DUAL: Acceleration of Clustering Algorithms using Digital-based Processing In-Memory

Mohsen Imani[‡], Saikishan Pampana[*], Saransh Gupta[*], Minxuan Zhou[*], Yeseong Kim[†], Tajana Rosing[*]
[‡]Department of Computer Science, UC Irvine
[†]Department of Information and Communication Engineering, DGIST
[*]Department of Computer Science and Engineering, UC San Diego
m.imani@uci.edu; yeseongkim@dgist.ac.kr; {spampana, sgupta, miz087, tajana}@ucsd.edu

***Abstract*—Today's applications generate a large amount of data that need to be processed by learning algorithms. In practice, the majority of the data are not associated with any labels. Unsupervised learning, i.e., clustering methods, are the most commonly used algorithms for data analysis. However, running clustering algorithms on traditional cores results in high energy consumption and slow processing speed due to a large amount of data movement between memory and processing units. In this paper, we propose** DUAL**, a Digital-based Unsupervised learning AcceLeration, which supports a wide range of popular algorithms on conventional crossbar memory. Instead of working with the original data,** DUAL **maps all data points into high-dimensional space, replacing complex clustering operations with memory-friendly operations. We accordingly design a PIM-based architecture that supports all essential operations in a highly parallel and scalable way.** DUAL **supports a wide range of essential operations and enables in-place computations, allowing data points to remain in memory. We have evaluated** DUAL **on several popular clustering algorithms for a wide range of large-scale datasets. Our evaluation shows that** DUAL **provides a comparable quality to existing clustering algorithms while using a binary representation and a simplified distance metric.** DUAL **also provides 58.8× speedup and 251.2× energy efficiency improvement as compared to the state-of-the-art solution running on GPU.**

***Index Terms*—Processing in-memory, Unsupervised learning, Hyperdimensional computing, Algorithm-hardware co-design**

I. Introduction

With the emergence of the Internet of Things (IoT), sensory and embedded devices generate massive data streams and demand services that pose huge technical challenges due to limited device resources. Today IoT applications analyze raw data by running machine learning algorithms. Since the majority of data generated are not associated with any labels, clustering algorithms are the most popular learning methods used for data analysis [1]. Clustering algorithms are unsupervised and have applications in many fields including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics [2]–[5]. These algorithms are used to group a set of objects into different classes, so that objects within the same class are similar to each other. The process of clustering datasets involves heavy computations as most algorithms need to calculate pairwise distances between all the points in the dataset [6]–[8].

Running clustering algorithms with large datasets on conventional processors results in high energy consumption and slow processing speed. Although new processor technology has evolved to serve computationally complex tasks more efficiently, data movement costs between the processor and memory still hinder the higher efficiency of application performance. Processing in-memory (PIM) is a promising solution to accelerate applications with a large amount of parallelism [9]–[16]. Several recent works have explored the advantage of PIM-based architectures to accelerate supervised learning algorithms such as Deep Neural Networks (DNNs) [17], [18]. These approaches mostly use PIM architecture as a dot product engine to perform the vector-matrix multiplication involved in the DNN computation.

There are three main challenges in using existing PIM architectures to accelerate clustering algorithms: (i) the main operations involved in clustering algorithms are pairwise distance computation, e.g., Euclidean distance, and similarity search which cannot be supported entirely by existing PIM architectures [16]. (ii) Most existing PIM architectures are analog-based [15], [17], [18]; thus they use Digital-to-Analog Converter (DAC) blocks to transfer data to the analog domain for the computation and Analog-to-Digital Converter (ADC) to transfer it back to the digital domain. In the existing PIM architectures, the DAC/ADC blocks are dominating the total chip power/area, e.g., 98% of DNN accelerators [18], resulting in very low throughput/area. (iii) They require separate storage and computing memory units, resulting in a large amount of internal data movements. This not only reduces the computation efficiency but also affects the design scalability.

In this work, we present a digital-based PIM architecture, called DUAL, which accelerates a wide range of popular clustering algorithms on conventional crossbar memory. DUAL supports all essential clustering operations in memory, in a parallel and scalable way. DUAL eliminates the necessity of using any ADC/DAC blocks and addresses the internal data movement. The main contributions are listed as follows:

- To the best of our knowledge, DUAL **is the first digital-based processing in-memory architecture that accelerates unsupervised learning tasks.** In contrast to the existing PIM designs, DUAL enables all PIM computation on digital data stored in memory. This eliminates the necessity of using ADC/DAC blocks, providing high throughput/area. DUAL is also the first PIM architecture that support digital search-based Hamming distance computing.
- Instead of working on the original data, DUAL **maps all data points into high-dimensional space enabling the main clustering operations to process in a hardware-**

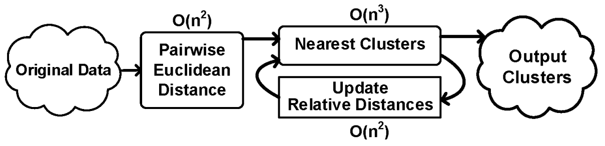


Fig. 1. Hierarchical Clustering Overview.

friendly way. DUAL proposes a novel non-linear encoder that preserves the similarity of the neighbor values in high dimensional space. This encoding simplifies the distance similarity metric from Euclidean to Hamming distance.

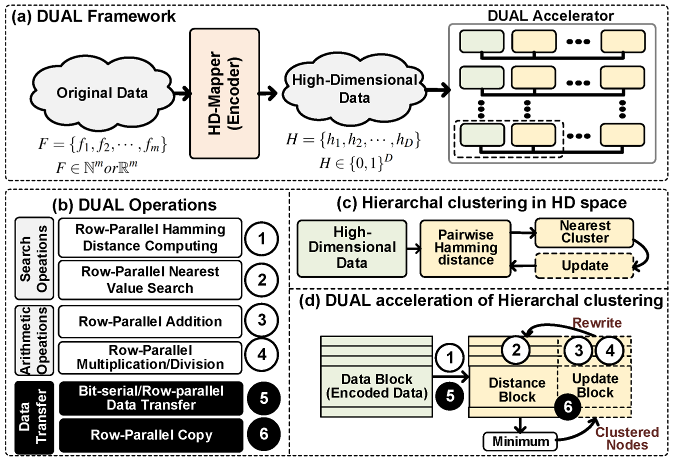
- **We design a PIM architecture that accelerates various clustering algorithms on conventional crossbar memory.** DUAL performs in-place computation in a highly parallel and scalable way, where the data points can be processed without transferring between the storage and computing blocks. Therefore, it eliminates internal data movements between memory blocks. The proposed solution supports a wide range of essential clustering operations, e.g., in-memory distance computations and the nearest search, which can be programmed in high-level languages.

We have evaluated DUAL efficiency on several popular clustering algorithms and a wide range of large-scale datasets. Our evaluations show that DUAL provides a comparable quality of clustering to the baseline clustering algorithms. In terms of efficiency, DUAL provides 58.8× speedup and 251.2× energy efficiency improvement as compared to the state-of-the-art solution running on NVIDIA GTX 1080 GPU. Enabling 1% and 2% lower quality of clustering, DUAL speeds up the computation to 72.5× and 87.4× respectively.

II. Background

Clustering algorithms categorize data points based on the similarity between them in the space. These algorithms use different metrics to compute pairwise distance. Euclidean distance is a commonly used metric in most of the clustering algorithms such as K-means and hierarchical clustering [19]–[21]. Here we provide a detailed explanation of hierarchical clustering, one of the most popular and complex clustering algorithms [22]–[26] as an example. In Section VI-C, we explain how DUAL accelerates other popular clustering algorithms.

Hierarchical clustering is a class of clustering algorithms that start out with each data point being their own cluster [27] and then iterates over the data until only one cluster remains. In each iteration, two clusters are combined, hence reducing the number of clusters by one and so we have to iterate n (size of data set) times to finish clustering. This method of clustering has a time complexity of $O(n^3)$ with space complexity of $O(n^2)$ and hence is both compute and memory-bound as the size of the data set used in clustering increases. Figure 1 presents a high-level overview of hierarchical clustering. The first part of the algorithm uses a distance metric to find pairwise distances between all points in the dataset. State-of-the-art implementations of hierarchical clustering use Euclidean distance for pairwise distance computation [24], [28], [29]. After creating the pairwise distance matrix, we iterate through all values in the matrix and find the pair of data points with the minimum distance. Then, it merges these selected data points into a single cluster. Finally, the algorithm updates the


(a) DUAL Framework
DUAL Accelerator
Original Data
HD-Mapper (Encoder)
High-Dimensional Data
$F = \{f_1, f_2, \cdots, f_m\}$
$F \in \mathbb{N}^m or \mathbb{R}^m$
$H = \{h_1, h_2, \cdots, h_D\}$
$H \in \{0,1\}^D$
(b) DUAL Operations
Search Operations
Row-Parallel Hamming Distance Computing
Row-Parallel Nearest Value Search
Arithmetic Opeations
Row-Parallel Addition
Row-Parallel Multiplication/Division
Data Transfer
Bit-serial/Row-parallel Data Transfer
Row-Parallel Copy
(c) Hierarchal clustering in HD space
High-Dimensional Data
Pairwise Hamming distance
Nearest Cluster
Update
(d) DUAL acceleration of Hierarchal clustering
Rewrite
Data Block (Encoded Data)
Distance Block
Update Block
Minimum
Clustered Nodes
Fig. 2. Overview of DUAL platform accelerating clustering algorithms.

distance of the merged data points with respect to all other clusters based on an updated policy.

There are several different linkages to update the distance matrix: single-linkage, complete-linkage, average-linkage, and ward-linkage [30]. For disjoint clusters a_i, a_j, and a_k with sizes s_i, s_j, and s_k, the linkages are defined as follows:

$$Single-linkage: d(a_i \cup a_j, a_k) = Min[d(a_i, a_k), d(a_j, a_k)]$$

$$Complete-linkage: d(a_i \cup a_j, a_k) = Max[d(a_i, a_k), d(a_j, a_k)]$$

$$Average-linkage: d(a_i \cup a_j, a_k) = \frac{s_i \times d(a_i, a_k) + s_j \times d(a_j, a_k)}{s_i + s_j}$$

The state-of-the-art algorithms are using *Ward*-linkage to update the pairwise distance matrix [31].

$$d(a_i \cup a_j, a_k) = C_1 \times d(a_i, a_k) + C_2 \times d(a_j, a_k) - C_3 \times d(a_i, a_j)$$
$$C_1 = \frac{s_i + s_k}{s_i + s_j + s_k};\ C_2 = \frac{s_j + s_k}{s_i + s_j + s_k};\ C_3 = \frac{s_k}{s_i + s_j + s_k}$$

We call C_1, C_2, and C_3 as Ward's coefficients. The clustering algorithm continues by iteratively finding the next two closet clusters in the distance matrix and merging them into a cluster.

III. DUAL Overview

In this paper, we propose DUAL, a novel platform to accelerate unsupervised learning in a fully digital PIM architecture. Figure 2a shows an overview of DUAL framework consisting of an HD-Mapper and a digital-based PIM accelerator. Instead of working on original data, our architecture maps all data points to long-size binary vectors. This data mapping replaces complex clustering operations with hardware friendly operations. DUAL also exploits the resistive characteristics of Non-Volatile Memory (NVM), in particular memristor devices [32], [33], to support all necessary clustering operations in memory.

A. HD-Mapper

The goal of the HD-Mapper is to encode data points into high-dimensional vectors, called hypervector, such that the data can keep their similarity using a PIM-friendly Hamming distance metric. The HD-Mapper can be a Locality Sensitive Hashing (LSH) or any other encoding function [34]–[37]. There are several approaches based on Hyperdimensional (HD) computing to perform the encoding functionality. However,

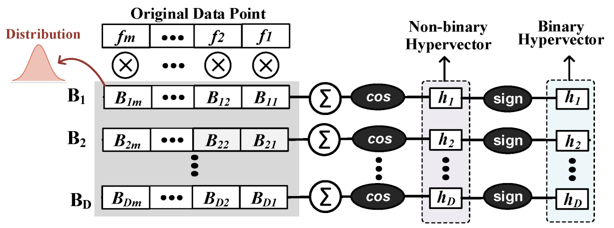


Fig. 3. HD-mapper: encoding data points into high-dimensional space.

all existing approaches linearly map each input feature into the hyperspace [38]–[40]. In contrast, we propose HD-mapper that explicitly considers non-linear interactions between input features. The proposed encoding is inspired by the Radial Basis Function (RBF) kernel trick method [41], [42]. The underlying idea of HD-mapper is that data that is not linearly separable in original dimensions, might be linearly separable in higher dimensions. Consider certain functions $K(x,y)$ which are equivalent to the dot product in a different space, such that $K(x,y) = \Phi(x)\cdot\Phi(y)$, where $\Phi(\cdot)$ is often a function for high dimensional projection. The RBF or Gaussian Kernel is the most popular kernel:

$$K(x,y) = e^{\frac{-||x-y||^2}{2\sigma^2}}$$

We can take advantage of this implicit mapping by replacing their decision function with a weighted sum of kernels:

$$f(\cdot) = \sum_{i=0}^{N} c_i K(\cdot, x_i)$$

where (x_i, y_i) is the training data sample, and the c_is are constant weights. The study in [41] showed that the inner product can efficiently approximate Radial Basis Function kernel:

$$K(x,y) = \Phi(x)\cdot\Phi(y) \approx z(x)\cdot z(y)$$

The Gaussian kernel function can now be approximated by the dot product of two vectors, $z(x)$ and $z(y)$.

Figure 3 shows our encoding procedure. Let us consider an encoding function that maps a feature vector $\mathbf{F} = \{f_1,\ f_2,\ldots,\ f_m\}$, with m features ($f_i \in \mathbb{R}$) to a hypervector $\mathbf{H} = \{h_1,\ h_2,\ldots,\ h_D\}$ with D dimensions ($h_i \in \{0,1\}$). We generate each dimension of encoded data by calculating a dot product of feature vector with a randomly generated vector as $h_i = cos(\mathbf{B}_i \cdot \mathbf{F})$, where B_i is a randomly generated vector from a Gaussian distribution (mean $\mu = 0$ and standard deviation $\sigma = 1$) with the same dimensionality of the feature vector. The random vectors $\{\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_D\}$ can be generated once offline and then can be used for the rest of the classification task ($\mathbf{B}_i \in \mathbb{R}^m$). After this step, each element, h_i of a hypervector $\mathbf{H}$, has a non-binary value. We prefer binary hypervectors for computation efficiency. We thus obtain the final encoded hypervector by binarizing it with a sign function ($\mathbf{H}' = sign(\mathbf{H})$) where the sign function assigns all positive hypervector dimensions to '1' and zero/negative dimensions to '0'. The encoded hypervector stores the information of each original data point with D bits. HD-Mapper gives an analytical model to estimate the required dimensionality of the encoder depending on the number of data points and the number of clusters. The discussion about this estimation is out of the scope of our paper. More information can be found by looking at the amount of orthogonal information that each hypervector can store in the HD space [43].

B. DUAL Accelerator

The second module is a digital-based PIM architecture that enables parallel encoding and clustering computation over the encoded hypervectors stored in memory. Unlike prior PIM designs that use large ADC/DAC blocks for analog computing [15], [17], [18], DUAL performs all clustering computations on the digital data stored in memory. This eliminates ADC/DAC blocks, resulting in high throughput/area and scalability. DUAL uses two blocks for performing the computation; a *data block* and a *distance block*. The data block stores the encoded data points and computes pairwise similarity using a row-parallel Hamming distance computation. Each distance/data block supports the following set of operations (shown in Figure 2b): (i) *search-based operations*: row parallel Hamming distance computation and nearest search. (ii) *Arithmetic operations*: row-parallel addition, multiplication and division.

Figure 2c,d shows how DUAL maps hierarchical clustering into PIM acceleration. In each iteration, DUAL computes the Hamming distance of each data point with all stored data points in all data blocks using the row-parallel search operation and the result is written in a *distance memory*. After computing all pairwise distances, DUAL performs the search for the nearest value over the distance matrix. Our design supports the nearest search operation in a row-parallel way. Next, DUAL clusters the two data points with the highest similarity and then updates the relative distance of all other data points with the clustered nodes. The distance update is computed using linear arithmetic operations, e.g., addition, multiplication, which can be performed in a row-parallel way in the *update block* (Figure 2c,d). The updated distance vectors will be written back into the corresponding row/column of the *distance block*. DUAL continues computation by iteratively finding and clustering data points with the closest distance. DUAL exploits the supported PIM operations to perform clustering tasks where data is already stored in memory. DUAL also uses interconnects to enable bit-serial/row-parallel data transfer between the data and distance blocks. This eliminates the overhead of internal data movement between the data and distance blocks (Figure 2c,d).

IV. DUAL Supported Operations

A. Search/Similarity Computation

The exact search is one of the native operations supported by crossbar memory. During the search, the crossbar memory gets the configuration of a Content Addressable Memory (CAM), where each CAM cell is represented using two memristor devices (0T-2R) [44], [45]. These devices store complementary values. During the search, a row-driver of the CAM block pre-charges all CAM rows (match-lines:*ML*s) to supply voltage (V_{dd}). The search operation starts by loading the input query into the vertical bit-lines (BLs) connected to all CAM rows. Similar to CAM cells, each input query is represented using two complementary bits. Consider a CAM cell (shown in Figure 4a), if a query input matches with the stored value in the CAM cell, the *ML* will stay charged. However, in case of a mismatch between the CAM cell and the query data, the CAM starts discharging the *ML*. Conventionally, CAM blocks exploit the *ML* discharging current to enable the exact search operation.

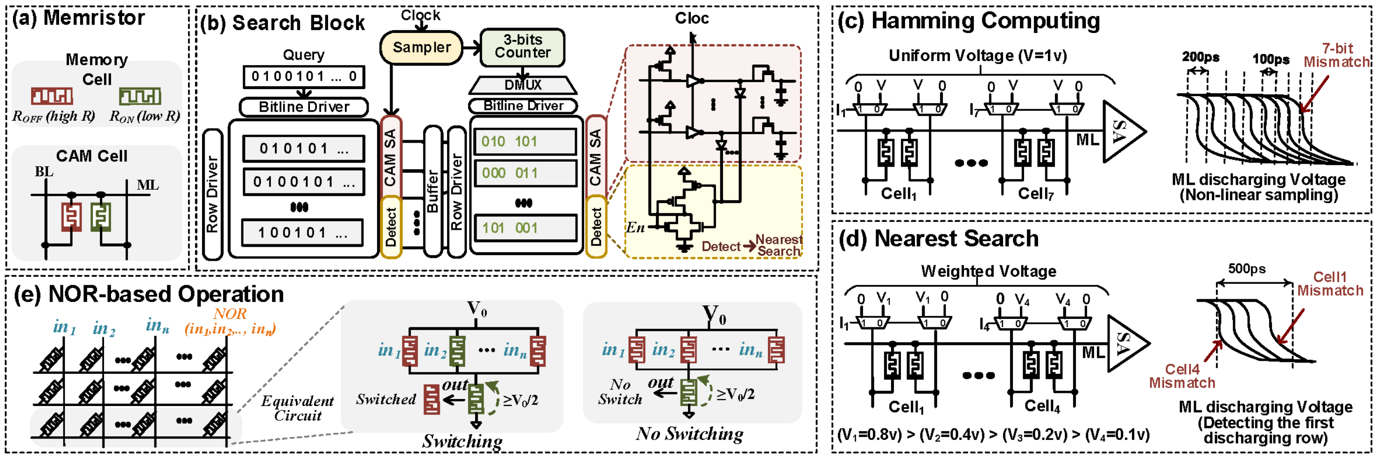


Fig. 4. DUAL operations: search-based and arithmetic

Here, we modify the CAM block to enable Hamming distance computation and nearest search in a parallelized way. Note that although there are several work used NVM-based CAM to support the nearest absolute search functionalities [39], [46], DUAL is the first digital-based PIM architecture that supports Hamming computing without using costly ADC blocks.

1) Hamming Distance Computation: Figure 4b shows the architecture of the modified CAM block. Our design exploits the timing characteristic of *ML* discharging current in order to detect the Hamming distance of each CAM row with an input query. The search sense amplifier, "CAM SA" shown in Figure 4b, samples the *ML* current in different time stamps and finds the number of mismatches depending on the cycle that the *ML* voltage is dropped. Note that during Hamming computing, the detector circuit, shown in Figure 4b, is deactivated ($En = 0$). More mismatches between a query and CAM row results in a faster-discharging current of *ML*. *ML* discharging current does not change linearly with the number of mismatches when the search is performed on a long row with too many cells. As Figure 4c shows, when we search on a row with a 4-bit length, we can detect the difference between 2-bits, 3-bits, and 4-bits mismatching by sampling *ML* in linear time (200ps). However, considering a 7-bit CAM row, the mismatches of 6-bits and 7-bits are happening much faster than 2-bits to 3-bits. This limits the maximum possible bit-search parallelism to 4-bits using linear search. In contrast, we propose a non-linear sampling time, shown in Figure 4c, which enables DUAL to search up to 7-bits in parallel.

To enable fast Hamming distance computation, the distance result should be written in a row-parallel way on the distance memory block. This requires translating sampling time to a Hamming distance and writing it into the distance block. However, this approach requires an extra processing step. Here, we present an approach that enables the result of the 7-bit Hamming distance search to be written in the distance memory in a single cycle. Our approach assigns a single 3-bit counter to each memory block, where the counter value increments with the same clock used for the search sampling. Depending on the discharge CAM rows in each sampling time, DUAL activates corresponding rows of the distance memory. Finally, DUAL writes the counter value on all selected rows in the distance block. Write operation happens in a row-parallel way, where all activated rows will get counter values in a single write cycle.

The write latency in the non-volatile memory is slower than the search operation frequency (1ns write latency vs. 200ps/100ps search sample). Therefore, to provide high throughput, we use a 7-bit register next to each data block in order to store the sampling time that each row has been discharged. After the Hamming distance computation, DUAL sequentially activates the rows of distance block (depending on the values stored in each column of the buffer), and accordingly writes the counter values to them. DUAL performs the Hamming distance computation serially on 7-bits windows. The result of distance computation will be written as $D/7$ values of 3-bits on the distance memory (shown in Figure 4b).

2) Nearest Value Search: In the Hamming distance operation, DUAL computes the actual distance value of query, rather than finding a row with the nearest Hamming distance. However, the second popular clustering operation is to find a row that has the nearest distance to query data. In this search, the memory stores integer/fixed-point values, thus bits in different positions have different weights. To consider the impact of each bit indices, we weight different bitlines by connecting them to different bitline voltages. During the search, the most significant bits (MSBs) are assigned to a higher voltage than bits in lower positions (shown in Figure 4d, $V_1 = 0.8V$, $V_2 = 0.4V$, $V_3 = 0.2V$, $V_4 = 0.1V$). By adjusting the bitline voltages, we enable a 4-bit parallel search operation. In a nominal voltage/process technology, we can increase the number of bits up to 8-bits. However, considering variations in voltage and process technology, 4-bit parallel search provides enough noise margin, ensuring exact nearest search computation over 5000 Monte-carlo simulations (considering 10% variations in technology and memristor values).

Assume a CAM block storing m bit integer numbers. The search for the nearest value starts from four MSBs. The first row that discharges the *ML* is the row which has the highest similarity with query data. Each discharging match-line flows a current into a detector circuit (shown in Figure 4b). In this mode, the detector circuit is activated ($En = 1$), and it has two responsibilities: (i) it stops the search operation by precharging all match-lines to V_{dd} voltage, and (ii) it transfers the discharged rows to the output stage. The activated rows at the end of the search cycle (500ps) have the nearest distance to the query data. Depending on the row with the highest matches, DUAL activated those rows of the memory and continues the search operation on the next 4-bits. This sequential search increases the weight of the bits in the MSB as compared to the bit located in a lower stage. The search continues over all the bits, ending up with a single activated row in the last stage.

B. Row-Parallel PIM-based Arithmetic

DUAL supports arithmetic operations directly on digital data stored in memory without reading them out of sense amplifiers [16], [47]–[52]. Our design exploits the memristor switching characteristic to implement NOR gates in digital memory [16], [48]. DUAL selects two or more columns of the memory as input NOR operands by connecting them to ground voltage (Shown in Figure 4e). Next, DUAL connects the bitline corresponding to the output of NOR operation to a write voltage (V_0). In addition, all output memristors located in the output column are initialized to R_{ON} in the beginning. To execute NOR in a row, an execution voltage, V_0, is applied at the p terminals of the inputs while the p terminal of the output memristor is grounded, as shown in Figure 4e. During NOR computation, the output memristor is switched from R_{ON} to R_{OFF} when one or more inputs stored '1.' value (R_{ON}). In fact, the low resistance input passes a current through an output memristor resulting in writing R_{off} value on it. This NOR computation performs in row-parallel on all the activated memory rows by the row-driver.

Since NOR is a universal logic gate, it can be used to implement other logic operations like addition [53] and multiplication [54]. Our approach also supports division by approximately modeling it with the multiplication of numerator and the inverse of denominator [55]. The inversion is computed by filliping all denominator bits, adding it with 1, and left shifting of the result [55]. For example, 1-bit addition (inputs being A, B, C) can be represented in the form of NOR as,

$$C_{out} = ((A+B)' + (B+C)' + (C+A)')'. \tag{1a}$$

$$S = (((A' + B' + C')' + ((A+B+C)' + C_{out})')')'. \tag{1b}$$

Here, C_{out} and S are the generated carry and sum bits of addition. Also, $(A+B+C)'$, $(A+B)'$, and A' represent $NOR(A,B,C)$, $NOR(A,B)$, and $NOR(A,A)$ respectively. We need to reserve extra memory columns for DUAL arithmetic operations to store intermediate results (discussed in Section V and Table III). Note that DUAL can also support floating point arithmetic operations using the same approach shown in [16]. Note that arithmetic operations in DUAL are in general slower than the corresponding CMOS-based implementations. This is because memristor devices are slow in switching. However, this PIM architecture can provide significant speedup with massive parallelism. For example, DUAL takes the same amount of time for addition in a single row or all memory rows. However, the processing time in conventional cores highly depends on the data size.

V. DUAL Implementation

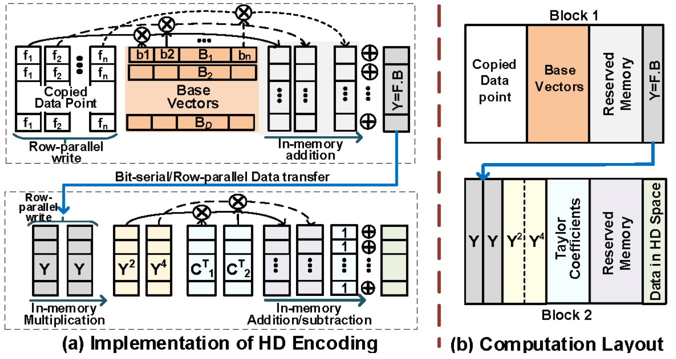


Fig. 5. (a) DUAL encoding in-memory implementation, (b) computation layout.

A. Encoding Implementation

The computation of the HD-Mapper is vector-matrix multiplication between the feature vector and the base vectors. This is followed by applying the cosine function on the dot-product result (as explained in Section III-A). DUAL accelerates the encoding module by performing a row-parallel in-memory multiplication and addition. Figure 5a shows the structure of DUAL accelerating encoding module on two crossbar memories: the first block computes the dot product of the input data with the base vectors, and the second block applies cosine functionality on the dot product result. These two blocks are working in a pipeline, meaning that when the first block computes the dot product of the i^{th} data point, the second block computes the cosine similarity of the dot product result of the $i-1^{th}$ data point. Figure 5b shows the layout of vectors stored in memory, implementing encoding computation.

Dot product Implementation: DUAL supports row-parallel arithmetic operations in memory. To execute the encoding using those operations, we perform multiplication of the input vector with the transposed base vectors stored in the same memory block. Figure 5 shows the overview of DUAL accelerating encoding module. Our design performs a row-parallel write operation to store multiple copies of a data point in different memory rows ($\{f_1, f_2, \cdots, f_m\}$), where features can be an integer or fixed-point number with any bitwidth. The row-parallel write can be performed by activating all memory rows during the write operation. The same memory block stores all base vectors ($\{B_1, B_2, \cdots, B_D\}$) horizontally in different rows. For an application, base vectors are fixed; thus they need to be written in memory block only once. The encoding computation starts by multiplying each feature column with a corresponding column of base vectors. We use the reserved memory, shown in Figure 5b, to perform intermediate arithmetic operations and store the multiplication results. After covering the multiplication of all m features, DUAL performs a row-parallel in-memory addition to accumulate all multiplication results ($Y = F \cdot B$).

Cosine Implementation: Next, we apply the cosine function on the dot product (Y). DUAL approximates cosine function by using the first three terms of Taylor expansion. DUAL sends two copies of the dot product vector to the next memory block (Block 2 shown in Figure 5b) in a row-parallel/bit-serial way. Then, it exploits in-memory multiplication to compute different powers of the product vector (Y^2 and Y^4 shown in Figure 5). DUAL multiplies the result vectors with the

Taylor expansion coefficients, which are already pre-stored in the second memory block. Finally, our model computes the result of Taylor explanation by performing row-parallel addition/subtraction between different memory columns. We consider the inverse of output vector sign-bit as encoded data with binary representation. Note that DUAL uses the reserved memory to compute the intermediate results of Taylor's expansion.

DUAL is a scalable architecture. If the number of base vectors exceeds the number of memory rows ($D > 1k$), we store the rest of the base vectors in another memory block.If the number of features exceeds the size of a block bitline, we compute the dot product results in two neighbor blocks and aggregate the dot product results before passing it through cosine function.

B. Pairwise Distance Computation

Encoding is a single-pass process. After encoding, DUAL starts the clustering task on the encoded data points stored in memory. DUAL exploits two types of memory blocks for clustering: (i) data blocks that store the encoded data points and are responsible for pairwise distance computation, (ii) distance blocks that store the pairwise distance matrix and perform the clustering task. DUAL assigns a column of the distance memory to store a flag bit and another column to store the size of the cluster ($\{s_1, s_2, \cdots, s_n\}$). These sizes are initially set to 1, as each data point is a separate cluster. Figure 6 shows an overview of mapping Hierarchical clustering to DUAL hardware. Figure 7 also shows the layout of DUAL operations performing clustering in memory. Each data block supports a row-parallel Hamming distance computation (explained in Section IV-A1). DUAL first computes the distance of the first encoded data point (a_1) with all data stored in the data block (Figure 6**A**). The Hamming distance computation is performed serially over 7-bits windows. The distance results will be written as a 3-bit value in the distance memory. Assuming a data point with D dimensions, the distance memory stores $D/7$ 3-bit values. DUAL exploits in-memory addition (explained in Section IV-B) to accumulate all the partial distance values in a row-parallel way. The results of accumulation are stored in the same memory block, only using $log\ D$ bits. Note that when we compute the Hamming distance of a_i to all data points, we write the maximum value on the diagonal distance well ($d(a_i, a_i) = D$). This write happens after the accumulation of all 3-bits counters. We continue a similar search operation for all data points in order to find all pairwise distances.

C. Nearest Cluster

The next step is to compare all distance values stored in the distance memory and find data points with the highest similarity. We exploit nearest search functionality (explained in Section IV-A2) to implement row-parallel minimum search. Our design starts the search operation in the first column of the distance memory with valid flag bit set by searching for a query which is actually the lowest value, i.e., 000...0. Any row which has the highest similarity to query data is the smallest value in the column (Figure 6**B**). Next, we perform the same search operation on other columns of the distance memory with a valid flag bit set. After each search, DUAL writes a selected row along with its index in another memory block. Finally, the nearest search in that memory determines indices of the closest points.

D. Distance Update

After clustering two data points with the closest distance, the relative distances of all data points with the merged nodes need to be updated. DUAL supports all popular linkage distance update used for clustering algorithms. Here, we first explain the implementation of the *Ward method*, which is a popular and complex update method. Then, we explain how DUAL supports other linkages using the same operations.

Ward Update: Ward update requires all data points to update their similarity with the new cluster data points (a_i and a_j). Ward update has three coefficients and three distance values (explained in Section II). Our design first computes the numerator and denominator of the coefficients by: (i) performing a row-parallel write of the size of s_i and s_j on two columns of the coefficient block (Figure 6**C**). This write in all rows is performed in a single cycle. (ii) since the weight corresponding to each data point is already stored in the memory (s_K column), we perform row-parallel in-memory addition to compute $X:\ s_i + s_k$ and $Y:\ s_j + s_k$, and $Z:\ s_i + s_j + s_k$, the numerator and denominator of the Ward coefficients (Figure 6**D**). (iii) Next, we compute the coefficients by performing row-parallel in-memory division of X, Y, and s_k with the denominator (Figure 6**E**) (explained in Section IV-B) .

To compute new distance of all data points to $a_i \cup a_j$, we multiply each coefficient column with corresponding column of the distance memory storing $d(a_i, a_k)$, $d(a_j, a_k)$ (Figure 6**F**). We also multiply the third coefficient (C_3), with the distance of the clustered nodes, $d(a_i, a_j)$ which has been written in a row-parallel write on another column of the distance memory (Figure 6**C**). Finally, we add the first two terms of the Ward metric and subtract from it the third term stored in the same memory (Figure 6**G**). The result of this operation is stored in i^{th} and j^{th} columns of the distance memory. We also update $i^t h$ and $j^t h$ rows of the distance memory. Since these two nodes are clustered, we only update one of the rows (i^{th} rows if $s_i > s_j$), setting s_i to $s_i + s_j$ and unset the valid flag of s_j (Figure 6**H**).

Figure 7 shows the layout of DUAL operations performing hierarchical clustering in memory. All DUAL operations can perform column-wise using row-parallel search-based or arithmetic operations. Ideally, two memory blocks, i.e., data and distance blocks, can compute the entire clustering tasks. In section VI-A, we talk about the scalability of DUAL when the data size is much larger to fit into those memories.

Other Linkage Updates: DUAL supports single and complete linkage by performing compare operation between two columns of the distance memory ($d(a_i, a_k)$ and $d(a_j, a_k)$). This comparison is performed by subtracting the distance vectors in a row parallel way and looking at the sign bit of the subtracted vector. Depending on the sign bit, we select one of the distance values as a relative distance to clustered data points. DUAL supports average linkage using a similar approach as Ward linkage. We perform row-parallel write of the s_i and s_j values and multiply them with the corresponding columns of the

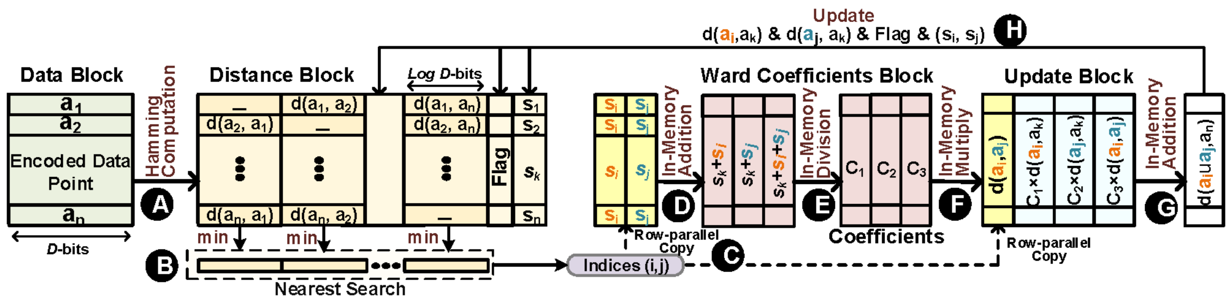


Fig. 6. Overview of DUAL functionalities supporting hierarchical clustering.

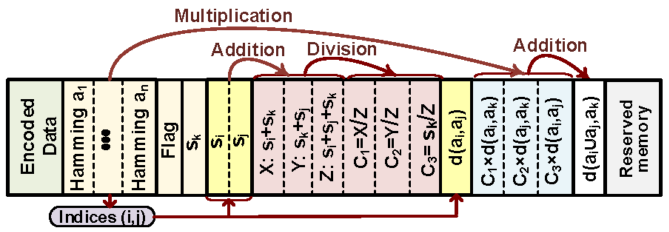


Fig. 7. memory Layout performing clustering.

distance values ($d(a_i,a_k)$ and $d(a_j,a_k)$). Then, DUAL adds the multiplied vectors and divides the results by $s_i + s_j$.

VI. DUAL Architecture

Figure 8Ⓐ shows an overview of the DUAL architecture. DUAL consists of 64 tiles. Each tile consists of 256 crossbar memories. Due to the existing challenges of crossbar memory [56], [57], each memory block is assumed to have a size of $1k \times 1k$. The memory blocks located in each row of a tile are connected together using an interconnect. This interconnect sends the signals from the CAM sense amplifier of the data block to row drivers of all distance blocks. The interconnect enables the Hamming distance computation of the data block to be written in any distance block located in the same row. To minimize the cost of the interconnect, we enable bit-serial/row-parallel data transfer which limits the interconnect bandwidth to 1K-bits. A single 3-bit counter is located at the top of each crossbar memory. During each sampling cycle of Hamming distance computation, the interconnects transfer the activate rows to a destination distance block, and the counter values will be written in parallel on all the activated rows (Figure 8Ⓑ).

A. Scalability & Parallelism

Figure 8Ⓒ shows DUAL configuration when the data points do not fit in a single memory. DUAL assigns the first block of each tile row to a data block while others are assigned to distance block storing the pair-wise distances. DUAL provides both row-level and block-level parallelism. To enable fast Hamming computing, our design exploits interconnects to write the result of distance computation in any distance block located in the same row. In addition, the bit-serial/row-parallel data transfer between the neighbor blocks accelerates DUAL computation (Figure 8Ⓓ). To enable parallelism, DUAL stores multiple copies of the data blocks in other tiles to perform the distance computation and clustering in parallel. Although this approach speeds up the computation, it adds two overheads: (i) during the clustering, data located in different tiles need to be aggregated into a single memory block, this results in a large internal data movement. (ii) In this configuration, DUAL also requires a larger memory to store repeated data blocks. In Section VIII-F, we explore the impact of parallelism on DUAL computation efficiency.

B. DUAL Pipeline

We design a pipeline that enables DUAL to work with maximum throughput. DUAL has two main phases: Hamming computing, and clustering. The Hamming computation happens only once, while the clustering phase repeats iteratively.

Hamming Computing Pipeline: Our pipeline initially assigns each distance block to store the relative Hamming distances corresponding to a single point. This starts by writing the Hamming distances relative to the first data point in the first distance memory. To maximize the throughput, DUAL continues the Hamming computing relative of the second data point and writes the results in the second distance block. At the same time, the first block accumulates all $D/7$ partial distances in order to represent them using $\log D$ bits. Typically, this accumulation is slow; thus the distance computation continues sequentially in multiple distance blocks until the first block becomes available. DUAL again uses the remaining bitlines of the first distance block ($1k - \log D$ bits) for distance computation of the next data points (Figure 8Ⓔ).

Clustering Pipeline: After creating a pairwise distance matrix, DUAL searches for the index with least Hamming distance in all distance blocks with valid flag bit set ("Nearest"). These indices along with the hamming distance are sent to another memory block which performs a comparison among the hamming distance to find the minimum distance ("Comp"). Based on the comparison result, DUAL first computes Ward coefficients and then transfers the corresponding vectors of the distance matrix into another block ("Data Transfer"). Finally, DUAL updates the distance vectors using the moved distance vectors and the computed Ward coefficients ("Distance Update"). This process continues iteratively until having one cluster.

C. Other Clustering on DUAL

DUAL is a general platform that can be used to accelerate a wide range of unsupervised learning algorithms. A similarity check is a common operation in most clustering problems. Here, we explain how DUAL can accelerate other popular algorithms by using Hamming similarity on the encoded data.

DBSCAN: is another popular clustering algorithm [58]–[60]. DBSCAN starts the clustering from an initial data point. For the selected data point, it computes the Hamming distance with all encoded data points and clusters it to the data point

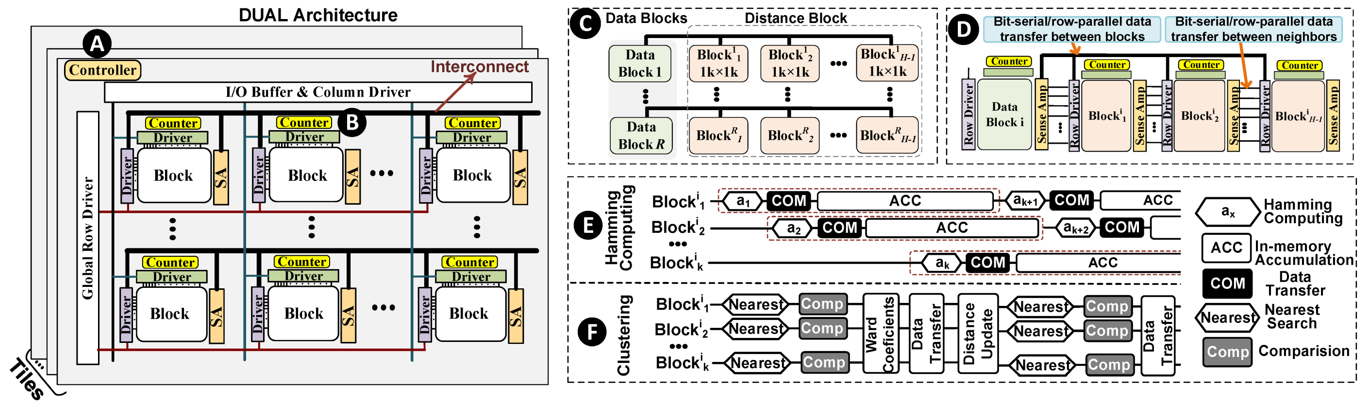


Fig. 8. DUAL architectural details, and the pipeline stage supporting Hamming computing and clustering.

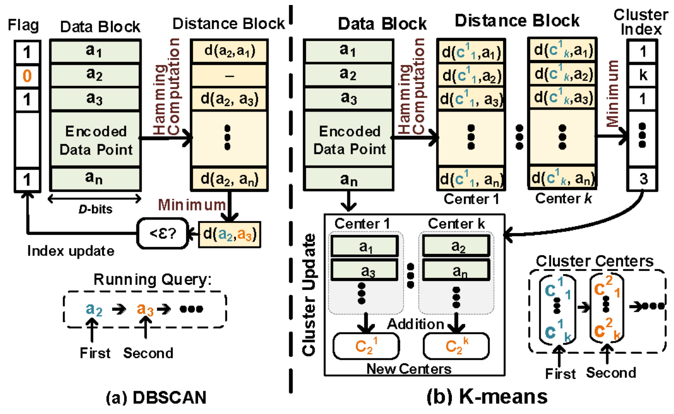


Fig. 9. DUAL computation flow accelerating DBSCAN and K-means algorithm.

with the closest similarity. This clustering happens if the point with the maximum similarity is within a pre-defined ε distance. If it is not, DBSCAN selects another initial point and continues the clustering until merging all close enough points.

We map DBSCAN to DUAL in a very similar way to hierarchical clustering. Figure 9a shows the computation steps of DUAL accelerating DBSCAN. Our design stores all encoded data points in a data block. Then, it computes the Hamming distance of an initially selected data with all other data points in a row-parallel way. The partial Hamming distance values will be written in a distance memory and then accumulates in order to represent using a single *logD* bits value. Similar to hierarchical clustering, DUAL searches for the minimum Hamming distance value in a distance memory. The controller checks the distance of the selected point with ε and clusters selected points if their distance is within an acceptable range. The clustering happens by simply activating the flag bit of the clustered point, indicating that the point does not need to be involved in future similarity checks. The same procedure repeats by considering the recently clustered data point as a new clustering query.

K-means: algorithm starts the clustering with a set of generated cluster centers [61]. The clustering continues by checking the similarity of each data point with all cluster centers and assigning it to a cluster with the highest similarity. The k-means stops the clustering if the changes between the centers in two consecutive iterations is less than a pre-defined ε.

Algorithm 1: The implementation of DBSCAN

```
0: $centers = vlca¡10000¿[N]$
0: $(cur\_p, cur\_err) = (0, infinite)$
0: while $cur\_err > thres$ do
0:     $h\_dist = hamming(centers[cur\_p], centers)$
0:     $(idx, vec) = near\_search(h\_dist, 0)$
0:     $(cur\_p, cur\_err) = (idx, vec - centers[cur\_p])$
0: end while=0
```

Figure 9b shows the computation steps of DUAL accelerating k-means. DUAL exploits the same hardware to compute the distance of centers $\{c_1^1, c_2^1, \cdots, c_2^1\}$ to all data points stored in the data block. The result of distance computation will be written in all distance blocks (k columns each storing the distance relative to a center). Since the cluster centers are usually small, even a single distance block might be enough to store Hamming distances of all centers with data points (if $k \times logD < 1k$, where k is the number of centers). For each data point, DUAL compares the distance values over all centers using a series of row-parallel subtraction and writes the index of a center with the minimum distance on the *index buffer*. This minimum functionality is implemented in a row-parallel way by comparing the distance values two-by-two, starting from their most significant bits. After finding the index corresponding to all data points, we activate all rows of the data block corresponding to center 1. DUAL performs in-memory addition to accumulate the activated rows. In the next iterations, DUAL accumulates data points corresponding to other centers. The k accumulated values are the new centers. To stay in the binary, the controller binarizes the new centers and repeaters the clustering using the new centers. The controller also checks for the converges and stops the algorithm if the number of bit changes of cluster centers in two consecutive iterations is less than a pre-defined value.

VII. Programming Support

A. Variable-Length Column Array

All DUAL operations (i.e., search and arithmetic) are column-wise. So, we introduce a family of data structures, Variable-Length Column Array (VLCA), to represent all operation values used in DUAL programs. Here, we denote a D-bit VLCA with N elements as $vlca\langle D\rangle[N]$. We enable two-dimensional indexing in VLCA where $vlca\langle D\rangle[i][j]$ represents j^{th} bit of i^i row in a D-bit vector. VLCA supports *slicing* indexing which

TABLE I
PIM INSTRUCTIONS OF DUAL.

Instruction	R. Reg.	W. Reg.
`set_qinput`	b, <addr>, <size>	q
`hamm_7`	b, c1, c2	-
`add/sub/mul/div`	b, d, c1, c2, c3	-
`near_search`	b, nc, c, q	rst, idx
`row_mv`	b1,r1,c1,b2,r2,c2,nr,nc	-

enable programmers to extract specific rows/columns from the whole data. For instance, $vlca\langle D\rangle[i:j][n:m]$ denotes data slice of $n^{th}-m^{th}$ columns in the $i^{th}-j^{th}$ rows. A VLCA, which cannot fit in one memory block, is stored across multiple blocks. Operations on long VLCAs can be broken into multiple parallel operations over all blocks storing the vector.

B. Built-in Functions

We define several built-int functions as C library for implementing operations in DUAL using VLCAs as main operators.

Hamming Computing: `hamming`($input$, $refs$) function computes the Hamming distance between an input vector ($input$) and an array of reference vectors ($refs$). The function has two parameters, where $input$ is a D-bit vector and $refs$ has a format of $vlca\langle D\rangle[N]$. Calculations of the Hamming distance is completed by comparing the vectors in a 7-bit windows. Each 7-bit comparison generates 3-bit distance result. The output of `hamming()` function has a format of $vlca\langle 3\rangle[\lceil\frac{D}{7}\rceil][N]$.

Row-Parallel Arithmetic: For each of the arithmetic operations, we use a row-parallel computation in memory. (e.g. `addition`($input1$, $input2$)) has two parameters, where both inputs and the output have the format of $vlca\langle D\rangle[N]$.

Nearest Search: The function `near_search`($input$, $target$) is used to finish the nearest search operation. The nearest search finds the entry in a $vlca\langle D\rangle[N]$ ($input$) with a value which is closest to a target D-bit vector ($target$). The output of `near_search()` function consists of the index of the matched entry as well as the value of the entry.

Row-Parallel Data Transfer: We copy the value of a vector to another vector by normal assignment statements $a=b$, with the same vector dimensionality. Data movements between VLCAs are processed by the PIM hardware in a row-parallel way, resulting in a single bit movement for all memory rows.

C. PIM Interface

Algorithm 1 shows the DBSCAN implementation using DUAL interface. We implement a runtime library to transform the function calls into DUAL instructions issued through a custom device driver (listed in Table I). There are several specialized registers required for these PIM instructions. Registers starting with b, r, and c store values indicating the memory location in terms of the block, row, and column respectively. Register q stores query data. nr and nc represent the number of rows and columns for the current instruction.

The mapping of the function to the PIM instruction is straightforward; to allocate a VLCA, we should find enough space to store the vectors in consecutive rows. In our implementation, we exploit a simple management scheme that uses a list of free blocks with a global allocation table. Once an allocation is issued, it checks the free block list to return one or multiple pages with consecutive rows and adds an allocation entry into the allocation table to store the allocation information including the address, bit-width, and the number of elements. When an address is reclaimed, it returns the corresponding blocks to the list and merges the list items if needed. The more advanced management scheme (e.g., handling memory fragmentation) is out of our scope; there exist many solutions for similar problems, e.g., SSD and persistent memory [62], [63].

D. Application Mapping

In DUAL, a register is located next to each memory that stores the sequence of operations that needs to be computed on the memory block. Each register, which acts as a controller for each memory block, is initialized once using the tile's controller. This initialization happens depending on several parameters including: clustering algorithm, number of pipeline stages, dimensionality, number of clusters, number of data points, and computation precision. Depending on these factors, our software interface estimates the worst-case memory requirement for each pipeline stage and decides the suitable data layout that results in maximum parallelism. We pre-evaluate each clustering algorithm once offline. This enables our software interface to find an optimal data layout and register/instructor values using a very limited algorithm and workload parameters.

VIII. RESULTS

A. Experimental Setup

We have designed a cycle-accurate simulator based on scikit-learn [64], [65] that emulates DUAL functionality during different clustering algorithms. For the hardware design, we use HSPICE for circuit-level simulations to measure the energy consumption and performance of all the DUAL operations in 28nm technology. Energy consumption and performance are also cross-validated using NVSim [66]. We used system Verilog and Synopsys *Design Compiler* [67] to implement and synthesize the DUAL controller. For parasitics, we used the same simulation setup considered by work in [53]. In DUAL, the interconnects are model in both circuit and architecture levels. In circuit-level, we simulate the cost of inter-tile communication while in architecture we model and evaluate intra-tile communications. The robustness of all proposed circuits, i.e., interconnect, has been verified by considering 10% process variations on the size and threshold voltage of transistors using 5000 Monte Carlo simulations. DUAL works with any bipolar resistive technology which is the most commonly used in existing NVMs. In order to have the highest similarity to commercially available 3D Xpoint, we adopt the memristor device with a VTEAM model [68]. The model parameters of the memristor are chosen to produce a switching delay of 1ns, a voltage pulse of 1V and 2V for RESET and SET operations in order to fit practical devices [48], [49].

Table II shows detailed configurations of DUAL consisting of 64 tiles. Each tile has 256 crossbar memory blocks. In each tile, the crossbar memory takes the majority of the area and power consumption, while the counters are taking less than 0.7% and 3.1% of the tile area and power. Each tile takes $0.84mm^2$ area and consumes 1.76W power. The total DUAL area and average power consumption are $53.57mm^2$ and 113.51W respectively.

Table III lists the energy consumption, execution time, and the required memory of each DUAL operation. All results are

TABLE II
DUAL PARAMETERS.

Components	Param	Spec	Area	Power
Crossbar array	size	1Mb	$3136\mu m^2$	6.14mW
Sense Amp	number	1K	$57.13\mu m^2$	2.38mW
Counter	number	1	$24.06\mu m^2$	0.27mW
Memory Block	number	1	$3217.19\ \mu m^2$	8.79mW
Tile Memory	num_block	256	$0.82mm^2$	1.57W
Interconnect	num_wire	1k/row	$0.01mm^2$	62.08mW
Controller	number	1	$289.2\mu m^2$	131.75mW
Tile	size	32MB	$0.84mm^2$	1.76W
Total	number size	64 Tiles 2GB	**53.57**mm^2	**113.51**W

TABLE III
DETAILS OF DUAL SUPPORTED OPERATIONS.

Operations	Size	Energy Consumption	Execution Time	Required Memory
Hamming Computing	7-bits	1632fJ	200/100 ps	3-bits/row
Nearest Search	4-bits	1214fJ	200 ps	1-bit/row
Addition	8-bit	2.3pJ	98.4ns	12-bits/row
Multiplication	8-bit	67.7pJ	448.3ns	155-bits/row
Division	8-bit	72.5pJ	561.4ns	168-bits/row
Data Transfer	1-bit	748fJ	1.1ns	1bit/row

reported for a row-parallel case in 28nm technology node when we perform the computation on a single block with 1k rows. In contrast to conventional CAM architectures that consume a huge amount of power, DUAL enables the search operation in 4-bi/7-bit granularity's, resulting in very lower power density.

B. Workloads

We evaluate DUAL efficiency on three popular clustering algorithms: hierarchical clustering, K-means, and DBSCAN. The evaluations are performed on several large-scale datasets including actual and synthetic datasets. Table IV lists 7 popular datasets selected from UCI machine learning website [69]. We also evaluated DUAL efficiency on large-scaled synthetic data consisting of 10k, 1 million, and 10 million data points. The synthetic data is generated random data with 100 cluster centers, radius range of $[r_l, r_h] = [0..\sqrt{2}, \sqrt{2}..\sqrt{32}]$, and noise rate of 0-10%. Synthetic data has differet sizes, from 400 MB (Synthetic 1) to 40 GB (Synthetic 3). To measure cluster quality, we rely on correct labels of data points and find out how many points were classified in a cluster that does not reflect the label associated with the point. For assigning a label to a cluster, we find a label that is repeated the maximum number of times in a cluster and assign that label to the cluster. We set the number of clusters formed to be the same as the number of labels available in the data set.

We compare DUAL with the efficient implementation of clustering algorithms running on GPU. For hierarchical clustering, we used the NVIDIA Graph Analytics library (nvGRAPH) [70]. We used [71] and [72] for GPU implementation of k-means and DBSCAN algorithms, respectively. The experiments are performed on an NVIDIA GTX 1080 GPU. The performance and energy of GPU are measured by the `nvidia-smi` tool.

C. Quality of Clustering

Figure 10a compares the quality of DUAL on three clustering algorithms. For DUAL, each data point is encoded to $D = 4{,}000$ dimensions. The results are compared to the baseline algorithms working with original data and using Euclidean distance. Our result shows that DUAL provides comparable accuracy to the

TABLE IV
WORKLOADS.

Datasets	# Data Point	# Features	# Clusters	Description
MNIST	60000	784	10	Handwritten Digits [73]
FACIAL	27965	300	2	Grammatical Facial Expressions [74]
UCIHAR	7667	561	12	Human Activity Using Smartphones [75]
SEIZURE	11500	178	5	Epileptic Seizure [76]
SENSOR	13910	129	6	Gas Sensor Array Drift [77]
GESTURE	9880	50	5	Gesture Phase Segmentation [78]
ISOLET	7797	617	26	Speech data [79]
Synthetic 1	100k	1000	50	100k data points
Synthetic 2	1M	1000	50	1 Millions data
Synthetic 3	10M	1000	50	10 Millions data

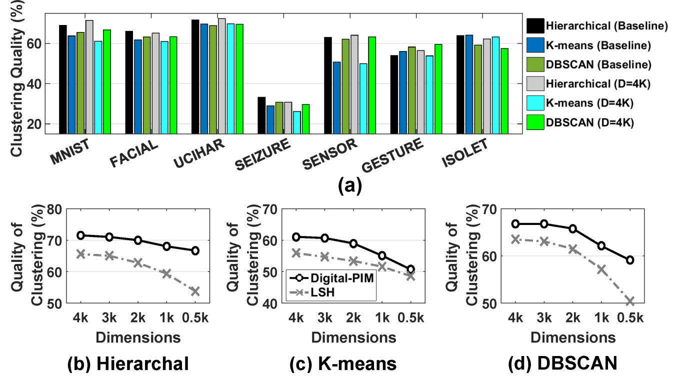


Fig. 10. (a) The quality of DUAL clustering and the baseline algorithm. (b-d) comparison of DUAL and LSH when mapping data to different dimensionality.

baseline clustering algorithms. For example, over hierarchical and DBSCAN (and k-means) DUAL provides on average 1.2% and 0.4% higher (1.3% lower) average quality of clustering as compared to the baseline algorithms.

We also compare the quality of DUAL with the clustering algorithm using Locality-Sensitive Hashing (LSH) [34], [80], [81]. Similar to DUAL, LSH can map data points to binary vectors with large dimensionality in order to replace Euclidean to hardware-friendly Hamming distance metric [24], [82], [83]. Figure 10b-d compares the quality of clustering in DUAL and LSH-based approaches. The results are shown for the MNIST dataset. Our evaluation shows that in the same dimensionality, DUAL provides a significantly higher quality of clustering than LSH-based approaches. For example, DUAL using $D = 4{,}000$ provides 5.9%, 5.2%, and 3.3% higher quality of clustering than LSH-based approach implementing hierarchical, k-means, and DBSCAN clustering. This higher quality comes from the non-linearity of the HD-mapper that keeps the similarity of the original data in high-dimensional space, while LSH tries to keep approximate distances in a linear way. In addition, we observe that algorithms have different sensitivities to dimension reduction. For example, DUAL accelerating hierarchical clustering can provide a high quality of clustering even when dimensionality reduces to $D = 2{,}000$. In contrast, k-means is more sensitive to dimension reduction.

Figure 11 visualizes the clustering of UCIHAR dataset using the baseline hierarchical clustering and DUAL using $D = 4{,}000$ and $1{,}000$. For visualization, we used `t-SNE` technique [84] which represents high-dimensional data in 2-dimensional space. In the UCIHAR dataset, the clustering space is 561-dimensional. The true cluster labels are indicated by different colors. The visualization indicates that DUAL using $D = 4{,}000$ results in

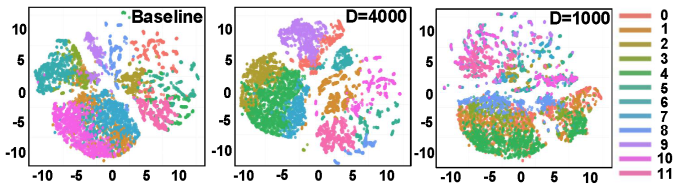


Fig. 11. t-SNE visualizations of the baseline hierarchical learning and DUAL with different dimensionalities.

a more clustering-friendly space compared to clustering in the original space. As we decrease the dimensionality of the mapped hypervectors, DUAL quality of clustering reduces. For example, DUAL using $D = 1,000$ provides 5.7% lower quality as compared to DUAL using $D = 4,000$ dimensionality.

D. DUAL Efficiency

Figure 12 compares DUAL energy efficiency and performance with the baseline clustering algorithms running on GPU. Our evaluation shows that DUAL running all clustering algorithms provides on average 58.8× speedup and 251.3× energy efficiency improvement as compared to the baseline GPU-based approach. The higher DUAL efficiency comes from: (i) enabling a large amount of parallelism, supported by ensuring the data availability in each block. The GPU has resources/cores to parallelize up to four thousand operations. Due to the weakness of the existing von-Neumann architectures, even these small number of cores are usually underutilized (28% utilization running hierarchical clustering). In contrast, DUAL can perform up to 8 million parallel operations by enabling row-level and block-level parallelisms. We also maximize DUAL utilization using the proposed pipelined that ensures data availability in each core, i.e., memory block. (ii) DUAL addresses the overhead of data movement by not only eliminating the external data movement (between the processing cores and memory), but also eliminating the internal data movement between the memory blocks. This internal data transfer is minimized by exploiting the interconnects and bit-serial/row-parallel data transfers between the neighbor blocks.

DUAL is an algorithm-hardware co-design. Without HD-mapper, DUAL cannot support euclidean distance entirely in PIM. Similarly, without DUAL Hamming computing hardware, using HD-mapper does not provide computation efficiency. We run DUAL code (high-dimensional clustering) on the same GPU as the baseline. We observe that clustering in high-dimensional space using HD mapper (LSH mapper) runs on average 12.8× and 3.1× (2.8× and 1.06×) slower and less energy efficient than clustering on the original space running on the same GPU. This is because GPUs have lower parallelism (# of cores) than PIM architecture, thus they get higher benefit running arithmetic operations on low-dimensional vectors, rather than binary computation over long vectors. In other words, DUAL efficiency comes from revisiting clustering algorithms based on the hardware/technology requirements.

Algorithms Efficiency: DUAL efficiency depends on the operations involved in each algorithm; a portion of search-based and arithmetic operations. In search-based operations, DUAL is significantly faster and more efficient than the equivalent CMOS-based logic. In terms of arithmetic operations, DUAL efficiency highly depends on the amount of parallelism. In fact, in a single arithmetic operation, e.g., 32-bit multiplication, DUAL is about 60× slower than CMOS-based logic. This is because our approach supports arithmetic using a series of NOR-based operations. However, the large amount of parallelism supported by DUAL results in a higher overall DUAL efficiency. Looking at different algorithms, DUAL provides the maximum efficiency over hierarchical clustering and DBSCAN as these algorithms are mostly involved in search-based operations. For example, DUAL running hierarchical clustering (DBSCAN) provides on average 67.1× (71.7×) speedup and 328.7× (293.3×) energy efficiency as compared to a GPU-based approach. In contrast, k-means involves a large amount of slow arithmetic operation (during cluster update), thus providing only 37.5× speedup and 131.6× energy efficiency as compared to GPU.

DUAL Configurations: To show the impact of each optimization, Figure 12 shows the DUAL efficiency without using the proposed interconnects and counters. Our evaluation shows that DUAL without interconnects can still outperform GPU efficiency. The impact of the interconnects is more crucial on the hierarchical clustering algorithm (3.9× slow down without interconnect), as it requires interconnects to write all pairwise computations on different distance blocks located in the same row. In k-means, DUAL computes the distance to a limited amount of cluster centers; thus, datasets with more number of cluster centers are more affected by interconnect elimination. DBSCAN has the least sensitivity to interconnects (1.6× slow down without interconnect), as it computes the Hamming distance to a single data point at each iteration. Therefore, it uses very few neighbor distance blocks to store the distance vector.

Figure12 also shows DUAL efficiency without counter blocks. As NVMs are slow in the write operation, without the counter, the Hamming computing will be significantly slow down. This slow down in more obvious on hierarchical clustering as Hamming computing takes 41% of the total GPU execution. For example, the results show that without counters DUAL works on average 2.7×, 2.1×, and 2.4× slower than the baseline DUAL running hierarchical, k-means, and DBSCAN algorithms.

E. DUAL Quality-Efficiency Tradeoff

As we explained in Section VIII-C, DUAL quality of clustering depends on the dimensionality of the encoded data. We exploit the robustness of DUAL to dimension reduction in order to improve the computation efficiency. DUAL using lower dimension provides: (i) faster computations, e.g., fewer iterations for Hamming computing and nearest search, and accordingly lower bit-width to perform distance update. (ii) In addition, a lower D reduces the amount of memory requirement and internal data movement between the blocks during the cluster update. Figure 13 shows the impact of dimension reduction on DUAL computation efficiency. The results are reported when we ensure less than 1% and 2% quality loss on all tested datasets as compared to DUAL using full dimensionality ($D = 4,000$). Our evaluation shows that DUAL computation efficiency depends on the clustering algorithm. Hierarchical clustering has high robustness to dimension reduction (as shown in Figure 10b), thus provides 90.6× and 443.9× (116.7× and 572.2×) speedup and energy efficiency while providing only 1% (2%) quality loss. In contrast, k-

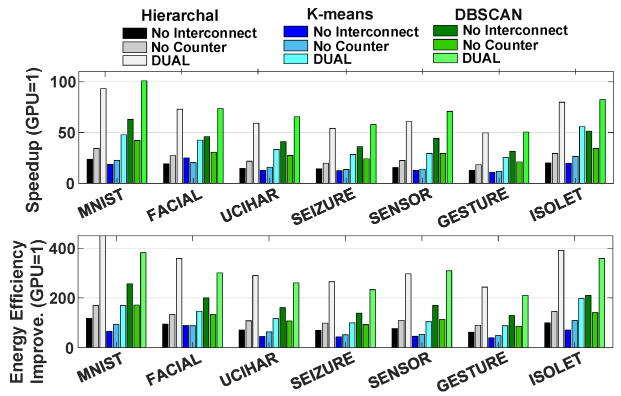


Fig. 12. Efficiency in different configurations.

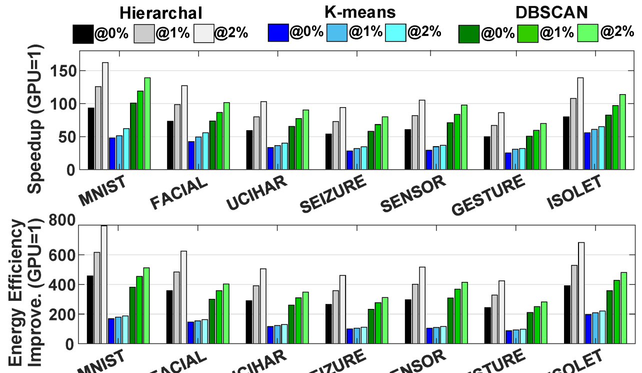


Fig. 13. Efficiency in different levels of quality loss.

means quality changes significantly with dimension reduction (as shown in Figure 10c), resulting in only 42.2× and 139.5× (46.5× and 146.4×) average speedup and energy efficiency improvement with 1% (2%) quality loss.

F. Parallelism & Scalability

Figure14 shows the energy consumption and execution time of DUAL providing different levels of parallelism. The results are normalized to DUAL using a single copy of encoded data (parallelism:1). In this low-power mode, DUAL saves the encoded data in the data blocks, enabling serially Hamming distance computation over data points. The clustering task can be also performed without significant data transfer between the blocks and tiles. In high-performance mode, DUAL saves multiple copies of the encoded data points into a single or different tile, providing a parallel Hamming computing. However, this parallelism comes at the expense of increasing the number of data transfer between the blocks, especially during the clustering phase. We evaluate the efficiency of parallelism over two different dataset sizes: 1K and 100K data points. Our results show that using small datasets, DUAL computation speeds up linearly with the level of parallelism. However, using large datasets, the overhead of internal data transfer saturates the DUAL speedup. We also observe that hierarchical clustering requires a lower amount of parallelism for performance saturation. This is because hierarchical clustering stores all pairwise distance values, requiring more blocks/tiles to cluster the same number of data points. In contrast, k-means and DBSCAN only store *k* (number of cluster centers) and one distance similarity, providing higher computation density.

Figure 14 shows DUAL computation speedup for hierarichal clsutering when using synthetic datasets with 100K, 10M, and 100M data points (listed in Table IV). Similar to parallelism, increasing the number of chips results in higher performance, but increases the cost of data transfer between different chips. For example, increasing the level of parallelism by a factor of two, only results in 1.6× and 1.4× speedup over 100k and 10M data point. The lower improvement in larger datasets comes from the overhead of a large amount of data movement. In a fair comparison, DUAL using 16 chips provide the same area as an NVIDIA GPU. In this configuration, DUAL using 10M data points provides 621.1× and 4.6× speedup as compared to GPU and DUAL using 1-chip, respectively.

G. Breakdown

Figure 15a shows the average DUAL computation efficiency with In-Memory data-parallel Processor (IMP) [15]. IMP offloads the PIM-compatible operations of a program into analog-PIM. These operations are the addition, multiplication, and dot product. For hierarchical clustering and DBSCAN, IMP can only offload/accelerate the similarity search (Euclidean distance), which only takes 24.5% and 29% of total GPU execution (Figure 15b). This results in about 1.6× and 1.3× speedup over GPU. In contrast, in k-means, IMP operations can be used to accelerate both similarity check and cluster update (92% of GPU execution as shown in Figure 15b), resulting in 12.1× speedup and 27.2× energy efficiency improvement as compared to GPU. DUAL using a 4-chips provides a similar chip area as IMP. In this configuration, DUAL provides 136.2×, 9.8×, and 168.1× speedup than IMP over hierarchical clustering, k-means, and DBSCAN, respectively. This efficiency mainly comes from DUAL capability in processing all clustering operations in a parallel.

Figure 15b shows the breakdown of the DUAL operations running different clustering applications. Our evaluation shows that DUAL encoding module takes less than 5% of the total execution time. For hierarchical clustering, clustering (nearest search) dominates the execution, as DUAL performs clustering functionality multiple times over the distance matrix. For k-means, the center update takes the majority of the execution time, as DUAL arithmetic operations are slower than search-based operations. For DBSCAN, the center update is computationally simple; thus the Hamming computing and nearest search take the majority of the execution time.

H. DUAL Lifetime and Device Variability

We evaluate the lifetime of DUAL depending on the number of write operations in the memory devices. one of the main advantages of DUAL is its high robustness to noise and failure. DUAL store information as holographic distribution of patterns in high-dimensional space. In this representation, all dimensions are equally contributing to storing information. Therefore, failures on a dimension may not result in losing the entire data. Prior work shows that the endurance of the memristor device has ranged between 10^9 to 10^{11} [85]. During the lifetime of DUAL, the number of write operations affects the functionality of the device. Here, we manage the DUAL endurance by uniformly distributing the number of writes to all bitlines.

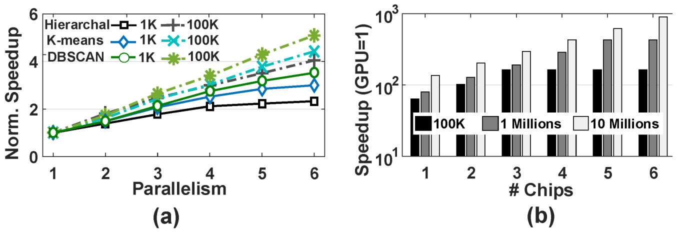


Fig. 14. (a) DUAL speedup in different level of parallelism, (b) scalability using different # of chips.

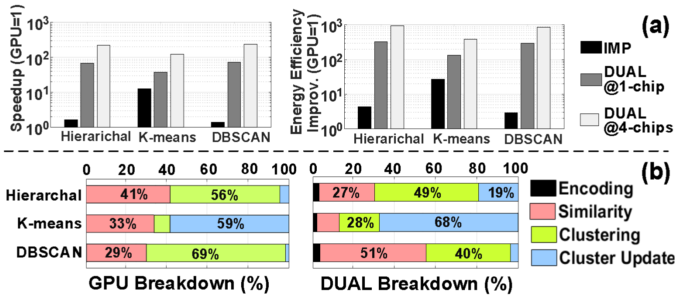


Fig. 15. (a) DUAL vs. IMP [15] efficiency. (b) Computation breakdown of GPU and DUAL.

Since all memory blocks support the same functionality, in a long time period, DUAL uses different blocks as data blocks. In addition, DUAL exploits different columns of each memory to perform arithmetic operations. Our evaluation shows that assuming the arrays are continuously used, DUAL can work accurately for 13.5 years. Considering a Gaussian distribution for device failure (endurance distribution), DUAL will still work with less than 1% and 2% quality loss after 17.2 and 19.6 years. In DUAL, each tile controller keeps track of the number of times that each memory block is used for computation. This provides an estimation of the number of writes in the memory without paying significant cost for endurance management.

In practice, the memristor devices might have variation due to thermal or process variations. Here, we consider the impact of up to 50% variations on the OFF/ON memristor values. Our evaluation shows that decreasing the R_{off}/R_{on} slows down the arithmetic and search-based operations. In 50% variations ($R_{off}/R_{on} \sim 50$), DUAL ensures the exact computation by using 350ps and 1.8ns clock frequency for the search and NOR-based operations. In architecture-level, this variation results in 1.83× slower and 1.45× less energy efficiency of DUAL. However, DUAL efficiency is still much higher than the GPU.

IX. Related Work

Clustering Acceleration: Several prior works tried to accelerate clustering algorithms on GPU, FPGA, and ASIC designs [72], [86]–[90]. To accelerate computation, prior works tried to simplify clustering operations [35]–[37]. For example, Locality Sensitive Hashing (LSH) was introduced in order to give an efficient algorithm for nearest neighbor search in high-dimensional space. This approach simplifies the similarity search of hashed data to the Hamming distance metric which can be implemented more efficiently in the hardware [91], [92]. However, the current computing systems are still significantly slow in processing large datasets, as the main computation still relies on CMOS-based cores, thus has limited parallelism.

Processing in-memory: To address the data movement issue, work in [93] proposed a neural cache architecture that re-purposes caches for parallel in-memory computing. Work in [94], [95] modified DRAM architecture to accelerate DNN inference by supporting matrix multiplication in memory. In contrast, DUAL performs a row-parallel and non-destructive bitwise operation inside non-volatile memory without using any sense amplifiers. In addition, these approaches do not support Hamming computing and the nearest search which are essential for clustering applications. The capability of non-volatile memories (NVMs) to act as both storage and processing units has encouraged research in Processing In-Memory (PIM) [16], [17], [96]. NVM-based PIMs have been used to accelerate a wide range of big data applications such as supervised learning [14], [17], [18], [96]–[99], graph processing [9], [11], [100]. Work in [101], [102] designed NVM-based Boltzmann machine capable of solving a broad class of deep learning and optimization problems. Work in [18] and [17] designed new architectures to accelerate Deep Neural Network (DNN) inference using analog-PIM. Work in [14], extended PIM functionality to support the training phase of DNN algorithms. Work in [15] exploited analog-PIM to accelerate general data-intensive workloads by offloading the PIM-compatible operations into the PIM accelerator. However, these architectures: (i) do not support the majority of operations involved in clustering algorithms [16], [18], e.g., similarity/nearest search, (ii) require ADC/DAC blocks that dominate total chip area/power, e.g., 98% of chip area and 87% of total power in [18].

In addition, work in [16] proposed a PIM architecture that supports floating-point arithmetic operations over digital data. However, all existing clustering algorithms are based on the search-based operations that cannot be supported by arithmetic operations. Prior works also exploited digital PIM operations to accelerate different applications such as DNNs [46], [103], [104], brain-inspired computing [39], [105], [106], object recognition [107], graph processing [108], [109], and database applications [45], [110]. However, the designs do not support high-level operations used in clustering. In contrast, DUAL supports all operations involved in clustering algorithms using parallelized in-memory operations. It also removes the necessity of ADC/DAC blocks; thus providing high throughput/area.

Search-based PIMs: There are several work used NVM-based CAM to support the nearest absolute [46] or Hamming distance [39] search. These approximate analog searches cannot be used to perform clustering that required actual distance value. In contrast, DUAL is the first digital-based architecture that computes Hamming distance without using costly ADC/DAC blocks. There are recent work used other NVM technologies, e.g., FeFET and STT-RAMs or multi-level cells (MLCs), to design CAM blocks [111]–[113]. For example, work in [114] designed CAM with exact search capability using MLCs. DUAL idea is orthogonal to NVM technology as it can exploit the same peripherals to support Hamming computing on any CAMs.

X. Conclusion

In this paper, we propose the first digital-based processing in-memory architecture (DUAL) to accelerate a wide range of popular unsupervised learning algorithms. DUAL maps all data points into high-dimensional space, replacing complex clustering operations with memory-friendly operations. We accordingly designed a PIM-based architecture that supports all essential operations in a highly parallel and scalable way. Our evaluation shows that DUAL provides a comparable quality of clustering to existing clustering algorithms while providing a 58.8× speedup and a 251.2× energy efficiency improvement as compared to the existing NVIDIA GTX 1080 GPU.

Acknowledgements

This work was partially supported in part by SRC-Global Research Collaboration grant AIHW 2020, in part by SRC JUMP CRISP program, and also NSF grants #1527034, #1730158, #1826967, and #1911095. Mohsen Imani and Yeseong Kim are co-corresponding authors of the paper.

References

[1] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of big data*, vol. 2, no. 1, p. 8, 2015.

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

[3] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 13, 2013.

[4] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *Advances in neural information processing systems*, pp. 521–528, 2003.

[5] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer Science & Business Media, 2012.

[6] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "Cd-hit: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.

[7] X. Cai, F. Nie, and H. Huang, "Multi-view k-means clustering on big data," in *Twenty-Third International Joint conference on artificial intelligence*, 2013.

[8] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE transactions on emerging topics in computing*, vol. 2, no. 3, pp. 267–279, 2014.

[9] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[10] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The terasys massively parallel pim array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[11] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 336–348, IEEE, 2015.

[12] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: throughput-oriented programmable processing in memory," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pp. 85–98, ACM, 2014.

[13] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, *et al.*, "The architecture of the diva processing-in-memory chip," in *Proceedings of the 16th international conference on Supercomputing*, pp. 14–25, ACM, 2002.

[14] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, IEEE, 2017.

[15] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1–14, ACM, 2018.

[16] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 802–815, ACM, 2019.

[17] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[18] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[19] V. Batagelj, "Generalized ward and related clustering problems," *Classification and related methods of data analysis*, pp. 67–74, 1988.

[20] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.

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

[22] V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu, "Hierarchical clustering: Objective functions and algorithms," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 378–397, SIAM, 2018.

[23] A.-A. Liu, Y.-T. Su, W.-Z. Nie, and M. Kankanhalli, "Hierarchical clustering multi-task learning for joint human action grouping and recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 1, pp. 102–114, 2017.

[24] H. Koga, T. Ishibashi, and T. Watanabe, "Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing," *Knowledge and Information Systems*, vol. 12, no. 1, pp. 25–53, 2007.

[25] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *The Computer Journal*, vol. 26, no. 4, pp. 354–359, 1983.

[26] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel computing*, vol. 21, no. 8, pp. 1313–1325, 1995.

[27] S. Bandyopadhyay and E. J. Coyle, "An energy efficient hierarchical clustering algorithm for wireless sensor networks," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, vol. 3, pp. 1713–1723, IEEE, 2003.

[28] 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.

[29] O. Yim and K. T. Ramdeen, "Hierarchical cluster analysis: comparison of three linkage measures and application to psychological data," *The quantitative methods for psychology*, vol. 11, no. 1, pp. 8–21, 2015.

[30] "Wikipedia linkage." https://en.wikipedia.org/wiki/Hierarchical_clustering.

[31] "Ward method." https://en.wikipedia.org/wiki/Ward's_method.

[32] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, p. 13, 2013.

[33] L. K. John and E. E. Swartzlander, "Memristor-based computing," *IEEE Micro*, no. 5, pp. 5–6, 2018.

[34] "Lsh clustering." https://github.com/usc-isi-i2/dig-lsh-clustering.

[35] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

[36] S. Kanj, T. Brüls, and S. Gazut, "Shared nearest neighbor clustering in a locality sensitive hashing framework," *Journal of Computational Biology*, vol. 25, no. 2, pp. 236–250, 2018.

[37] J. Zamora, M. Mendoza, and H. Allende, "Hashing-based clustering in high dimensional data," *Expert Systems with Applications*, vol. 62, pp. 202–211, 2016.

[38] M. Imani, S. Bosch, M. Javaheripi, B. Rouhani, X. Wu, F. Koushanfar, and T. Rosing, "Semihd: Semi-supervised learning using hyperdimensional computing," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8, 2019.

[39] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.

[40] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 64–69, 2016.

[41] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, pp. 1177–1184, 2008.

[42] B. Schölkopf, "The kernel trick for distances," in *Advances in neural information processing systems*, pp. 301–307, 2001.

[43] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[44] J. Li, R. K. Montoye, M. Ishii, and L. Chang, "1 mb 0.41 μm^2 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 4, pp. 896–907, 2013.

[45] M. Imani, S. Gupta, A. Arredondo, and T. Rosing, "Efficient query processing in crossbar memory," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2017.

[46] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.

[47] M. Imani, S. Gupta, Y. Kim, M. Zhou, and T. Rosing, "Digitalpim: Digital-based processing in-memory for big data acceleration," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 429–434, 2019.

[48] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[49] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, and S. Kvatinsky, "Simpler magic: synthesis and mapping of in-memory logic executed in a single row to improve throughput," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[50] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, no. 1, pp. 64–74, 2015.

[51] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: design principles and methodologies," *TVLSI*, vol. 22, no. 10, pp. 2054–2066, 2014.

[52] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[53] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[54] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky, "Efficient algorithms for in-memory fixed point multiplication using magic," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2018.

[55] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, "Truncapp: A truncation-based approximate divider for energy efficient dsp applications," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 1639–1642, Europe, 2017.

[56] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 476–488, IEEE, 2015.

[57] A. Nag, R. Balasubramonian, V. Srikumar, R. Walker, A. Shafiee, J. P. Strachan, and N. Muralimanohar, "Newton: Gravitating towards the physical limits of crossbar acceleration," *IEEE Micro*, vol. 38, no. 5, pp. 41–49, 2018.

[58] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[59] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[60] B. Borah and D. Bhattacharyya, "An improved sampling-based dbscan for large spatial databases," in *International Conference on Intelligent Sensing and Information Processing, 2004. Proceedings of*, pp. 92–96, IEEE, 2004.

[61] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 7, pp. 881–892, 2002.

[62] J. Xu and S. Swanson, "{NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories," in *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pp. 323–338, 2016.

[63] I. Oukid, D. Booss, A. Lespinasse, W. Lehner, T. Willhalm, and G. Gomes, "Memory management techniques for large-scale persistent-main-memory systems," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1166–1177, 2017.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[65] "Scikit-learn library." https://scikit-learn.org.

[66] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*, pp. 15–50, Springer, 2014.

[67] D. Compiler, R. User, and M. Guide, "Synopsys," *Inc., see http://www. synopsys. com*, 2000.

[68] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[69] "Uci machine learning repository." https://archive.ics.uci.edu/ml/index.php.

[70] "nvgraph." https://developer.nvidia.com/discover/cluster-analysis.

[71] "k-means gpu." https://github.com/NVIDIA/kmeans.

[72] G. Andrade, G. Ramos, D. Madeira, R. Sachetto, R. Ferreira, and L. Rocha, "G-dbscan: A gpu accelerated algorithm for density-based clustering," *Procedia Computer Science*, vol. 18, pp. 369–378, 2013.

[73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[74] "Grammatical facial expressions." https://archive.ics.uci.edu/ml/datasets/Grammatical+Facial+Expressions.

[75] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, pp. 216–223, Springer, 2012.

[76] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state," *Physical Review E*, vol. 64, no. 6, p. 061907, 2001.

[77] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, "Chemical gas sensor drift compensation using classifier ensembles," *Sensors and Actuators B: Chemical*, vol. 166, pp. 320–329, 2012.

[78] R. C. Madeo, C. A. Lima, and S. M. Peres, "Gesture unit segmentation using support vector machines: segmenting gestures from rest positions," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 46–52, ACM, 2013.

[79] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.

[80] H. Koga, T. Ishibashi, and T. Watanabe, "Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing," *Knowledge and Information Systems*, vol. 12, no. 1, pp. 25–53, 2007.

[81] X. Shen, W. Liu, I. Tsang, F. Shen, and Q.-S. Sun, "Compressed k-means for large-scale clustering," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[82] S. Pandit, S. Gupta, *et al.*, "A comparative study on distance measuring approaches for clustering," *International Journal of Research in Computer Science*, vol. 2, no. 1, pp. 29–31, 2011.

[83] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in neural information processing systems*, pp. 1061–1069, 2012.

[84] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[85] J. B. Kotra, M. Arjomand, D. Guttman, M. T. Kandemir, and C. R. Das, "Re-nuca: A practical nuca architecture for reram based last-level caches," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 576–585, IEEE, 2016.

[86] J. D. Hall and J. C. Hart, "Gpu acceleration of iterative clustering," in *The ACM Workshop on General Purpose Computing on Graphics Processors*, pp. 45–52, 2004.

[87] R. Wu, B. Zhang, and M. Hsu, "Clustering billions of data points using gpus," in *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pp. 1–6, ACM, 2009.

[88] J. Bhimani, M. Leeser, and N. Mi, "Accelerating k-means clustering with parallel implementations and gpu computing," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, IEEE, 2015.

[89] 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, IEEE, 2016.

[90] F. Winterstein, S. Bayliss, and G. A. Constantinides, "Fpga-based k-means clustering using tree-based data structures," in *2013 23rd International Conference on Field programmable Logic and Applications*, pp. 1–6, IEEE, 2013.

[91] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2938–2945, 2013.

[92] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *arXiv preprint arXiv:1408.2927*, 2014.

[93] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *arXiv preprint arXiv:1805.03718*, 2018.

[94] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 288–301, ACM, 2017.

[95] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 273–287, IEEE, 2017.

[96] Y. Zha, E. Nowak, and J. Li, "Liquid silicon: A nonvolatile fully programmable processing-in-memory processor with monolithically integrated reram," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 908–919, 2020.

[97] S. Angizi, Z. He, and D. Fan, "Dima: a depthwise cnn in-memory accelerator," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[98] S. Angizi, Z. He, and D. Fan, "Parapim: a parallel processing-in-memory accelerator for binary-weight deep neural networks," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 127–132, ACM, 2019.

[99] S. Angizi, Z. He, and D. Fan, "Pima-logic: a novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *Proceedings of the 55th Annual Design Automation Conference*, p. 162, ACM, 2018.

[100] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Graphr: Accelerating graph processing using reram," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 531–543, IEEE, 2018.

[101] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pp. 1–13, IEEE, 2016.

[102] M. N. Bojnordi and E. Ipek, "The memristive boltzmann machines," *IEEE Micro*, vol. 37, no. 3, pp. 22–29, 2017.

[112] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müeller, S. Beyer, M. Niemier, X. S. Hu, *et al.*, "Ferroelectric ternary

[103] S. Gupta, M. Imani, H. Kaur, and T. S. Rosing, "Nnpim: A processing in-memory architecture for neural network acceleration," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1325–1337, 2019.

[104] M. Imani, M. S. Razlighi, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Deep learning acceleration with neuron-to-memory transformation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–14, IEEE, 2020.

[105] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*, p. 55, ACM, 2018.

[106] M. Imani, X. Yin, J. Messerly, S. Gupta, M. Niemier, X. S. Hu, and T. Rosing, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[107] Y. Kim, M. Imani, and T. Rosing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 25–32, IEEE, 2017.

[108] M. Zhou, M. Imani, S. Gupta, and T. Rosing, "Gas: A heterogeneous memory architecture for graph processing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 1–6, 2018.

[109] M. Zhou, M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Gram: graph processing in a reram-based computational memory.," in *ASP-DAC*, pp. 591–596, 2019.

[110] M. Imani, S. Gupta, S. Sharma, and T. Rosing, "Nvquery: Efficient query processing in non-volatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[111] F. Zokaee, M. Zhang, and L. Jiang, "Finder: Accelerating fm-index-based exact pattern matching in genomic sequences through reram technology," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 284–295, IEEE, 2019.
content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[113] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging trends in design and applications of memory-based computing and content-addressable memories," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1311–1330, 2015.

[114] C. Li, C. E. Graves, X. Sheng, D. Miller, M. Foltin, G. Pedretti, and J. P. Strachan, "Analog content-addressable memories with memristors," *Nature communications*, vol. 11, no. 1, pp. 1–8, 2020.