Hardware-Software Co-Design of K-Means Clustering on Zynq FPGAs via OpenCL

Eya JABALLI[1], Soufien GDAIM[2], Noureddine LIOUANE[3]
LARATSI Research Laboratory, ENIM, University of Monastir, Tunisia[1,3]
PRINCE Research Laboratory, ISITCom, University of Sousse, Tunisia[2]

***Abstract*—Clustering algorithms such as K-means are widely employed in machine learning and data analytics; however, their computational complexity becomes a significant limitation when applied to large-scale or latency-sensitive workloads. To overcome this challenge, this study presents a hardware-accelerated K-means clustering implementation using Field-Programmable Gate Arrays (FPGAs) within the Open Computing Language (OpenCL) framework. The proposed architecture leverages the fine-grained parallelism of FPGAs to accelerate the assignment phase, which is dominated by repetitive distance computations between data points and centroids. The proposed heterogeneous architecture accelerates only the assignment stage of the K-means algorithm on the FPGA, while the centroid update and convergence-control stages remain executed on the embedded ARM Cortex-A9 processor. Experimental results obtained on a two-dimensional synthetic floating-point dataset demonstrate significant performance gains over a CPU-only implementation, achieving speedups of up to 39× in execution time. The results also indicate consistent and favorable scalability trends across the evaluated synthetic two-dimensional configurations. To further assess the generalizability of the proposed approach, the accelerator was additionally validated on a real-world geospatial dataset, confirming consistent performance gains beyond synthetic benchmarks.**

***Keywords*—FPGAs; Hardware acceleration; high-level synthesis; OpenCL; K-means; reconfigurable hardware**

I. Introduction

Clustering plays an essential role in data mining research and applications and remains an active area of investigation in various disciplines, including computer science, data science, statistics, pattern recognition, artificial intelligence, and machine learning [1], [2]. The K-means algorithm is particularly significant among the various clustering algorithms because it is effective in discovering intrinsic patterns and revealing hidden structures in unlabeled datasets. Its use spans a wide range of application fields, including real-time data processing, IoT edge computing [3], [4], and big data processing environments [5], all of which are important for cluster classification for unlabeled data. Nevertheless, the increasing size and dimensionality of today's datasets pose serious computational challenges. With an increase in dataset size comes an increase in complexity of the clustering process, which causes slowdowns and performance bottlenecks that can be especially relevant in latency-sensitive or energy-limited real-time or edge computing environments. Performance bottlenecks such as these need to be solved to maintain the responsiveness and efficiency of a platform for real-time or edge processing. Multi-core central processing units (CPUs) and graphics processing units (GPUs) have been utilized for accelerating the computation involved in clustering [6]. While some performance improvement through parallelization can be achieved on these architectures, neither platform can match the power efficiency, deterministic latency, or real-time characteristics often required in edge computing applications, where device resources and energy consumption are at a premium. In this context, Field-Programmable Gate Arrays (FPGAs) have quickly gained traction as a viable option for hardware acceleration of clustering algorithms. FPGAs are reconfigurable architectures, which allows for fine-grained parallelism and deep pipelining, which gives developers the possibility to create hardware that is more closely aligned to the computational structure in algorithms. Moreover, unlike GPUs, which utilize a Single Instruction, Multiple Data (SIMD) that focuses on throughput rather than determinism, FPGAs execute tasks with low-latency and power-efficient results, which is particularly rewarding for real-time applications. Ultimately, the combination of flexible hardware paths, elasticity with scalability, and energy efficiency underscores the potential of FPGAs as an effective platform for hardware acceleration of unsupervised learning algorithms. The deployment of clustering algorithms on FPGAs introduces two significant performance bottlenecks. First, the latency of transferring data between the host and the FPGA device acts as an important constraint. In many cases, the communication pipeline between the CPU and FPGA is restricted by the use of synchronous data transfers, and this can negatively affect overall system throughput. This can be most detrimental to workloads that operate iteratively, such as clustering algorithms. A second contribution to performance limitations arises from an insufficient utilization of memory reads, because a sequence of memory operations in a clustering algorithm is irregular and often data-dependent. To improve the efficiency of the embedded CPU–FPGA implementation, the design adopts two implementation-level optimization strategies:

1) Host–kernel communication management: OpenCL command queues, non-blocking memory operations, and event-based synchronization are used to coordinate data migration, kernel execution, and result retrieval between the ARM processor and the FPGA kernel. These mechanisms reduce unnecessary host-side blocking and provide a structured execution flow for the heterogeneous implementation.
2) Burst-compatible memory access organization: Input observations and centroid arrays are stored in contiguous memory regions and accessed sequentially through AXI-based memory interfaces. This organization enables the HLS tool to infer burst-oriented memory transactions and reduces inefficient random memory accesses in the distance-computation kernel.

It is important to clarify the scope of this work. The proposed FPGA implementation accelerates the K-means assignment stage, specifically the nearest-centroid distance computation. The centroid update and convergence-control operations remain executed on the ARM processor. Therefore, the reported speedups refer to the heterogeneous implementation of the profiled K-means workload under the evaluated configurations, not to a fully hardware-implemented K-means pipeline. The current implementation evaluates two-dimensional floating-point datasets generated synthetically. The main contributions of this work are summarized as follows:

- An OpenCL-based heterogeneous CPU–FPGA implementation that accelerates the assignment stage of K-means on the FPGA while preserving the centroid update and convergence-control operations on the embedded ARM processor.
- A profiling-guided hardware/software partitioning approach that identifies the nearest-centroid distance computation as the main acceleration candidate.
- A lightweight FPGA kernel design using local centroid buffering, loop-level parallelism, and burst-compatible AXI memory access for accelerating the distance-computation stage.
- A host–kernel communication flow based on OpenCL command queues, non-blocking memory operations, and event-based synchronization to coordinate data migration and kernel execution.
- An embedded CPU–FPGA performance evaluation using synthetic two-dimensional floating-point datasets with varying numbers of data points and clusters, complemented by FPGA resource utilization analysis on the XC7Z020 device and validation on a real-world geospatial dataset to confirm the generalizability of the proposed accelerator.

The remainder of this study is organized as follows: Section II reviews the related work, including previous FPGA-based hardware acceleration approaches, K-means algorithm optimizations, and OpenCL programming for FPGAs. Section III presents the necessary background, including a detailed overview of the K-means clustering algorithm. Section IV describes the proposed methodology and the transition from the baseline implementation to the OpenCL-based FPGA-accelerated version. Section V presents the experimental setup, performance evaluation, and comparative analysis with existing works, followed by a discussion of the limitations. Finally, Section VI concludes the study by summarizing the main findings and outlining directions for future research.

II. Related Work

Due to the growing need to process large-scale datasets efficiently, scientific research has recently engaged in hardware acceleration for machine learning algorithms. Effective training of various machine learning algorithms, such as K-means, requires substantial computational resources and time. Therefore, these algorithms are often executed on FPGAs to take advantage of the power of reconfigurable hardware and accelerate their processing [7].

A. FPGA-Based Hardware Acceleration

FPGAs stand as a highly promising option for executing high-performance computations due to their remarkable attributes. They present a good balance between performance and power consumption, coupled with an interesting level of flexibility that enables users to configure these architectures to implement a customized application. The advantage of reconfigurability in FPGAs allows them to be fine-tuned to match the demands of the target computation. Recently, many studies have been published discussing software and hardware optimization and implementation methods of hardware accelerators using FPGAs [8], [9], [10], [11], [12]. Among them, our earlier work [8] introduced a preliminary approach, which the current study significantly extends and refines.

In addition, FPGAs have been presented as a mid-point solution providing a trade-off of GPU performance and ASIC efficiency while maintaining high flexibility and reconfigurability. They have been broadly recognized in the literature as being well-suited to accelerating machine learning. A review of more than 120 FPGA implementations of neural network accelerators demonstrates the wide generality of applications of FPGAs to diverse deep learning architectural styles, i.e., CNNs and RNNs, including ResNet-2 and LSTM [13]. The study also proposes a set of benchmark metrics and optimization techniques that cross many design classes with a useful starting point for benchmarking and optimizing accelerator performance.

As stated above, the effective utilization of machine learning methods demands a substantial amount of processing power. Even conventional multi-core CPUs are burdened by the computational complexity of these algorithms. In light of this, [14] proposed a reconfigurable architecture to deploy machine learning algorithms by implementing a Neural Network in an FPGA. The final results were compared with a Raspberry Pi (a conventional CPU) implementation. The proposed FPGA implementation architecture is 33646 times faster than the traditional CPU or core-based method.

In the ESA-funded CloudScout project, a bespoke FPGA-based accelerator was compared with the Intel Myriad 2 Vision Processing Unit (VPU) for on-board inference of a Convolutional Neural Network (CNN) for hyperspectral cloud detection in a CubeSat context [15]. The research showed that FPGA integration provided shorter inference time and greater flexibility than VPU, foreseeing the promise of FPGAs in resource-limited environments such as space. While the FPGA took more power and development time, its better performance and space qualification made it a suitable candidate for long-duration Low Earth Orbit (LEO) and interplanetary missions. These results demonstrate the applicability of FPGAs to a wide range of applications demanding effective on-board processing.

In addition to their benefits in deep learning applications, FPGAs have effectively sped up traditional machine learning algorithms. For instance, a recent study described a parameterized and scalable FPGA-based hardware accelerator for an embedded systems Support Vector Machine (SVM) classifier that relies on convex optimization [16]. With high classification accuracy, the architecture offers up to 79× acceleration over its software counterpart, allowing real-time operation and scalability to datasets of various sizes. This work validates

the feasibility of FPGAs as a general-purpose platform for machine learning acceleration by showing their adaptability and efficiency in accelerating data-intensive machine learning computations.

B. OpenCL for Heterogeneous Computing

Considerable progress in parallel computing has been made possible by the advent of heterogeneous computing systems. Given this, OpenCL has attracted attention as a standard that facilitates task-parallel and data-parallel heterogeneous computing on a variety of processing platforms, such as CPUs, GPUs, DSPs, and other coprocessor accelerators [17]. Its flexibility lies in the way it manages arithmetic-intensive data-parallel workloads on these various processing units. OpenCL's flexibility and efficiency for heterogeneous computing have been the subject of numerous research studies. Paper [18] provides an overview of the key architectural features of current microprocessor designs and outlines the OpenCL programming model as a standard developed for these architectures. Furthermore, OpenCL has been used to accelerate computationally demanding methods, such as Convolutional Neural Networks (CNNs). A study in [19] suggested using OpenCL to accelerate CNNs on Xilinx ZYNQ-7000 FPGAs. This approach achieved speedups of up to 10× over CPU implementations while maintaining comparable accuracy on well-known datasets such as MNIST and CIFAR-10. This illustrates OpenCL's ability to enable effective hardware acceleration without the complexity of conventional HDL design. It also supports the deployment of machine learning models for platforms with limited resources and rapid prototyping. Yongbon *et al.* have also presented a study analyzing the efficacy of OpenCL to speed up machine learning applications [20]. Their study's main goal was to create an implementation of Darknet, a deep learning-based object identification framework, that is hardware-compatible with universal accelerators. Darknet's limited utility resulted from its initial support of only NVIDIA CUDA for processing acceleration. Getting over the limitation and beating the original CUDA version is the primary objective of this research. The performance of OpenCL-Darknet was evaluated on two different AMD hardware setups: an AMD Radeon RX560 with OpenCL 1.2 and an AMD R7-integrated APU with OpenCL 2.0. For this assessment, the VOC 2007 dataset was used. Furthermore, they performed a performance comparison between OpenCL-Darknet and the original Darknet using an NVIDIA GTX 1050 with CUDA 8.0 and cuDNN 6.0. The findings revealed that compared to CUDA and cuDNN, OpenCL-Darknet exhibited a performance discrepancy of 2.5x to 4.4x for YOLOv2 and of 3.2x to 6.4x for TinyYOLO.

C. FPGA-Based Hardware Acceleration of K-means Algorithm with OpenCL

Tang *et al.* provides an exploration of the K-means algorithm via HLS of OpenCL code [21]. Due to inefficient memory bandwidth use, the initial implementation of the sequential baseline K-means algorithm performs poorly on the FPGA. The optimal approach relied on two kernels that utilized the FPGA's local memory. In the best scenario, speedups of up to 21× are obtained for implementations on an Altera Stratix V FPGA in comparison to an Intel Xeon Hexa-core processor.

In [22], a hybrid implementation of the K-means is presented to compare the speedup to an ARM CPU implementation. The proposed implementation carries out only some of the steps of the algorithm; the software carries out the remaining steps. The goal of this hybrid method is to reduce FPGA area overhead while increasing processing performance. The FPGA circuits are used to compute the similarity distance metric, which is based on the Manhattan distance, and to group data points based on their closest centroid. Moreover, the centroid values are updated by the software. Compared to the software version of the ARM processor, the experimental results reveal a speedup of about 10×.

In [23], the presented approach used High-Level Synthesis to test 10 distinct OpenCL implementations of the K-means algorithm on an FPGA platform. In addition to evaluating memory optimization strategies, including on-chip buffering and vectorization, the study investigated NDRange and task kernel models and examined performance across a variety of datasets and configurations. Their implementation showed energy savings of up to 80% and a speedup of increase to 1.54× over a high-performance CPU baseline.

III. K-means Clustering

The K-means clustering algorithm is one of the most popular unsupervised learning methods to partition datasets into unique groups by similarity. K-means has been used successfully across many research areas, including image processing [24], market analysis [25], data processing [26], medical image segmentation [27], risk evaluation [28], [29], etc. K-means is a partitional clustering algorithm, meaning that it partitions a dataset into a pre-defined number of non-overlapping subsets, or clusters. The core principle of K-means is to cluster data points to maximize intra-cluster similarity (data points are grouped together as closely as possible) and minimize inter-cluster similarity (data points in different clusters are distinguishable). In general, this is done by minimizing the sum of squared distances (errors) from an individual data point to the centrally located (mean) representative of its assigned cluster, which is where the K-means algorithm gets its name. The K-means procedure uses iteration where it initially provides a set of cluster centers and then performs alternating steps, where in each step the algorithm first assigns each point to the nearest cluster center, and then the next step updates each cluster center to be the mean of the points that were assigned to it until reaching convergence. Convergence is usually reached when either the assignments change no longer, or when the error is reduced to less than some previously assumed degree.

A. Problem Formulation

Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the i-th data point in a d-dimensional space, the goal of K-means is to partition $\mathbf{X}$ into k clusters $\{C_1, C_2, \ldots, C_k\}$. The optimization objective is to minimize the within-cluster sum of squared distances (WCSSD):

$$\mathcal{J} = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|^2 \tag{1}$$

where, μ_j denotes the centroid of cluster C_j, calculated as:

$$\mu_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \tag{2}$$

The optimization alternates between assigning data points to clusters and updating centroids until convergence.

B. Algorithm Steps

The K-means algorithm involves the following steps:

- Initialization: Choose k initial centroids, $\mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_k^{(0)}$. Initialization methods include random selection or heuristic techniques like K-means++.
- Assignment Step: For each data point $\mathbf{x}_i$, determine its cluster assignment based on the nearest centroid:

$$c_i^{(t)} = \arg\min_j \|\mathbf{x}_i - \mu_j^{(t)}\|^2 \tag{3}$$

Alternatively, this can be represented using a binary indicator matrix $\mathbf{r}_{ij}$:

$$\mathbf{r}_{ij} = \begin{cases} 1, & \text{if } \mathbf{x}_i \text{ is assigned to cluster } j, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

- Update Step: Recalculate the centroids for each cluster based on the current assignments:

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{N} \mathbf{r}_{ij} \mathbf{x}_i}{\sum_{i=1}^{N} \mathbf{r}_{ij}} \tag{5}$$

- Convergence Check: Repeat steps 2 and 3 until the number of observations whose cluster assignment changes between consecutive iterations falls below a predefined threshold.

C. Computational Complexity

K-means's computational complexity is affected by the number of data points N, the number of clusters k, the dimensionality d, and the number of iterations T. Each iteration requires $O(Nkd)$ computations for the assignment step and $O(kd)$ for updating centroids, leading to an overall complexity of:

$$O(TNkd)$$

While this complexity is manageable for small datasets, it becomes prohibitive for large-scale data due to the iterative nature of the algorithm and the repeated distance computations.

Despite its simplicity, the K-means algorithm has several limitations, particularly its low scalability as the dataset size increases. The computational cost of the algorithm, mainly due to distance calculations, increases rapidly as the number of data points and dimensions increases. However, in hardware implementations, these distance calculations, which dominate the execution time, can be efficiently accelerated using FPGAs. By offloading these intensive calculations to FPGAs, computational bottlenecks are reduced, leading to significant improvements in execution time. This hardware acceleration approach is particularly beneficial when dealing with large, high-dimensional datasets or real-time applications, as explored in subsequent sections of this study.

IV. Adopted Methodology

The proposed methodology is divided into three principal phases, as illustrated in Fig. 1. The first phase focuses on the design process of the overall application architecture, which describes the functional and structural components needed for the implementation. The second phase concentrates on the development of the OpenCL kernel, where computationally intensive tasks are defined and optimized for execution on the FPGA. The final phase involves establishing efficient communication between the host processor and the FPGA kernel through the use of OpenCL APIs, ensuring seamless data transfer and coordination between software and hardware components.

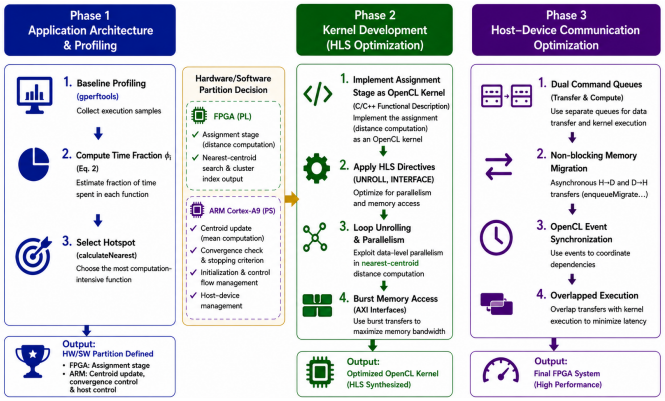

Fig. 1. Overview of the adopted methodology.

A. Application Architecture Design

Before embarking on the design of a hardware-accelerated application, it is imperative to conduct thorough architectural planning. This initial planning phase encompasses the development of an implementation plan, specifying the level of computational parallelism to achieve, and selecting which software tasks should be migrated to hardware as custom kernels. The process begins with a baseline profiling of the application's non-optimized CPU version. Profiling tools (e.g., GNU `gprof`) are used to extract the execution time of each function T_i^{CPU}. The fraction of time consumed by a given function i is then computed as:

$$\phi_i = \frac{T_i^{\text{CPU}}}{\sum_j T_j^{\text{CPU}}} \tag{6}$$

where, ϕ_i represents the relative contribution of function i to the total runtime. Functions with large ϕ_i values are natural candidates for FPGA acceleration.

The second step involves identifying the portions of the application that will experience the most significant performance improvement through the acceleration of relevant functions

in hardware. This selection process requires a critical evaluation of each function's computational load and its potential efficiency gains when implemented in hardware. Not every function will provide a meaningful acceleration by implementing the function in hardware, and it can conceivably cost the application overall performance. Therefore, the analysis in this iterative process is critical to ensure that hardware resources are optimized.

B. OpenCL Kernel Development

The second stage of the suggested process focuses on using OpenCL to convert the application's computationally demanding components into effective hardware kernels. In order to get notable performance improvements through hardware acceleration on FPGA platforms, this step is essential. Task selection, kernel implementation, and kernel optimization are the three primary sub-stages that comprise this phase. The first step in this phase is to identify functions or code segments that consume the most computational resources. These segments typically involve nested loops, large matrix computations, or repetitive arithmetic operations. Tools such as gprof, Google Performance Tools (gperftools), or platform-specific profilers such as Vitis Analyzer can be used to detect these hotspots. Once identified, the selected tasks are isolated and analyzed for suitability for parallelization and offloading. Priority is given to functions that operate on large datasets, exhibit high levels of parallelism (data or task parallelism), and have minimal dependencies between iterations. After task selection, the identified code blocks are rewritten as OpenCL kernels, which are functions that can be executed in parallel on the FPGA. The kernel design follows a load-compute-store structure:

- Load: Data are transferred from the host to the FPGA.
- Compute: The kernel performs the required operations on the hardware part.
- Store: The results are sent back to the host.

The OpenCL kernel is described at a high level using C/C++-based functional specifications. High-Level Synthesis (HLS) tools, such as Vitis HLS, automatically translate these functional descriptions into Register Transfer Level (RTL) implementations suitable for FPGA synthesis. The execution time of a kernel K on the FPGA can be expressed as:

$$T_K^{\text{FPGA}} = T_{\text{load}} + T_{\text{compute}} + T_{\text{store}} \tag{7}$$

where, T_{load} and T_{store} correspond to the data transfers between the host and the FPGA, and T_{compute} represents the kernel computation time. To enhance performance, HLS optimization directives (pragmas) are applied to control loop unrolling, pipelining, and data access patterns, thereby reducing T_{compute}. Additionally, techniques such as burst transfers, buffer reuse, and memory coalescing are employed to minimize T_{load} and T_{store}, ensuring efficient utilization of both computational and memory resources.

C. Code Adaptation: Host-Kernel Communication

Effective communication between the FPGA kernel and the host application is a crucial component of performance improvement in hardware-accelerated systems. While the host application coordinates overall execution, controls data transfers, and handles input/output activities, the kernel code, which is executed on the FPGA, carries out computationally demanding tasks. This section describes the methodology used to integrate the host and kernel components, ensuring efficient communication within the implementation of the K-means algorithm. As presented in Fig. 2, the host application is responsible for:

- Allocating memory buffers for input data (e.g., observations and cluster centroids) and output data (e.g., cluster assignments).
- Transferring data between the host and the FPGA device.
- Launching the kernel on the FPGA and monitoring its execution status.
- Retrieving results from the kernel and performing any post-processing required.

For this work, the host code was written in C/C++ and interfaced with the FPGA kernel using the OpenCL framework, which offers an abstraction layer that facilitates communication between the host and the kernel through a unified programming model.

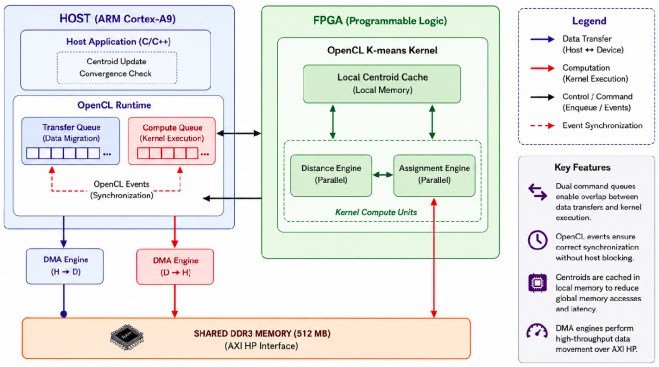

Fig. 2. OpenCL framework overview.

V. Performance Evaluation

A. Experimental Setup

The proposed heterogeneous K-means accelerator was implemented on a Xilinx ZedBoard featuring a Xilinx Zynq-7000 XC7Z020-CLG484-1 SoC integrating a dual-core ARM Cortex-A9 processing system (PS) and programmable logic (PL). The FPGA design was developed using the Xilinx OpenCL development flow under PetaLinux, while the software-only baseline was executed entirely on the embedded ARM Cortex-A9 processor. Consequently, all reported speedup values correspond to comparisons between the heterogeneous CPU–FPGA implementation and the ARM processor executing the same K-means algorithm under identical experimental conditions. The input data consisted of randomly generated

floating-point values. Experiments were conducted on datasets ranging from 4,000 to 30,000 data points with varying numbers of clusters, and performance was evaluated using kernel execution time measured in milliseconds. To further validate the proposed accelerator on real-world data, additional experiments were conducted using the SimpleMaps World Cities dataset. The hardware and software configuration used throughout the experiments is summarized in Table I.

TABLE I. HARDWARE AND SOFTWARE CONFIGURATION OF THE FPGA-BASED IMPLEMENTATION.

Parameter	Specification
FPGA board	Xilinx ZedBoard
FPGA device	XC7Z020-CLG484-1
Processing System	Dual-core ARM Cortex-A9
Operating system	PetaLinux
Development environment	Vitis Unified ide 2021.2
HLS tool	Vitis HLS 2021.2
OpenCL version	OpenCL 1.2
Target clock period / frequency	10 ns / 100 MHz

1) Selection of the acceleration candidate: The acceleration candidate was selected based on the computational structure of the K-means algorithm and preliminary profiling of the CPU implementation. In each iteration, the assignment stage compares every data point with every centroid. For a dataset with N points, k clusters, and dimensionality d, this stage requires $O(Nkd)$ distance computations. Therefore, the nearest-centroid computation is expected to dominate the execution time, especially when N or k increases.

In the implemented software version, this computation is performed by the `calculateNearest` function, which iterates over the centroids and computes the Euclidean distance between each data point and each centroid. For this reason, `calculateNearest` was selected as the main candidate for FPGA acceleration. Preliminary profiling confirmed that this function was the main hotspot; however, the profiling output is used only as qualitative guidance and not as an exact timing breakdown.

2) OpenCL kernel development: The selected `calculateNearest` function was implemented as an OpenCL kernel targeting the programmable logic of the Zynq device. The kernel computes the nearest centroid for each input observation and stores the corresponding cluster index in the output array. The centroid update and convergence-control operations remain executed on the ARM processor. This implementation incorporates various HLS directives that were strategically employed to guide the HLS tool during synthesis. The following directives were utilized to interface with the memory and control units:

a) Memory interfaces: `#pragma HLS INTERFACE m_axi` directives were employed for the input and output arrays. These directives define the memory interface for the input and output arrays, allowing seamless communication between the hardware accelerator and the external memory.

b) Control interfaces: `#pragma HLS INTERFACE s_axilite` directives were utilized for scalar control variables. These directives facilitate the creation of lightweight control registers for efficient communication between the host (Processing System PS) and the hardware accelerator (Programmable Logic PL).

Algorithm 1 K-means Kernel Algorithm

1.5em 0.75em
Input: `ptClusX`, `ptClusY`, `obsX`, `obsY` `numObservations`, `k`, `minD`
Output: `indices`
Initialize local arrays `localptClusX` and `localptClusY`
Copy cluster coordinates into local arrays
for $i = 0$ **to** `numObservations` $- 1$ **do**
    `localMinD` $\leftarrow$ `minD`
    `bestCluster` $\leftarrow 0$
    for $c = 0$ **to** `k` $- 1$ **do**
        `#pragma HLS UNROLL`
        $d_x \leftarrow$ `localptClusX[c]` $-$ `obsX[i]`
        $d_y \leftarrow$ `localptClusY[c]` $-$ `obsY[i]`
        `localDistance` $\leftarrow d_x^2 + d_y^2$
        if `localDistance` $<$ `localMinD` **then**
            `localMinD` $\leftarrow$ `localDistance`
            `bestCluster` $\leftarrow c$
        end if
    end for
    `indices[i]` $\leftarrow$ `bestCluster`
end for

In addition to the interface directives, other optimization directives were considered to enhance the algorithm's performance. The incorporation of the loop unrolling pragma (`#pragma HLS UNROLL`) emerged as a crucial strategy. Specifically, the loop unroll pragma was strategically applied to the loop responsible for computing the distance between data points and cluster centroids, as mentioned in the pseudocode below. This directive instructs the HLS tool to generate multiple copies of the loop body to reduce loop control overhead and enhance parallelism. It is particularly impactful in scenarios where the loop iterations are independent, as is often the case in distance calculations. By unrolling the loop, the compiler is empowered to exploit pipeline parallelism and optimize the overall execution time. K-means kernel algorithm is presented in Algorithm 1.

The independence of the data and the loop iterations served as the motivation behind the implementation of the (`#pragma HLS UNROLL`) directive within the distance computation method. Multiplication and accumulation operations are carried out on separate data elements in each iteration without creating dependencies on earlier or later iterations. This characteristic makes the loop highly amenable to unrolling, as the compiler can generate multiple parallel hardware instances that execute simultaneously. This takes advantage of the arithmetic resources on the FPGA giving higher throughput and less overall latency. Utilizing unrolling should be advantageous in clustering algorithms where distance calculations apply repeatedly to massive datasets, and this key aspect is a performance bottleneck.

3) Optimizing data communication between host and device: In hardware-accelerated systems, optimizing the data transfer between the host (CPU) and the FPGA is critical for achieving high performance. Therefore, this study seeks to improve the communication pipeline with asynchronous data transfers and non-blocking memory transactions. These two approaches are critical for reducing idle times and processing workloads concurrently between the host and the FPGA. We utilize asynchronous data transfers via two independent command queues in OpenCL: (`transfer_q`) for memory transfers, and (`compute_q`) for kernel executions. Both com-

mand queues are instantiated in out-of-order execution mode, allowing for overlapping transfers and kernel execution. This design reduces the waiting times for data to arrive before processing and ensures that the FPGA is continuously utilized. In this configuration, the transfer of the input data from host memory to the device memory and the kernel execution in the FPGA memory are non-blocking operations; this enables the accelerator to begin the computation as soon as it has enough of the required data to start, while the rest of the data transfers to host occur simultaneously. This structure will maximize the overall throughput of the system, since both computation and data movement will execute concurrently. To improve performance, we leverage non-blocking memory transactions. The OpenCL call - `enqueueMigrateMemObjects()` - facilitates the migration of data between host and device. Data transfer is performed asynchronously so that the host can do further work concurrently with the Data transfer. OpenCL events synchronize the different stages of the process without blocking. This allows the host to start executing the kernel as soon as the required data points have been received. Also, the host can continue reading the output data while the device writes to memory without waiting for all operations to complete. These techniques are essential for reducing idle time and ensuring that the data movement and computation phases are not bottlenecked by synchronization or waiting for data.

B. Experimental Results

1) Convergence criterion: To ensure a fair performance comparison, both the software-only and heterogeneous CPU–FPGA implementations were evaluated using the same K-means convergence criterion and termination condition. The algorithm iteratively alternates between the centroid recomputation and cluster assignment stages until convergence is achieved. Convergence is assessed by counting the number of observations whose cluster assignments differ between two consecutive iterations. Let $c^{(t)}$ denote the number of reassigned observations at iteration t. The iterative process terminates when the number of changed assignments falls below or equals a predefined threshold proportional to the dataset size N:

$$c^{(t)} \leq \left\lfloor \frac{N}{10000} \right\rfloor \tag{8}$$

Consequently, both implementations perform the same number of K-means iterations and satisfy an identical convergence condition, ensuring that the reported execution times and speedup measurements correspond to algorithmically equivalent workloads.

2) Execution time results of OpenCL kernel: In this section, we report the findings of our performance study, which compares the execution times of our heterogeneous solution with FPGA acceleration with regular CPU execution. The objective of this study is to show the efficiency of non-blocking memory transactions and asynchronous data transfer in enhancing the overall system performance. Performance evaluation was conducted through two distinct experiments using a randomly generated float dataset:

- Varying the Number of Clusters: The data points were kept fixed at 15,000 while the number of clusters was varied across a range of 32 to 256 (Fig. 3 and Table II).
- Varying the Number of Data Points: The cluster number was maintained at 128, and the number of data points was changed in the range of 4,000 to 30,000 (Fig. 4 and Table III).

To quantify the performance improvement of the FPGA implementation over the CPU, we calculate the **overall speedup** as the ratio of the CPU execution time to the FPGA execution time. Formally, the Speedup is defined as:

$$Speedup = \frac{T_{\text{CPU}}}{T_{\text{FPGA}}} \tag{9}$$

where, T_{CPU} and T_{FPGA} represent the execution times on the CPU and FPGA, respectively. A speedup greater than 1 indicates that the FPGA implementation is faster than the CPU.

In the initial experiment, as presented in Table II and illustrated in Fig. 3, the execution times of the CPU and FPGA implementations were evaluated while maintaining a constant dataset size (15,000 data points) and varying the number of clusters k. The results demonstrated a pronounced divergence in performance between the two architectures.

TABLE II. FPGA VS. CPU K-MEANS KERNEL PERFORMANCE WITH 15000-POINT DATASET.

k	FPGA+CPU time (ms)	CPU time (ms)	Speedup
32	3.177	16.543	5.2x
64	3.206	32.353	10.1x
128	3.223	64.149	20x
256	3.25	127.613	39.26x

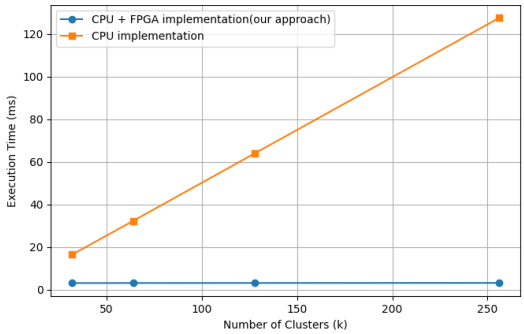

Fig. 3. FPGA vs. CPU K-means kernel performance with 15000-point dataset.

The CPU experienced a marked increase in execution time with increasing cluster counts, which can be attributed to the algorithmic processing complexity that increased cardinally with cluster count, thereby demanding more processing time. The FPGA implementation, by contrast, demonstrated much more consistent and scalable performance. Although a marginal increase in execution time was observed with additional clusters, the FPGA's parallel processing capabilities and optimized memory management mitigated the impact of the rising computational load, preventing a proportional escalation in runtime. Comparatively, the FPGA implementation achieved a speedup of up to 39X compared to the CPU, which is particularly useful for clustering, which is inherently computationally intensive.

In the second experiment, shown in Fig. 4 and summarized in Table III, the number of clusters was held constant (k=128) while the dataset size was scaled from 4,000 to 30,000 data points.

TABLE III. FPGA VS. CPU K-MEANS ALGORITHM PERFORMANCE WITH K=128

N	FPGA+CPU time (ms)	CPU time (ms)	Speedup
4000	2.706	17.08	6.3x
8000	2.792	34.215	12.25x
10000	2.875	42.774	14.87x
12000	3.078	50.323	16.34x
14000	3.203	59.843	18.68x
16000	3.616	68.361	18.9x
20000	3.883	85.522	22.02x
30000	4.258	128.171	30x

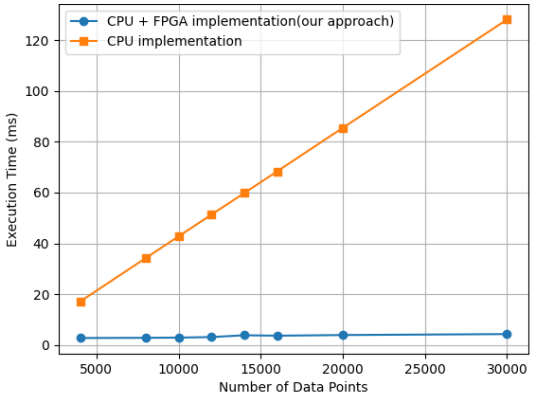

Fig. 4. FPGA vs. CPU K-means algorithm performance with k=128.

Consistent with the first experiment, the CPU exhibited a steep rise in execution time as the dataset expanded. This is largely attributed to the inherent limitations in CPU-based architectures to deal with large amounts of data, where the overhead of sequential processing and data transfer becomes a tremendous bottleneck. On the other hand, the heterogeneous execution model based on FPGA presented better scalability in big data processing. With asynchronous data transfer and non-blocking memory access, the FPGA initiated kernel execution as early as there was partial data, thus facilitating computation in parallel with ongoing data transfer. This was an optimization that significantly lowered the overall processing latency. When dealing with bigger datasets, the performance improvements were more noticeable, and the FPGA outperformed the CPU by up to 30X, which supported the findings from the initial trial. The results demonstrate the architectural advantages of FPGAs for carrying out data-intensive tasks since they can transfer data and conduct computations simultaneously, offloads the inefficiencies of conventional CPU-based processing.

To analyze the performance and reliability of the proposed K-means kernel on FPGA, we performed 10 independent executions for a dataset with a fixed problem size of N=30,000 data points and k=128 clusters. As illustrated in Table IV and Fig. 5, execution time for the CPU ranged from 128.206 ms to 128.596 ms, while FPGA execution times ranged from 4.066 ms to 4.723 ms. The resultant speedup, defined as the CPU execution time divided by FPGA execution time,

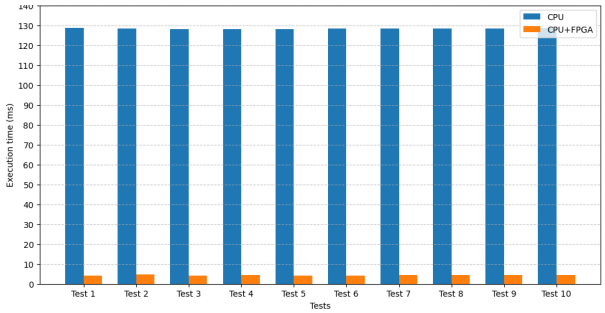

Fig. 5. K-means Kernel execution time (N=30000, k=128) for 10 tests.

ranged from 27.16× to 31.53×, with an average of 29.5×. These results show that the FPGA implementation constantly provided improved speed above the CPU, with stable and reproducible execution times, thereby confirming its suitability for real-time applications.

TABLE IV. CPU AND FPGA EXECUTION TIMES FOR 10 TESTS (N=30,000, K=128).

Test	CPU Time (ms)	CPU + FPGA Time (ms)	Speedup
1	128.849	4.225	30.49x
2	128.277	4.723	27.16x
3	128.206	4.188	30.61x
4	128.207	4.311	29.73x
5	128.211	4.066	31.53x
6	128.390	4.123	31.13x
7	128.275	4.505	28.47x
8	128.293	4.467	28.72x
9	128.324	4.541	28.25x
10	128.596	4.452	28.88x

3) Resource utilization: We also analyzed the FPGA resource consumption of the designed K-means kernel. Table V summarizes the hardware resource utilization of the proposed FPGA accelerator. The design occupies only 7.74% of the available LUTs, 5.17% of the registers, 0.71% of BRAM resources, and 3.18% of the DSP slices available on the XC7Z020 device. These results indicate that the proposed accelerator achieves substantial performance improvements while utilizing only a small fraction of the FPGA resources. The low BRAM utilization demonstrates that the design does not require extensive on-chip storage, while the limited DSP usage confirms that the implementation remains lightweight and scalable. Such moderate resource consumption leaves significant headroom for future optimizations, including the replication of processing elements, support for higher-dimensional datasets, or the integration of multiple clustering kernels within the same device. The reported speedups of up to 39× are therefore achieved without exhausting the available FPGA resources, highlighting a favorable performance-to-resource trade-off. Beyond resource consumption, post-implementation timing analysis confirms that the design reliably meets its performance target. The kernel was constrained to a 100 MHz (10 ns period) operating frequency, and static timing analysis after place-and-route reported all timing constraints as met, with a worst-case (critical path) positive margin of 0.754 ns. This confirms that the reported execution times and speedups correspond to a design that is fully timing-closed at its target operating frequency.

TABLE V. HARDWARE RESOURCE UTILIZATION OF THE K-MEANS KERNEL ON FPGA.

Resource Type	Used	Available	Utilization (%)
Slice LUTs	4118	53200	7.74
– LUTs as Logic	3558	53200	6.69
– LUTs as Memory	560	17400	3.22
Slice Registers	5498	106400	5.17
– Registers as Flip-Flop	5498	106400	5.17
Block RAM (BRAM)	1	140	0.71
DSP Slices (DSP48E1)	7	220	3.18

The finalized FPGA device layout with the K-means kernel implementation is shown in Fig. 6. The embedded ARM processor is visible in the left corner of the chip in the figure. The close integration of the processing system and programmable logic is demonstrated by this architectural layout, which also highlights an effective design approach specifically designed to improve the K-means algorithm's performance.

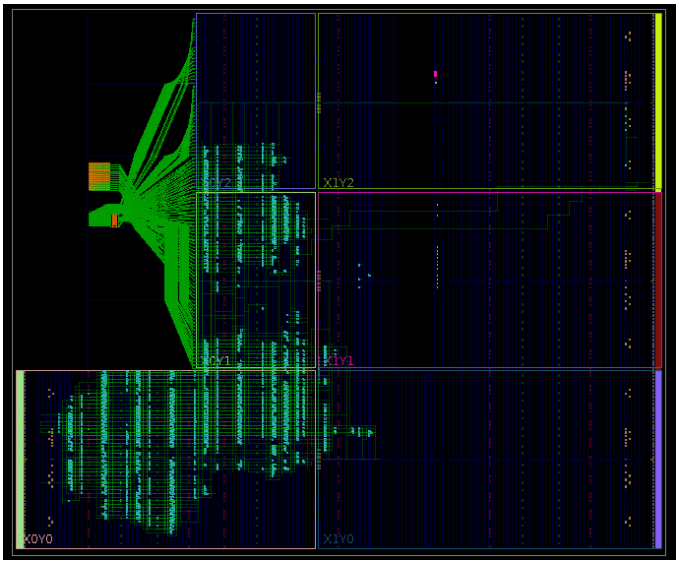

Fig. 6. The FPGA chip floorplan of the K-means kernel implementation.

4) Power consumption analysis: The proposed FPGA-based implementation exhibits a total on-chip power consumption of 1.833 W, including 1.688 W of dynamic power and 0.145 W of static power, as illustrated in Fig. 7.

The Processing System (PS7) accounts for approximately 90% of the dynamic power consumption, while the remaining power is distributed among the programmable logic resources, including clocks, signals, logic, DSP blocks, BRAM, and clock management circuitry. This behavior is consistent with the proposed hardware–software co-design approach, where the ARM Cortex-A9 processor is responsible for host execution, memory management, and OpenCL runtime control, whereas the FPGA fabric accelerates only the computationally intensive distance calculation kernel.
Despite the dominant contribution of the processing system, the overall power consumption remains below 2 W, demonstrating the suitability of the proposed architecture for power-constrained embedded platforms.

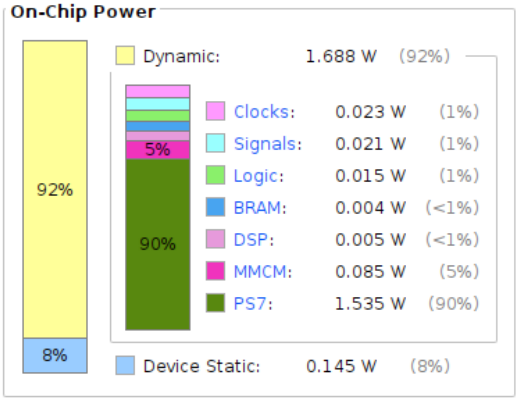

Fig. 7. On-chip power consumption of the proposed hardware-software co-design implementation.

5) Evaluation on a real-world dataset: To evaluate the proposed CPU–FPGA accelerator under realistic operating conditions, experiments were additionally conducted using the SimpleMaps World Cities dataset. This dataset contains real-world geospatial data with naturally non-uniform spatial distributions, providing a more representative benchmark for assessing the clustering performance on practical applications.

The dataset consists of 50,250 populated places worldwide. For the K-means clustering task, only the two numerical spatial attributes, latitude and longitude, were used as input features. The characteristics of the dataset are summarized in Table VI.

TABLE VI. CHARACTERISTICS OF THE SIMPLEMAPS WORLD CITIES DATASET.

Property	**Description**
Dataset Type	Real-world geospatial dataset
Instances	50,250 cities
Dimensions Used	2 (Latitude, Longitude)
Original Attributes	City name, country, administrative region, latitude, longitude, population, capital status, etc.
Domain	Geographic locations worldwide

The same experimental methodology adopted for the synthetic datasets was applied to the real-world dataset. Table VII reports the execution times obtained for the proposed CPU–FPGA implementation and the software-only CPU implementation.

TABLE VII. EXECUTION TIME COMPARISON ON THE SIMPLEMAPS WORLD CITIES DATASET.

N	k	FPGA+CPU Time (ms)	CPU Time (ms)	Speedup
50 250	128	5.4	214.757	$\sim 39.7\times$

The experimental results demonstrate that the proposed accelerator maintains its performance benefits on real-world data, achieving an approximate 39× speedup over the software-only implementation. This performance is consistent with the gains observed on the synthetic datasets, despite the larger size and more irregular spatial distribution of the real-world dataset, demonstrating the scalability and generalization capability of the proposed architecture.

TABLE VIII. COMPARISON OF THE PROPOSED IMPLEMENTATION WITH REPRESENTATIVE FPGA/OPENCL-BASED K-MEANS ACCELERATION APPROACHES.

Reference	Platform	Programming Model	Baseline	Dataset / Precision	Acceleration Scope	Speedup
Tang & Khalid [21]	Altera Stratix V A7	OpenCL	6-core Intel Xeon W3670	Benchmark suite / Integer	Complete K-Means using two OpenCL kernels with local-memory optimization	3×–21×
Canilho *et al.* [22]	FPGA Hybrid Platform	RTL/HLS	ARM CPU	Not specified	Partial acceleration: distance computation and assignment in hardware; centroid update in software	≈10×
Paulino *et al.* [23]	FPGA	OpenCL	High-performance CPU	Integer datasets / Integer	Complete K-Means using multiple OpenCL task and NDRange kernel implementations	Up to 1.54× (+80% energy savings)
Baydoun *et al.* [30]	GPU	CUDA / OpenMP	Sequential CPU	Multiple datasets and feature dimensions	Complete software implementation (GPU and multi-core CPU)	≈2× (OpenMP), ≈6× (CUDA)
Proposed Work	**Xilinx Zynq-7000 XC7Z020**	**OpenCL**	**Embedded ARM Cortex-A9**	**Synthetic 2D /SimpleMaps World Cities dataset/ Floating-point**	**Partial acceleration: nearest-centroid assignment on FPGA; centroid update and convergence control on ARM**	**Up to 39× (29.5× average)**

6) Performance comparison: Table VIII compares the proposed implementation with representative FPGA/OpenCL-based K-Means accelerators. Direct comparison of the reported speedups should be interpreted with caution because the evaluated studies differ in FPGA architecture, processor baseline, numerical precision, dataset characteristics, and implementation scope. For instance, Tang and Khalid [21] and Paulino *et al.* [23] implement complete K-Means accelerators, whereas the proposed architecture adopts a heterogeneous hardware/software partitioning in which only the computationally dominant assignment stage is offloaded to the FPGA, while centroid recomputation, convergence checking, and execution control remain on the embedded ARM processor. This partitioning is a deliberate design choice that exploits the FPGA for massively parallel, compute-intensive operations while retaining the control-oriented and less computationally demanding tasks on the processor, thereby reducing hardware complexity and resource utilization without sacrificing overall performance. Furthermore, several previous FPGA implementations rely on integer arithmetic, whereas the proposed design operates on floating-point data, which generally incurs higher computational and hardware costs. In addition, the reported speedups are obtained using different processor baselines, ranging from embedded ARM processors to high-performance desktop CPUs. Consequently, the reported performance figures should be interpreted within the context of each implementation rather than as direct performance rankings.

7) Limitations and scalability discussion: Although the proposed heterogeneous architecture achieves substantial acceleration for two-dimensional floating-point datasets, several limitations should be acknowledged. First, the current implementation is specialized for two-dimensional observations, and extending the design to higher-dimensional feature vectors requires additional arithmetic operations for each distance evaluation. Consequently, the computational complexity of the assignment stage increases proportionally with the feature dimension, leading to higher DSP, LUT, and routing requirements when parallelism is maintained. Moreover, larger feature vectors increase the amount of data transferred between external memory and the FPGA, making memory bandwidth progressively more critical and potentially limiting the achievable acceleration. Nevertheless, the adopted hardware/software partitioning remains applicable because the assignment stage continues to dominate the computational cost of the K-Means algorithm, while the centroid update and convergence-control operations remain comparatively lightweight and are efficiently handled by the embedded ARM processor. Future work will therefore investigate scalable kernel architectures based on parameterized dimensionality, data tiling, and optimized memory hierarchies to improve performance for high-dimensional clustering problems.

VI. CONCLUSION

This study presented an OpenCL-based FPGA acceleration of the K-means clustering algorithm on a Xilinx Zynq platform. The proposed heterogeneous architecture offloads the computationally intensive distance calculation stage to programmable logic while retaining control-oriented tasks on the embedded processor. This hardware/software partitioning enables efficient exploitation of parallelism while preserving implementation flexibility. Experimental evaluation demonstrated speedups of up to 39× compared with the CPU implementation on synthetic benchmarks, with consistent performance gains confirmed on a real-world geospatial dataset, supporting the generalizability of the proposed approach beyond controlled synthetic conditions. Resource utilization further showed that these performance gains are achieved with modest consumption of LUTs, BRAMs, and DSP slices. The results confirm the suitability of FPGA-based acceleration for computationally intensive clustering workloads. Our study indicates that a hybrid CPU–FPGA architecture is highly effective for accelerating iterative, data-intensive algorithms such as K-means. Future research will aim to expand the proposed methodology to handle more complex datasets, as well as more sophisticated algorithms, to thoroughly evaluate its scalability and performance in demanding scenarios. We also aim to explore the integration of multi-FPGA systems and distributed processing architectures to better exploit parallelism and accelerate deep learning algorithms. Furthermore, we will investigate advanced methods to optimize off-chip memory access to reduce data transfer latency and improve overall efficiency. These guidelines are meant to augment and expand

current design practice. They help in creating energy-efficient and high-speed hardware accelerators for a range of machine learning and deep learning applications.

Declaration on Generative AI

Generative AI tools were used exclusively to improve the grammar, language, and readability of this manuscript. The authors reviewed and edited all AI-assisted text and take full responsibility for the content of the manuscript.

References

[1] A. E. Ezugwu, A. M. Ikotun, O. O. Oyelade, L. Abualigah, J. O. Agushaka, C. I. Eke, and A. A. Akinyelu, *A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects*, Engineering Applications of Artificial Intelligence, vol. 110, pp. 104743, 2022.

[2] G. J. Oyewole and G. A. Thopil, *Data clustering: application and trends*, Artificial Intelligence Review, vol. 56, no. 7, pp. 6439–6475, 2023.

[3] Z. Yu, *Big data clustering analysis algorithm for internet of things based on K-means*, International Journal of Distributed Systems and Technologies (IJDST), vol. 10, no. 1, pp. 1–12, 2019.

[4] C. Jiang, J. Wan, and H. Abbas, *An edge computing node deployment method based on improved k-means clustering algorithm for smart manufacturing*, IEEE Systems Journal, vol. 15, no. 2, pp. 2230–2240, 2020.

[5] Y. Yang, Q. Chen, T. Yi Huang, and P. K. Pareek, *Application research of K-means algorithm based on big data background*, in 2023 IEEE International Conference on Integrated Circuits and Communication Systems (ICICACS), pp. 1–5, 2023.

[6] P. Mackey and R. R. Lewis, *Parallel k-means++ for multiple shared-memory architectures*, in 2016 45th International Conference on Parallel Processing (ICPP), pp. 93–102, 2016.

[7] D. G. Bailey, *Image processing using FPGAs*, Journal of Imaging, vol. 5, no. 5, pp. 53, 2019.

[8] E. Jaballi, S. Gdaim, and N. Liouane, *Design and implementation of FPGA-based hardware acceleration for Machine Learning using OpenCL: A case study on the K-Means algorithm*, in 2024 International Conference on Control, Automation and Diagnosis (ICCAD), pp. 1–6, 2024.

[9] S. Mittal, *A survey of FPGA-based accelerators for convolutional neural networks*, Neural Computing and Applications, vol. 32, no. 4, pp. 1109–1139, 2020.

[10] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, *[DL] A survey of FPGA-based neural network inference accelerators*, ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 12, no. 1, pp. 1–26, 2019.

[11] A. Shawahna, S. M. Sait, and A. El-Maleh, *FPGA-based accelerators of deep learning networks for learning and classification: A review*, IEEE Access, vol. 7, pp. 7823–7859, 2018.

[12] T. Wang, C. Wang, X. Zhou, and H. Chen, *A survey of FPGA based deep learning accelerators: Challenges and opportunities*, arXiv preprint arXiv:1901.04988, 2018.

[13] A. Nechi, L. Groth, S. Mulhem, F. Merchant, R. Buchty, and M. Berekovic, *Fpga-based deep learning inference accelerators: Where are we standing?*, ACM Transactions on Reconfigurable Technology and Systems, vol. 16, no. 4, pp. 1–32, 2023.

[14] A. Itagi, S. Krishvadana, K. P. Bharath, and M. R. Kumar, *FPGA architecture to enhance hardware acceleration for machine learning applications*, in 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), pp. 1716–1722, 2021.

[15] E. Rapuano, G. Meoni, T. Pacini, G. Dinelli, G. Furano, G. Giuffrida, and L. Fanucci, *An fpga-based hardware accelerator for cnns inference on board satellites: benchmarking with myriad 2-based solution for the cloudscout case study*, Remote Sensing, vol. 13, no. 8, pp. 1518, 2021.

[16] S. Ramadurgam and D. G. Perera, *An efficient fpga-based hardware accelerator for convex optimization-based svm classifier for machine learning on embedded platforms*, Electronics, vol. 10, no. 11, pp. 1323, 2021.

[17] S. Tatsumi, M. Hariyama, K. Ito, and T. Aoki, *An FPGA accelerator for PatchMatch multi-view stereo using OpenCL*, Journal of Real-Time Image Processing, vol. 17, pp. 215–227, 2020.

[18] J. E. Stone, D. Gohara, and G. Shi, *OpenCL: A parallel programming standard for heterogeneous computing systems*, Computing in Science & Engineering, vol. 12, no. 3, pp. 66, 2010.

[19] S. Gdaim and A. Mtibaa, *Accelerating convolutional neural networks on FPGA platforms: a high-performance design methodology using OpenCL*, Journal of Real-Time Image Processing, vol. 22, no. 2, pp. 67, 2025.

[20] Y. Koo, C. You, and S. Kim, *OpenCL-Darknet: an OpenCL implementation for object detection*, in 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 631–634, 2018.

[21] Q. Y. Tang and M. A. S. Khalid, *Acceleration of k-means algorithm using altera sdk for opencl*, ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 10, no. 1, pp. 1–19, 2016.

[22] 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), pp. 1–4, 2016.

[23] N. Paulino, J. C. Ferreira, and J. M. P. Cardoso, *Optimizing opencl code for performance on fpga: k-means case study with integer data sets*, IEEE Access, vol. 8, pp. 152286–152304, 2020.

[24] X. Zheng, Q. Lei, R. Yao, Y. Gong, and Q. Yin, *Image segmentation based on adaptive K-means algorithm*, EURASIP Journal on Image and Video Processing, vol. 2018, no. 1, pp. 1–10, 2018.

[25] M. Tleis, R. Callieris, and R. Roma, *Segmenting the organic food market in Lebanon: an application of k-means cluster analysis*, British Food Journal, vol. 119, no. 7, pp. 1423–1441, 2017.

[26] H. Hu, J. Liu, X. Zhang, and M. Fang, *An effective and adaptable K-means algorithm for big data cluster analysis*, Pattern Recognition, vol. 139, pp. 109404, 2023.

[27] H. Yadav, P. Bansal, and R. K. Sunkaria, *Color dependent K-means clustering for color image segmentation of colored medical images*, in 2015 1st International Conference on Next Generation Computing Technologies (NGCT), pp. 858–862, 2015.

[28] M. Er Kara and S. Ü. Oktay Fırat, *Supplier risk assessment based on best-worst method and K-means clustering: a case study*, Sustainability, vol. 10, no. 4, pp. 1066, 2018.

[29] H. Xu, C. Ma, J. Lian, K. Xu, and E. Chaima, *Urban flooding risk assessment based on an integrated k-means cluster algorithm and improved entropy weight method in the region of Haikou, China*, Journal of Hydrology, vol. 563, pp. 975–986, 2018.

[30] M. Baydoun, H. Ghaziri, and M. Al-Husseini, *CPU and GPU parallelized kernel K-means*, The Journal of Supercomputing, vol. 74, no. 8, pp. 3975–3998, 2018.