p))$$

where CG is the set of all cluster groups, $\max\{\delta(c)\}$ is the maximum distance shift among all clusters inside cluster group cg, all following distance calculations of point p can be avoided, the closest cluster of point p will not change in the current iteration. The distance bounds comparator module will generate a signal to trigger the BRAM to output the closest cluster of point p. Otherwise, if the distance bounds

comparison has:

$$\min_{\forall cg, cg \in CG} \{lb(p, cg) - \max_{\forall c, c \in cg} \delta(c)\} < ub(p) + \delta(bc(p))$$

Distance Bounds Comparison Module will generate a signal to trigger the Group-level Filter to explore the potential redundant distance computation of point p at cluster-group level based on its distance lower bounds towards all cluster groups, as shown in Fig. 5b (❺).

D. Group-level Filter

Group-level Filter will be invoked only if it gets the trigger signal sent by Point-level Filter. Group-level Filter is used to reduce 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, as shown in Fig. 5b (❻), will be used to filter cluster groups, which are unnecessary to compute with in the current iteration. In the distance bounds comparison, if we can have:

$$lb(p, cg) - \max_{\forall c, c \in cg} \delta(c) < ub(p) + \delta(bc(p))$$

the corresponding cluster group cg is not filtered out and will be sent to an **Ungroup Module**, as shown in Fig. 5b (❼). Then all distances between the clusters in this group and point p are going to be calculated in the next stage of processing pipeline. Otherwise, if the updated lower bound is greater than or equal to the updated upper bound for point p and cluster group cg, then distance calculations between point p and all clusters inside cluster group cg can be avoided.

E. Distance Calculator

Distance Calculator, as shown in 5b (❽), computes the precise distance between points and clusters' center. The Distance Calculator will only be triggered if the incoming point and cluster groups has passed the multi-level filtering. Then the minimum of these calculated distances becomes the new closest cluster distance $d(p, bc(p))$ by passing a **Minimum Distance Selection Module**, as shown in Fig. 5b (❾). The new closest cluster distance becomes new distance upper bound of this point. Meanwhile, the closest cluster $bc(p)$ which corresponds to this new distance upper bound is updated as well.

Besides the basic functionality of the exact distance computation, TiAcc distance calculator is highly configurable to accommodate different distance metrics. TiAcc distance calculator provides some built-in optimized hardware implementations of the general distance metrics (e.g. Euclidean distance (Level-1), Manhattan distance (Level-2)). Besides, TiAcc distance calculator allows users to define their own distance metrics, such as weighted Level-1 distance ($|w_1x_1^2 + w_2x_2^2 + ... + w_Dx_D^2|$) and weighted Level-2 distance ($\sqrt{w_1x_1^2 + w_2x_2^2 + ... + w_Dx_D^2}$), where $w_1, w_2, .., w_D$ are not all equal.

F. Center Updater

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

G. Data Labeling

Data labeling is a hardware-aware algorithm adaptation, which simplifies the hardware implementation of TiAcc K-means. In the multi-level filtering, each data point is marked as either PASS or NOPASS. Similarly, giving a data point labeled as PASS in the point-level filtering, each cluster group of this point is marked as PASS or NOPASS in the group-level filtering. In the distance calculation, the expensive and precise distance calculation only happens between the points labeled as PASS in point-level filtering and the clusters from cluster groups labeled as PASS in the group-level filtering. By applying the data labeling, the irregularity of computation can be gracefully handled by building uniformed processing pipelines on hardware, which has much lower complexity in hardware implementation.

H. Configurability and Adaptability

Configurability and adaptability distinguish TiAcc from most previous work in this research direction. TiAcc can be easily generalized for different datasets (based on the dataset size(n), dimensionality(d), cluster size(k) and FPGAs (based on computing resources (e.g., number of FFs (F)) and on-chip memory size (BRAM (M))), which minimizes the manual efforts in arduous K-means FPGA design process.

In addition, TiAcc offers several options for the trade-off between design performance and hardware resources consumption. We explore several simple but effective principles for searching an optimal design efficiently.

- We should tune the design parameter (k_c) at algorithm level first. Then based on the optimized algorithmic setting, we further explore the parameters setting (Data Batch Size B) of hardware design by considering the hardware constraints.
- The algorithmic tunable parameter k_c is related with d, k, F and M. If the d is large, which means the point-to-point distance computation is more costly, k_c should be tuned towards large value to reduce these expensive distance computation. If the input dataset k is large, k_c should also be tuned towards larger value to maximize the possibility of removing potential redundant distance

computation. If the M of the hardware device is large, which means more cluster groups can be held at on-chip memory, k_c should increase as well. However, if the F of the hardware device is large, the k_c should be adjusted towards a smaller value to reduce the computation irregularity while increasing the computation parallelism. Since more distance calculator can be implemented to fully exploit the FPGA computing units and improve the distance computation efficiency.

- After the value of k_c has been optimized in algorithmic tuning, we explore the data batch size configuration which is related with the design parameter n, k_c, d and F. If the F of the hardware device is large, B should be tuned towards larger value, because more data points can be held at on-chip memory for faster access. If the n of dataset large, we should increase B correspondingly to minimize the data transferring between the off-chip and on-chip memory. If the k_c is large, we should decrease B to reduce the on-chip memory overhead, since more cluster groups and distance bounds occupy the memory. Similarly, if the d is large, we decrease B to avoid memory overhead.

In this paper, we apply a useful mathematical model for tuning design parameters k_c and B. In addition, TiAcc design parameterization and hardware generation minimize the efforts of building and evaluating the design, we try only $5 \sim 15$ design points on average for each dataset to locate the optimal hardware design points. For the future work, machine learning and deep learning based performance-resource model can be applied to assist the design space exploration.

VI. Evaluation

This section evaluates the efficiency of TiAcc K-means FPGA-based design on real-world datasets.

A. Experiment Settings

Experiment Tools: We implement and evaluate TiAcc K-means design on ZCU102 (Xilinx Zynq UltraScale+ MPSoC ZCU102 [32]) by using Xilinx Vivado Design Suite v2018.3. All the experiments are based on post-place-and-route results for analysis and comparison. ZCU102 is a general purpose evaluation FPGA board with -2L speed grade for rapid-prototyping based on the Zynq UltraScale+ MPSoC (multi-processor system-on-chip). ZCU102 has a quad-core ARM Cortex-A53, dual-core Cortex-R5 real-time processors, and a Mali-400 MP2 graphics processing unit 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).

Datasets: Six real-world datasets obtained from [33] are used in our experiment, which cover a wide range of data point size (5,000 $\sim$ 430,000) and dimensionality (3 $\sim$ 28). Details of these datasets are listed in Table I. Parkinsons Updrs dataset is composed of a range of biomedical voice measurements through symptom progression monitoring from Parkinson's disease patients; Letter Recognition dataset is the 26 capital letters in the English alphabet presented in black-and-white rectangular pixels; KEGG Metabolic Reaction Network dataset includes the Relation Network derived from Metabolic pathways; Skin Nonskin is the skin dataset collected by randomly sampling RGB values from face images; 3D road network has precise evaluation information from eco-routing and fuel/Co2-estimation routing algorithms. Electronic board reading is the values of real-time readings of power consumption in various domains such as commercial, industrial, etc.

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

In order to achieve a fair efficiency comparison between our implementation and other state-of-the-art counterparts on CPU, GPU, and FPGA, we use the same set of initial cluster centers for all implementations to ensure the exact same final clustering results. To build the cluster groups, we run standard K-means on the initial cluster centers for five iterations. 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:

$$\textbf{Speedup} = \frac{Time(Other)}{Time(TiAcc)} \tag{1}$$

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

$$\textbf{Energy\_Efficiency} = Speedup \times \frac{Power(Other)}{Power(TiAcc)} \tag{2}$$

B. Compared with Tensorflow K-means

In most recent years, some popular software tools have been released to facilitate the development of machine learning algorithms and applications. One of the most famous tools is Tensorflow [28], which is an open source library for numerical computation and efficient large-scale machine learning. In the comparison, we take a publicly available Tensorflow K-means [34] as the CPU and GPU baselines for our TiAcc K-means. More detailed comparisons in terms of performance and energy-efficiency are presented in the following part of this section.

*1) **Compared with CPU-based Implementation**:* In the comparison with the desktop-level CPU, a Tensorflow K-means CPU-based implementation is running on a server-grade 8-core 16-thread Intel Xeon Silver 4110 Processor (Clock Fre-

TABLE II: Performance and Energy Efficiency Comparison

Dataset.	Time. TiAcc (s)	Power. (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	**30.85**×	35.21×	**6.38**×	103.21×	**2.13**×	101.18×
Letter Recognition	0.006	4.64	**25.36**×	32.82×	**6.07**×	111.34×	0.71×	38.53×
Electr Board Read	0.010	4.06	**9.38**×	13.85×	**3.23**×	67.61×	0.52×	32.07×
KEGG Meta Net	0.016	5.52	**51.25**×	55.71×	**9.81**×	151.10×	0.91×	41.04×
Skin NonSkin	0.061	3.95	**9.01**×	13.70×	**2.52**×	54.30×	0.26×	16.70×
3D Spatial Network	0.143	4.16	**8.66**×	12.49×	**2.00**×	40.81×	0.22×	13.22×

quency: 2.1GHz, Power: 85W, Memory: 64GB DDR4) with all cores fully in use. The result in the Table II shows TiAcc K-means can achieve speedup up to 9.81× speedup on KEGG Metabolic Reaction Network dataset, and has an average 5× speedup across all six experimental datasets. We also observe that TiAcc K-means has more advantage in its speedup in the datasets with higher dimensionality such as Parkinsons Updrs ($d = 21$), Letter Recognition ($d = 16$), KEGG Meta Net ($d = 28$). Moreover, the significant energy efficiency (up to 151.10×, 88.06× on average) compared with Xeon CPU demonstrates TiAcc K-means promising applications on the most lower-power settings, such as IoT. In the comparison with the embedded processor, we run Tensorflow K-means on a Quad-Core ARM Cortex-A57 Processor (Clock Frequency: 1.9GHz, Power: 6W, Memory: 4GB LPDDR4) as a reference. The result of speedup listed in Table II shows TiAcc K-means is an alternative with much better performance (8.66× $\sim$ 51.25×, average 24.22×) and energy efficiency (12.49× $\sim$ 55.71×, average 27.30×) to the popular ARM based solution under the power-constrained settings.

*2) **Compared with GPU-based Implementation**:* GPU-based computation highlights itself with massive parallelization, significant performance, and high power consumption. In this experiment, Tensorflow K-means is running on Nvidia Titan Xp GPU (Clock Frequency: 1.58GHz, Power: 250W, Memory: 12GB GDDR5X @11 Gbps). Titan Xp is a high-end GPU with specialized design for high-performance machine learning and deep learning tasks. From Table II, TiAcc K-means shows its strength in the dimension of energy saving, which provides 13.22× $\sim$ 101.18× better energy efficiency compared with Titan Xp.

C. Compared with the state-of-the-art FPGA K-means

In comparison with the state-of-the-art FPGA K-means solutions, two reference designs [15, 21] have been used to comprehensively evaluate TiAcc in two different dimensions: speedup and throughput. In [15], the presented design demonstrates its significant speedup performance and we choose it as a speedup performance benchmark, while in [21], the presented design shows its strength in high throughput performance and we choose it as throughput performance benchmark.

Specifically, In [15], a multi-core K-means has been implemented on FPGA and a detailed design performance evaluation has been presented in terms of iteration time by using datasets with different n, k and d. For a fair comparison, we get their best speedup performance estimates under their default experiment setting and use the same dataset settings to evaluate our TiAcc K-means FPGA design. From Table III, we observe that TiAcc K-means design is $5\times \sim 12.27\times$ faster in executing each iteration. Specifically, the speedup performance on large cluster size ($k = \{100, 50\}$) is remarkable because cluster grouping and multi-level filtering of TiAcc K-means can reduce distance computation efficiently. In the high-dimension small cluster size setting ($d = 55$, $k = 6$), the speedup is lower than the other two cases. Because, k ($= 6$) is already small, and the cluster group size k_c should be smaller than k. This prohibits the multi-level filtering from removing more redundant distance computations. Besides, the high data dimension ($d = 55$) causes the very costly point-to-point distance computation, which also leads to longer execution time.

TABLE III: Performance Comparison with [15]

Dataset	Time(s) [15]	Time(s) TiAcc	Speedup Vs. [15]
$n = 10^6$, $k = 100$, $d = 15$	3.50	0.32	10.93×
$n = 10^6$, $k = 6$, $d = 55$	1.00	0.20	5.00×
$n = 10^6$, $k = 50$, $d = 20$	2.00	0.16	12.27×

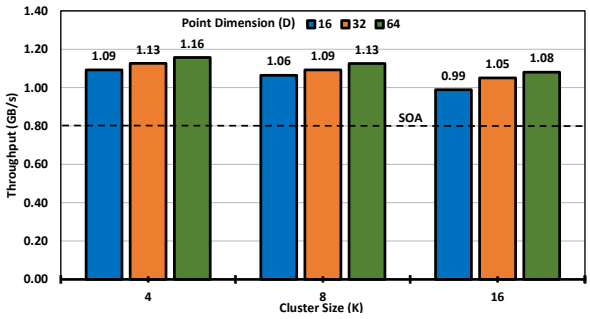


Fig. 6: Throughput Comparison with [21].

In [21], a flexible K-Means Operator for hybrid database has been developed on FPGA. The design throughput has been evaluated by using datasets with different d and k. In our evaluation, we cover $k = \{4, 8, 16\}$ and $d = \{16, 32, 64\}$ for comparison with [21]. The throughput result is presented in Fig. 6, the SOA horizontal dot line in the chart represents

their throughput performance in their paper. We can see that TiAcc K-means consistently outperforms the [21] with $23.6\% - 44.6\%$ in throughput improvement. Especially, in the high dimension ($d = 64$) setting, TiAcc demonstrates its design advantage in remarkably higher throughput and efficient distance computation reduction, whereas [21] is limited by applying standard K-means paradigm for guiding hardware design, which leads to a bottleneck in their design throughput improvement.

D. Optimization Benefits Analysis

In this section, we decompose the optimizations in TiAcc K-means to show their benefits in detail. We implement three designs (Design I: a standard K-means (Lloyd's Algorithm) CPU baseline; Design II: a CPU-based K-means equipped with TiAcc algorithmic optimizations (applying cluster grouping and multi-level triangle-inequality optimization); Design III: TiAcc K-means FPGA-based implementation. Based on these three designs, two experiments have been conducted to rationalize of TiAcc K-means incorporating these optimizations.

In the first experiment, 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 distance computations within the first twenty iterations of both TiAcc K-means and standard K-means, as shown in Fig. 7. To measure the distance computation reduction, we define the reduction ratio as:

$$\textbf{Reduction} = \frac{nc(standard) - nc(TiAcc)}{nc(standard)} \tag{3}$$

where the nc function 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.

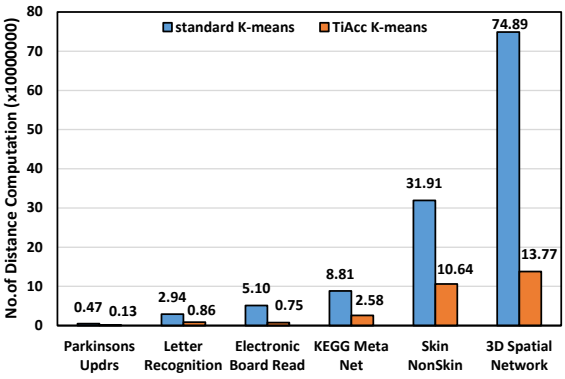


Fig. 7: Distance Computation Reduction.

In the second experiment, we put Design I, Design II and Design III together to illustrate the effectiveness of algorithmic optimization and hardware architecture design optimization of TiAcc K-means. From Fig. 8, we can see that only applying the algorithmic optimization (such as cluster grouping and triangle-inequality based multi-level filtering) and deploy it on CPU, we can get a speedup ($1.33\times \sim 3.62\times$) compared with standard K-means on CPU because of the algorithmic optimization (cluster grouping and multi-level filtering) in reducing expensive distance computations. In Design III, TiAcc combines the algorithmic optimization with the underlying hardware-level design and is deployed on ZCU102 FPGA board. We observe TiAcc achieves $5\times \sim 25\times$ compared with the algorithmic optimized K-means on CPU and $12.64\times \sim 80.63\times$ speedup with standard K-means on CPU, which demonstrates the potential of FPGA-based hardware for more efficient algorithm acceleration. Moreover, this result emphasizes the "core" of TiAcc K-means design, which does not rely on individual optimizations on either algorithm or hardware only but combines optimizations in a harmonious and efficient way for better overall performance.

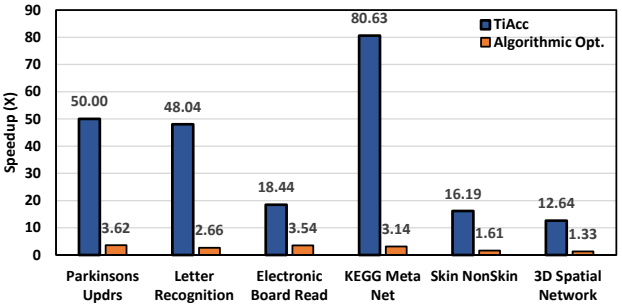


Fig. 8: Optimization Decomposition.

E. Study on 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 FPGA; 3) It can cover a wide range of data value and avoid the data overflow issues 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 calculator. 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)

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

is used as a metric for evaluating the clustering result of K-means clustering, which is defined as:

$$\textbf{WCSS} = \sum_{i=1}^{n} d(p_i, bc(p_i)) \tag{4}$$

We measure the percent error of the fixed-point and single-precision floating point solution with regard 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 datasets settings (d). From Table IV, we can see that in the same type of distance calculator, II (Single-precision floating point) always maintains the 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 overall power consumption and design complexity. This result rationalizes our design choice that uses the single-precision floating-point as the major data type in TiAcc design for the trade-off between accuracy and performance.

F. Study on Design Parameterization

Design parameterization allows users to customize their own K-means implementations based on several factors (e.g. expected performance, hardware resource, power budget). To demonstrate its effectiveness, we study two most representative design options, the number of cluster groups at the algorithm-level optimization and the number of the data batch size at the hardware-level design. In this study, we choose the Letter Recognition dataset ($n = 20000$, $d = 16$, $k = 70$) with median dataset size and dimensionality for design parameterization analysis. In the first study, we choose the different number of cluster groups for showing its impact on the performance of TiAcc K-means algorithm. In the Fig 9, we observe that within range of $1 \sim 8$ of the cluster group size, the increase of the cluster groups leads to the higher performance of TiAcc K-means. The reason behind this is that within that range of cluster group size, the performance improvement coming from distance computation reduction through cluster grouping and multi-level filtering outperforms the computation overhead increasing from maintaining more distance bounds. However, when the number of cluster groups exceeds 8, the overhead of distance bounds updating becomes prominent of the overall computation time.

In the second study, we explore the data batch size impact on TiAcc design performance. We customize TiAcc design for the Letter Recognition dataset, and the cluster groups number is set to 8 in order to exclude the effects of the cluster group size. We try different data batch size for searching a design point which can maximize the design performance (i.e. the minimized latency). As shown in Fig. 10, we can see that the latency Vs. data batch size follows the similar pattern of latency Vs. cluster group number in the above algorithmic optimization tuning. For the Letter Recognition dataset, at the number of the cluster groups equals 8 and the data batch size equals 32, the overall hardware accelerator design reaches a ”sweet” point with the minimized latency.

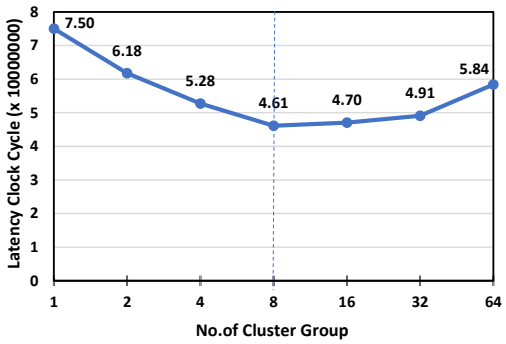


Fig. 9: Design Parameterization: Cluster Groups

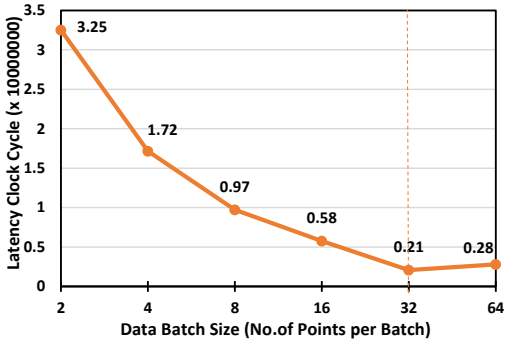


Fig. 10: Design Parameterization: Data Batch Size

VII. Conclusion

In this paper, we present TiAcc which targets at FPGAs for K-means clustering hardware acceleration. We highlight TiAcc K-means in its novel multi-level triangle-inequality based filtering for distance computation reduction, data labeling for hardware-aware algorithm adaptation, 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 design potentials. Extensive experiments show that our 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 under a wide range of hardware resource constraints over 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, pp. 561–580.

[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. [Online]. Available: http://papers.nips.cc/paper/4362-fast-and-accurate-k-means-for-large-datasets.pdf

[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*, no. 8, pp. 1026–1041, 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*, ser. ICML'03. AAAI Press, 2003, pp. 147–153. [Online]. Available: http://dl.acm.org/citation.cfm?id=3041838.3041857

[13] G. Hamerly, "Making k-means even faster," in *Proceedings of the 2010 SIAM international conference on data mining*. SIAM, 2010, pp. 130–140.

[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 - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 579–587. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045118.3045181

[15] J. Canilho, M. Vstias, 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, pp. 99–104.

[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, pp. 248–255.

[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'06. International Conference on*. IEEE, 2006, pp. 1–6.

[21] Z. He, D. Sidler, Z. Istvn, 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. [Online]. Available: http://doi.acm.org/10.1145/2964910

[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 (ISCAS2013)*, May 2013, pp. 1801–1804.

[25] 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)*, Aug 2012, pp. 437–442.

[26] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012. [Online]. Available: https://doi.org/10.14778/2212351.2212354

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

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[29] 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 and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, July 2002.

[30] 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.

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

[32] "Xilinx zynq ultrascale+ mpsoc zcu102 evaluation kit]." [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

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

[34] "tf.contrib.factorization.kmeansclustering." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/contrib/factorization/KMeansClustering#class_kmeansclustering