Accelerating k-Medians Clustering Using a Novel 4T-4R RRAM Cell

Yomi Karthik Rupesh, Payman Behnam, Goverdhan Reddy Pandla, Manikanth Miryala, and Mahdi Nazm Bojnordi

***Abstract*— Clustering is a crucial tool for analyzing data in virtually every scientific and engineering discipline. The U.S. National Academy of Sciences has recently announced "the seven giants of statistical data analysis" in which data clustering plays a central role. This report also emphasizes that more scalable solutions are required to enable time and space clustering for the future large-scale data analyses. As a result, hardware and software innovations that can significantly improve energy efficiency and performance of the data clustering techniques are necessary to make the future large-scale data analysis practical. This paper proposes a novel mechanism for computing bit-serial medians within resistive RAM arrays with no need to read out the operands from memory cells. We propose a novel four-transistor, four-memristor memory cell that enables *in situ* median computation within the data arrays. (If necessary, the proposed cell could be used as four ordinary one-transistor, one-memristor memory cells to store four bits of information.) The proposed hardware is used to accelerate a data clustering library using breast cancer samples, indoor localization, and the U.S. Census data sets, as well as two applications using *k*-means clustering. Our simulation results for the library indicate an average performance improvement of 15.5× and an energy reduction of 28.5× over a baseline CPU system. Also, we observe an overall speedup of 5.8× with an energy improvement of 14.1× over a baseline processing-in-memory accelerator. For the *k*-means applications, we observe speedups of 45.7× and 1.5× with respective energy improvements of 49.5× and 1.3× as compared with the CPU baseline.**

***Index Terms*— Computer architecture, *in situ* processing, *k*-medians, resistive RAM (RRAM) technology.**

I. Introduction

DATA clustering is one of the most fundamental components in learning and understanding the natural structure of data sets. Clustering techniques are increasingly used in science and engineering for important applications, such as precision medicine [1], World Wide Web [2], machine learning [3], self-driving cars [4], business marketing [5], and economy [6]. Due to recent advances in sensor and storage technologies, as well as the significant growth in the applications of unsupervised learning, data clustering has become one of the most critical tools for the future computer systems.

K-means [7] is one of the most commonly used algorithms for solving data clustering problems in various fields of science and engineering. (Detailed background on object classification and k-means applications can be found in the literature [8].) For instance, iterative k-means clustering is used to identify cancerous samples [9] or to perform unsupervised learning tasks [10]. The algorithm partitions a set of input data into k clusters, each of which is represented with a *centroid*. The original algorithm relies on the arithmetic mean to compute the centroids; therefore, the results are very sensitive to outliers. In response to this problem, more robust variants of the algorithm, such as k-medians and k-medoids, have been proposed and used in the past [11]. In particular, k-medians achieves better solution quality by setting the centroid of each cluster to its median. However, it requires excessive memory accesses to the data points and significantly limits the overall performance. Numerous techniques have been proposed in the literature to accelerate k-medians clustering, such as precise and approximate software solutions [12]–[14], field-programmable gate array (FPGA) accelerators using sorting networks [15], [16], parallel probabilistic platforms [17], graphics processing unit (GPU) accelerators [18], and application-specific hardware frameworks [19]. Regrettably, the required data movement between the main memory and the processor cores limits the performance of these recent efforts even for moderately sized data sets. Moreover, with the growing interest in the future data intensive applications—such as deep learning applications that rely on unsupervised classification—the importance of high performance data clustering techniques is expected to increase.

This paper proposes a memory-centric hardware accelerator for *in situ* k-medians clustering based on bit-serial median rank order filters (ROFs) [20] and recently developed resistive RAM (RRAM) technology. A bit-serial ROF—often used in signal and image processing—is capable of identifying the ith median of an input data set, which may be used for k-medians clustering. The performance of a bit-serial ROF is significantly limited due to the excessive memory accesses required for clustering large data sets. The *in situ* k-medians accelerator addresses the problem by leveraging the computational capabilities of RRAM cells to realize a memristive platform capable of performing bit-serial rank order median computation *in situ* memory arrays, thereby unlocking the potential massive parallelism in k-medians clustering.

Manuscript received August 19, 2017; revised December 18, 2017; accepted February 8, 2018. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1755874 and in part by The University of Utah Seed. *(Corresponding author: Mahdi Nazm Bojnordi.)*

The authors are with The University of Utah, Salt Lake City, UT 84112 USA (e-mail: bojnordi@cs.utah.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2808468

Algorithm 1 Basic k-Means Clustering

```
1: select $k$ initial centroids randomly
2: **repeat**
3:     form $k$ clusters: assign data points to their closest centroids
4:     recompute the centroid of each new cluster
5: **until** convergence is reached
```

The proposed hardware accelerator is evaluated on a k-means clustering library using breast cancer, indoor localization, and the U.S. census data sets, and two applications that use k-means clustering. We observe that the proposed accelerator achieves an average performance improvement of 15.5× and an average energy reduction of 28.5× over the software implementation of k-means on a CPU system. Moreover, the results indicate an overall speedup of 5.8× with an energy improvement of 14.1× over a processing-in-memory (PIM) accelerator. When used for accelerating k-means clustering in gene expression analysis (GEA) and classification based on term frequency–inverse document frequency (TF–IDF), we observe that the proposed *in situ* accelerator can achieve speedups of 45.7× and 1.5× with respective energy improvements of 49.5× and 1.3× as compared with the CPU baseline.

II. Background and Motivation

This section provides the necessary background knowledge on data clustering algorithms, ROFs, and resistive memory technologies.

A. Clustering Algorithms

Data clustering is a computationally difficult (NP-hard) problem that refers to partitioning a set of objects into meaningful groups (called clusters) with no predefined labels [21]. The entities of a cluster are more similar to each other than to those in other clusters. K-means and its variants are the most prominent clustering algorithms that have been successfully used in numerous fields of science and engineering [22]. The basic k-means operations are shown in Algorithm 1, where k centroids are used to represent the clusters. A centroid is either a representative member of the cluster (e.g., median) or an additional data point computed based on the similarities among all of the cluster members (e.g., the arithmetic mean). The former has been proven to find better clusters than the latter due to its resistance against outliers [21], [22]. Prior to partitioning, the k centroids are randomly initialized; then, the algorithm repeats two steps (Lines 3 and 4 in Algorithm 1) until convergence is reached. First, the clusters are formed by assigning data points to their closest centroid; second, the centroids are recomputed for each cluster.[1]

1) Example Applications of Data Clustering: Numerous applications of k-means clustering can be found in the literature. We review two representative examples on GEA and text data mining.

[1]Based on the application needs, this process may be repeated either for a fixed number of iterations or until none of the centroids changes.

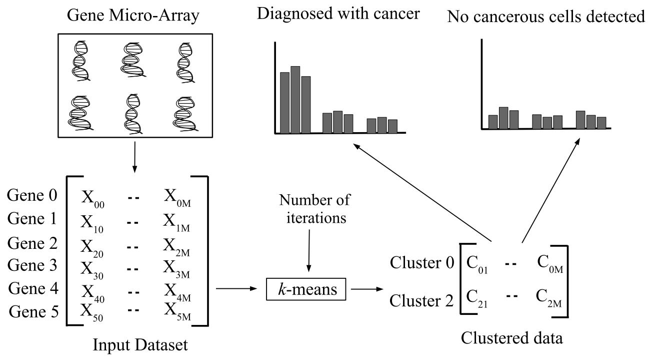


Fig. 1. Clustering gene samples to detect cancerous cells.

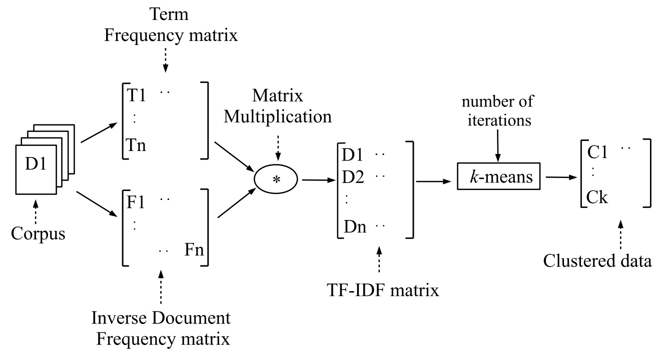


Fig. 2. TF–IDF text mining with k-means algorithm.

a) Gene expression analysis: Recently, clustering has seen wide use in medical research, such as cancer diagnosis and drug discovery. An accurate clustering algorithm often has a profound impact on the correctness of these applications. For example, Lu and Han [23] have shown that data clustering algorithms can be used for more accurate cancer classifications based on the abundance of gene expression data rather than the traditional morphological and clinical-based methods. A gene that forms the basic unit of heredity is defined as part of a deoxyribonucleic acid (DNA) transferred to an offspring by its parent. The process of transcribing a gene's DNA sequence into ribonucleic acid is called *gene expression* that changes during biological phenomena, such as cell development. In the case of diseases such as cancer, the genes of normal body cells undergo multiple mutations to evolve cancerous cells. As shown in Fig. 1, this anomaly is now possible to be detected through GEA that requires clustering of a large number of gene samples.

b) Document clustering: Clustering text documents is an important branch of text mining that refers to organizing paragraphs, sentences, and terms into meaningful clusters to improve information retrieval and document browsing [24]. Unlike the numerical data, text clustering requires preprocessing the documents to represent their features in the form of numerical vectors (see Fig. 2). These vectors are then used to group similar terms into the same clusters. A commonly used feature vector for text clustering is the TF, which represents the number of word occurrences in every document divided by the total number of words. Moreover, an IDF for every word is defined as the logarithmic ratio of the total number of documents to those that contain the word. These two metrics are then multiplied to compute a TF–IDF score ($w_{i,j}$) for the jth word of the ith document. Finally, the documents—represented as rows of the TF–IDF matrix—are partitioned into multiple groups with similar members using a clustering algorithm—e.g., k-means.

B. Data Clustering Using Rank Order Filters

ROFs are nonlinear digital components widely used in signal and image processing to filter out noise from input signals. In general, an ROF is characterized by: 1) the number of input signals (N) and 2) an index (i) that determines which input signal to appear in the output. The filter identifies the ith largest (or smallest) input signal to be sent to the output. In particular, a *median filter* can be realized if $i = (N/2)$, which is used to compute the centroids in the k-medians algorithm. Rank order median filters are memory intensive operations, for which numerous hardware and software optimizations have been proposed in the literature [25]–[27]. These proposals rely on two different approaches: 1) word-based search that sequentially examines all of the objects to find the median and 2) bit-serial process that computes the majority of selected bits from all of the objects in parallel. Both approaches suffer from excessive memory accesses when applied to large-scale data sets. This paper focuses on unlocking the significant potential for extremely parallelizing the bit-serial approach by *in situ* computation within memory arrays.

1) Bit-Serial Median Filter: In principle, the median of a list can be computed using a sorting algorithm, which is complex and inefficient. In 1981, the first bit-serial algorithm for median filters was proposed by Danielsson [28] that eliminates the need for sorting. Thereafter, numerous hardware and software implementations of bit-serial median filters have been proposed in the literature that relies on the *majority function* [20]. The majority function defines a mapping from N binary inputs to a single output, such that the output is **0** when $(N/2)$ or more inputs are **0**; otherwise, it is **1**. Assuming that the p_i values are the binary inputs, (1) can be used to compute the majority function

$$M(p_1, \ldots, p_N) = \left\lfloor \frac{1}{2} + \frac{\sum_{i=1}^{N} p_i - \frac{1}{2}}{N} \right\rfloor \qquad (1)$$

a) Computing the median value: Fig. 3 shows how to compute the median of five input numbers using the bit-serial algorithm. Every number is represented in a binary format within a single row. Starting from the most significant bit toward the least significant bit (LSB), the algorithm performs a vertical computation followed by a horizontal bit propagation, repeatedly.[2] During the vertical computation, the majority vote among all of the bits within a selected column is computed. The result of the majority function is used: 1) to determine a bit of the final result (median in Fig. 3) and 2) to identify the minority bits within the selected column. In the next step,

[2]Notice that the total number of iterations depends on the word size rather than the number of inputs.

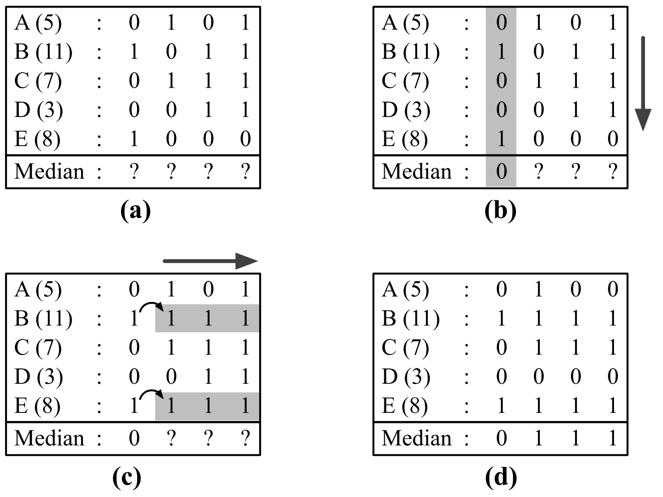


Fig. 3. Illustrative example of computing the median of five input numbers using the bit-serial algorithm. (a) Original data. (b) Compute majority. (c) Propagate minorities. (d) Final result.

the minority bits are used to replace all of the bits on their right-hand side.

b) Implementation challenges: Theoretically, the bit-serial median algorithm is amenable to massively parallel implementations: selected bits from all of the inputs can be processed in parallel. In practice, however, this potential parallelism is significantly constrained due to the required excessive memory accesses to the input numbers per every iteration. This paper designs a memory-centric accelerator that performs the majority function computation and bit propagation steps *in situ* memory arrays, thereby eliminating the unnecessary accesses to the inputs. As a result, the proposed platform will enable massively parallel execution of the bit-serial median algorithm.

2) Related Work on Median Filters: There have been several techniques coined in the past to implement 1-D/2-D median filters for data and speech processing, as well as image and video processing. Xilinx builds an FPGA-based 2-D ROF targeting image processing applications [29]. An example FPGA platform for computing 32-bit medians is found in prior work [16]. A 1-D median filter from nonrecursive sorting algorithms is proposed by Moshnyaga and Hashimoto [30]. Regrettably, the performance of these efforts has been significantly constrained due to the excessive memory accesses required during every iteration of the algorithm. The proposed accelerator will address this problem by computing the medians *in situ* novel memristive data arrays, which will eliminate all of the reads to the data points during median computation.

C. RRAM Array Structure

Resistive memories are nonvolatile, free of leakage power, and largely immune to radiation-induced transient faults [31], [32]. RRAM is one of the most promising memristive devices under commercial development that exhibits excellent scalability, high-speed switching, a high dynamic resistance range, and low power consumption [33]. Numerous

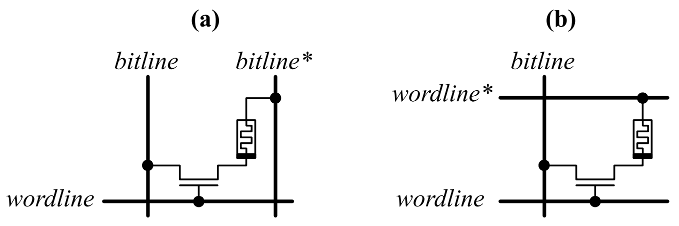


Fig. 4. Illustrative example of two common cell topologies for 1T-1R. (a) Double bitline cell. (b) Double wordline cell.

array topologies have been proposed in the literature that optimize RRAM for better reliability, density, and computational capabilities [34]–[36]. Fig. 4 shows two common example topologies for a one-transistor, one-memristor (1T-1R) memory cell comprising an access transistor and a resistive storage element. In both cases, the cell's content is read or written by applying an appropriate voltage across the memristive element. This can be accomplished by activating the access device using a *wordline* and applying the required read or write voltage across a *bitline* and the third terminal of the cell, which is a *bitline** or *wordline** for the dual bitline and dual wordline topologies, respectively. The proposed accelerator builds upon these cell topologies to design a new RRAM cell capable of storing data and performing the essential operations required for the bit-serial median filter (see Fig. 3).

D. Hardware Accelerators for In-Memory Processing

The proposed hardware accelerator builds upon existing mechanisms for near data processing (NDP) and *in situ* computation. Numerous hardware accelerators have been proposed on accelerating important applications in memory chips and arrays. PIM [37] aims at limiting round trip data movement by processing data directly on memory chips. Computational RAM [38] proposed a system that connects single instruction, multiple data (SIMD) pipelines with sense amplifiers. Parallel PIM [37] introduces a configurable chip to enhance the effect of PIM by operating as a conventional memory or for processing data as an SIMD processor. Active Pages [39] propose a microprocessor with adequate logic circuitry integrated near the dynamic random access memory (DRAM) arrays to reduce memory latency. FlexRAM [40] and intelligent RAM (IRAM) [41] are other techniques for NDP that have been evaluated on different technologies.

E. Hardware Accelerators for k-Means Clustering

The k-means and k-medians algorithms are widely used techniques for clustering data. As a result, researchers have proposed numerous hardware architectures for accelerating these techniques. Hussain *et al.* [42] propose an FPGA approach to accelerating k-means clustering on gene expression data sets obtained from gene microarray experiments. A recent work on clustering hyper spectral images proposes a reconfigurable hardware for accelerating the k-means algorithm [43]. This paper shows that using a hardware/software codesign operating at a moderate frequency can significantly reduce the clustering time.

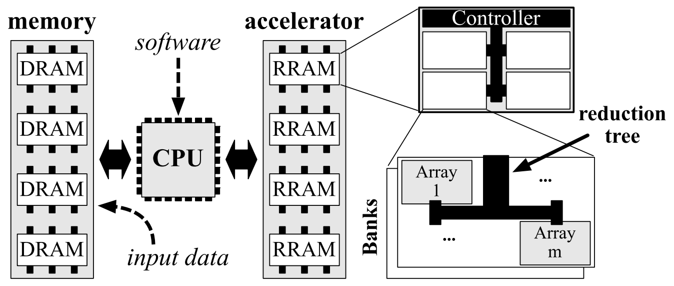


Fig. 5. Illustrative example of a multicore processor interfaced with the proposed memristive data clustering accelerator.

III. K-Medians Clustering Within RRAM Arrays

This section provides an overview of the proposed memristive accelerator and explains the design principles for realizing energy-efficient data clustering within memory arrays. The key idea is to exploit the computational capabilities of the memristive elements in RRAM arrays to perform the necessary computation of the bit-serial median filter algorithm in memory cells. As a result, the proposed architecture eliminates unnecessary latency, bandwidth, and energy overheads associated with streaming data out of the memory arrays during the clustering process. This novel capability will then unlock the unexploited massive parallelism in data clustering using bit-serial median filters.

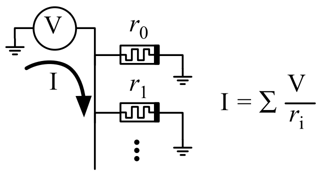
A. System Organization

As shown in Fig. 5, a memory module comprising multiple chips is designed to perform large-scale k-medians data clustering using the proposed accelerator. Each chip comprises a hierarchy of data arrays interconnected with a reconfigurable reduction network. The memory cells are capable of storing data bits and computing the basic operations required for bit-serial median filters. The interconnection network allows for retrieving or merging partial results from the data arrays. The accelerator module is connected to the processor via a standard double data rate memory interface [44]. This modular organization of the proposed accelerator allows the user to selectively integrate the proposed hardware in those computer systems that execute data clustering workloads.

Moreover, the proposed memory architecture supports two operational modes: the *storage* mode that allows the accelerator module to serve ordinary read and write requests and the *compute* mode, which is used for *in situ* data clustering. For every computation task, three steps are followed. First, the module is configured by software for solving a clustering problem. Next, the *in situ* computation will be triggered after transferring the data from the main memory to the accelerator chips. Finally, the processor will be notified by the accelerator to access the results.

B. Design Principles

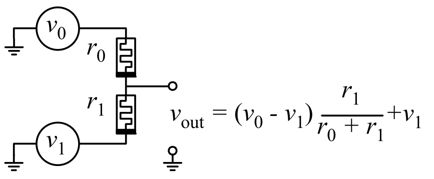
Three major operations are necessary to implement a bit-serial median filter within memory arrays: 1) computing the majority of bits within a selected column; 2) determining



(a)

v_1	v_0	r_1	r_0	v_{out}
V	0	R_{LO}	R_{HI}	~V
V	0	R_{HI}	R_{LO}	~0
0	V	R_{LO}	R_{HI}	~0
0	V	R_{HI}	R_{LO}	~V

(c)



(b)

v_0	r_0	v_{out}
0	0	1
0	1	0
1	0	0
1	1	1

(d)

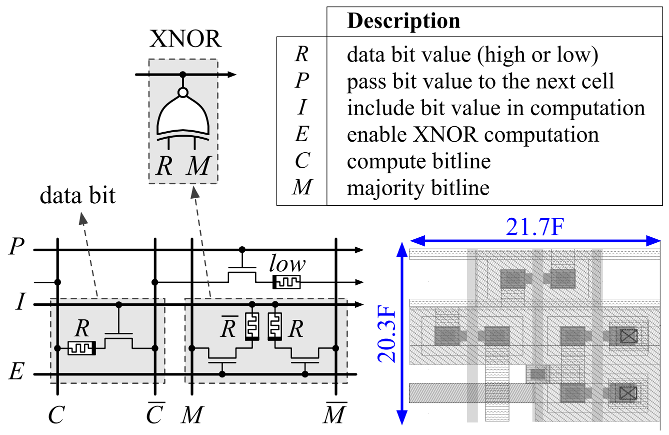
Fig. 6. Key ideas behind the proposed memristive data clustering accelerator. (a) Parallel resistors. (b) Serial resistors. (c) Signaling for serial resistors. (d) Logical XNOR.

which rows hold the minority bit; and 3) replacing the LSBs of those rows with the minority bit.[3] The key idea behind the proposed *in situ* data clustering with RRAM cells is shown in Fig. 6; the serial and parallel topologies of the resistive elements are used to build an XNOR circuit that computes the majority function and performs bit comparison.

1) Computing the Majority Vote: Fig. 6(a) shows how the majority function can be computed through parallel memristive cells connected to a single bitline. Assuming that each memory cell employs its high and low resistance states (i.e., R_{HI} and R_{LO}) to represent **1** and **0**, respectively. The number of **1**s determines the amount of current (I) flowing through the bitline. By measuring this current and comparing it with a threshold, one can determine whether the number of **1**s is greater than the half of bits or not, thereby computing the majority vote. This capability is leveraged to realize the vertical computation step in the bit-serial median filtering algorithm (see Fig. 3).

2) Performing In Situ Bit Comparison: Fig. 6(b) shows the proposed circuit for performing *in situ* bit comparison using two serially connected memristive elements that form a voltage divider network. Depending upon the input value (V) and the states of memristive elements (r_0 and r_1), the voltage divider produces an output (v_{out}) that varies between $(V R_{HI}/(R_{HI} + R_{LO})) \simeq V$ and $(V R_{LO}/(R_{HI} + R_{LO})) \simeq 0$. According to Fig. 6(c) and (d), this output value represents the results of a binary XOR/XNOR on v and r. The voltage divider circuit can greatly benefit from the high dynamic resistance range (R_{HI}/R_{LO}) provided by phase-change memory and RRAM technology, which is not easily available in other technologies, such as magnetoresistive RAM [45]. This paper examines the use of RRAM technology in leveraging this bit comparison concept to represent the memristive cell and the input data in their true and complement forms. This novel functionality is used in the bit-serial median filter for finding the minority bits, as well as in the k-medians algorithm for searching and selecting all the members of a

[3]The critical operations for these three steps are explained in Section II-B.



	Description
R	data bit value (high or low)
P	pass bit value to the next cell
I	include bit value in computation
E	enable XNOR computation
C	compute bitline
M	majority bitline

Fig. 7. Illustrative example of the proposed memory cell.

dynamically formed cluster prior to finding new centroids (see Section V-B).

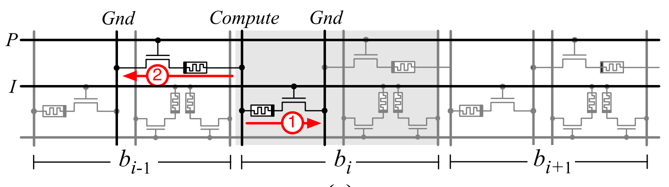
IV. Proposed Architectural Building Blocks

The proposed accelerator is designed based on three fundamental building blocks: a memory cell, a majority unit, and a network reduction unit. The building blocks are designed and optimized to achieve high memory density, low energy consumption, and massive parallel computational capabilities at the memory cells.

A. Memory Cell

Fig. 7 shows the circuit and an example physical layout for the proposed memory cell capable of: 1) serving ordinary reads and writes; 2) performing *in situ* XNOR between the cell and an external input; and 3) propagating the minority bit to its adjacent cell. The proposed a four-transistor, four-memristor cell comprises four transistors and four memristive elements that can be viewed as a combination of four conventional 1T-1R RRAM cells, as shown in Fig. 4. Three wordlines and four bitlines are provided to perform read, write, and compute operations on the cell. Every memristive element in the proposed cell can be read or written through a set of three bitlines and wordlines, which makes it possible to use the proposed cells as four individual 1T-1R memory cells to store data. For example, R in the *Data Bit* is accessed using $\{I, C, \overline{C}\}$; $\overline{R}$ in XNOR is accessed through $\{E, M, I\}$; similarly, $\{E, \overline{M}, I\}$ can be employed to access R in XNOR; and bitlines from adjacent cells are used to access low through $\{P, \overline{C}, C\}$.

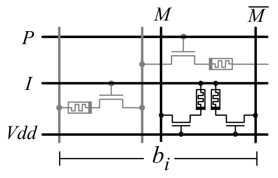
1) Computing the Median Within the Cell: Computing bit-serial median requires multiple steps, each of which involves activating the cells using bitlines and wordlines. Fig. 8(a) shows three adjacent memory cells in a row. On every iteration, only one cell of each memory row will be processed. Based on Fig. 8(b), P and I are initialized to determine if the cell should be included in computation. This is necessary to ensure that irrelevant data points are not included in computing the median value. To compute the majority vote of column b_i, bitlines $\overline{C}$ from columns b_i and b_{i-1} are connected to ground and bitline



(a)

P	I	Operation
0	0	unselected row
0	1	include *bi* in computation
1	0	include *bi*-1 in computation
1	1	unknown

(b)



(c)

Fig. 8. Illustrative example of computing with the proposed memristive cell. (a) Computing majority. (b) Computational modes. (c) Determining I and P.

C of the column b_i is connected to V_{dd}. As shown in Fig. 8(a), two paths are possible for flowing current between V_{dd} and ground. One path is through column b_i that determines the amount of current driven from the bitline based on the content of data bit (1). Another possible path includes the memristive element (low) of the previous column (b_{i-1}) to determine the amount of current pulled from the compute bitline in case of bit propagation (2). Notice that P and I are used to enable the current paths. As shown in Fig. 8(b), only one (or none) of the columns in every row contributes to the bitline current. First, none of the cells are selected if $PI = 00$; second, column b_i is selected to be included in the majority vote computation if $PI = 01$; and third, column b_{i-1} contributes to the majority vote computation if $PI = 10$. By measuring the total current driven through the compute bitline (C) and comparing it with a threshold, the majority is computed (see Section IV-B).

One difficulty in realizing the *in situ* bit-serial median computation is propagating the minority bit within each row that results in forming long chains of cells, thereby impacting the area, delay, and power dissipation. The proposed cell architecture avoids forming long chains by allowing only **1**s to be propagated from b_{i-1} to b_i. (Notice that **1** and **0** are represented with the low and high resistance states, respectively.) Because of the significant difference between the high and low resistance states in RRAM ($(R_{\text{HI}}/R_{\text{LO}}) \geq 10^5$ [33], [46]), the currents contributed to the bitline by those memristive cells in high resistance states can be omitted. This optimization is considered by dedicating a low memristive element in b_{i-1}, which is included in current summation only if P is high. After computing the majority vote in current column (b_i), P and I are recomputed prior to processing the next column (b_{i+1}). Fig. 8(c) shows how the new values for P and I are determined. The newly computed majority vote is applied to M and $\overline{M}$; E is connected to V_{dd} activating the access transistors; and the XNOR part of the cell generates an output ($\overline{M \bigoplus R}$) on the wordline I. On a **1**-to-**0** transition, first, I is set to **0** and will remain the same until the end of computation, and second, P is set to **1** only if M is equal to **1**.

2) Updating the Cell: Recall that the proposed cell stores the true and complement values of the data bit; therefore, updating the contents of every cell requires additional writes to the replicas. We employ a two-phase update mechanism that writes all **1**s in the first phase and then all of the **0**s. Notice that in a typical data clustering problem, the data set is written in the memory once and is read by the algorithm multiple times. As a result, the additional write operations do not impact the overall execution, significantly. Moreover, the performance and energy benefits of the proposed *in situ* computing significantly surpass such overheads.

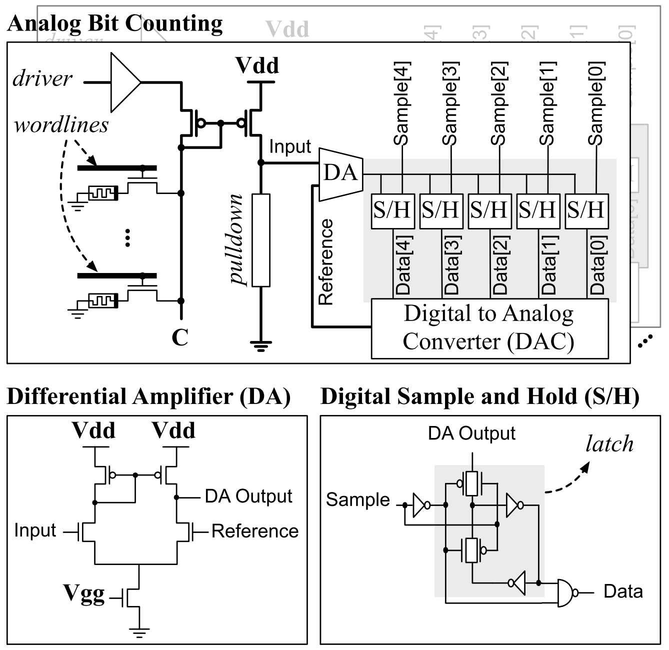


Fig. 9. Illustrative example of the proposed circuit for computing the majority vote.

B. Analog Bit Counter

In k-medians clustering, the cluster size determines the complexity of the majority vote unit. Fig. 9(a) shows an analog bit counter designed for the proposed accelerator to compute the majority vote per every column of data arrays. Notice that the bit counter unit is proposed to replace the conventional sense amplifier circuit in RRAM arrays. In addition to reading individual cells, the proposed analog bit counter is able to quantize the total amount of current pulled by the bitline into a multibit digital value—i.e., the number of **1**s within the column.

A successive approximation mechanism [47] is designed and employed to accomplish the analog bit counting for every column. To compute the majority vote of a column, V_{dd} is supplied to the compute bitline (C) using a line driver. A two-level amplification mechanism consisting a current mirror [48] and a current-mirror-based differential amplifier (DA) [49] is employed to quantize the total current of the compute bitline (C). Every time, the DA produces a high or low voltage based on the reference and input signals. We employ five novel digital sample and hold (S/H) units, each of which includes a latch and a NAND gate to sample the DA output and hold it in its data output. The S/H units produce a 5-bit input to a digital-to-analog converter [50] that generates the analog

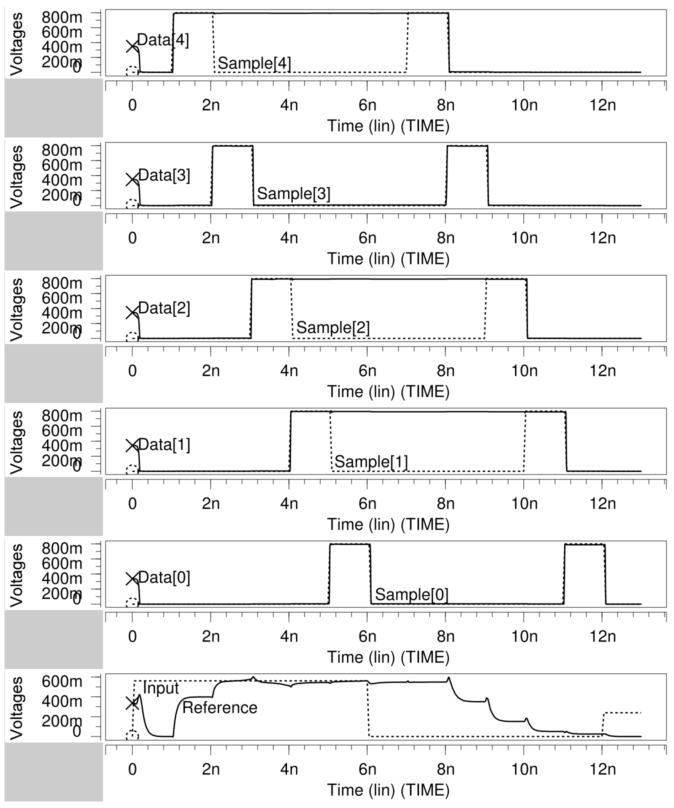


Fig. 10. Illustrative example of the waveforms generated by the proposed analog bit counting circuit for **22** and **0**.

reference signal fed to DA for comparison. Fig. 10 shows example waveforms generated through SPICE[4] simulations on different input signals that correspond to data values 22 and 0. Every input signal is quantized and sampled in five subsequent cycles in the S/H units. The output data bits from S/H units represent the quantized value (see time periods 6*n*–7*n* and 12*n*–13*n* in Fig. 10).

C. Reduction Unit

Theoretically, solving a large-scale data clustering problem requires computing the majority vote of a large number of data points stored in a single memory array, which becomes impractical due to significant sensing and reliability issues. Instead, data points are stored in multiple limited sized arrays, and only a fraction of the cells within each column is processed in every iteration—e.g., 32 cells of a 256×256 array. Notice that multiple such operations are performed in parallel data arrays to achieve significant performance improvements. As shown in Fig. 9, a hierarchical merging mechanism is proposed to compute the majority vote of a large number of data points stored in numerous data arrays across the accelerator chip.[5] An interconnection tree comprising reduction units is employed to merge the partial counts into a single value for computing the majority vote. The main purpose of the reduction tree is to merge the partial counts computed by the analog bit counters.

[4]Simulation program with integrated circuit emphasis.

[5]Section V provides the details of chip organization and data layout for the proposed accelerator.

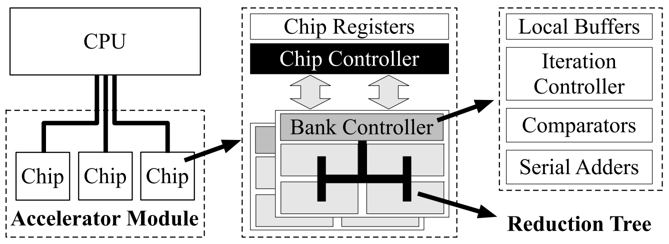


Fig. 11. Illustrative example of the proposed accelerator module.

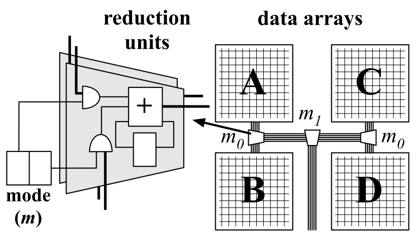
An energy-efficient reduction tree is designed to retrieve the partial data from the local analog bit counters and to compute the final bit counts. The reduction tree comprises a hierarchy of bit-serial adders to strike a balance between throughput and area efficiency.

V. System Architecture

The proposed hardware accelerator employs a hierarchical organization of data arrays, banks, and chips to form a memory module capable of solving clustering problems varying in size (see Fig. 11). Similar to prior work on the memristive Boltzmann machine [51], the accelerator module receives data and commands from CPU over an off-chip memory bus. Software initiates every clustering job by writing the configuration parameters into the chip registers—e.g., size and the number of iterations. Only, a small fraction of the memory address space (e.g., 1 KB) is dedicated to the chip registers, through which all of the configuration parameters, status flags, and data points are transferred between software and the hardware accelerator. Similar to Intel and advanced micro devices processors [52], [53], uncachable loads and stores are employed to access the chip registers in the same order as requested by software. Regardless of the data size, one chip register is dedicated for transferring data to the accelerator. The chip controller receives the data to be clustered word-by-word and distributes them into the data arrays across multiple accelerator banks. Similarly, a data output port is employed to collect the results of data clustering from the accelerator.

A. Bank Organization

As shown in Fig. 11, every accelerator chip comprises multiple banks that perform data clustering, independently. A reconfigurable reduction tree is used inside every bank to interconnect the data arrays and the chip controller. The proposed tree is capable of selectively merging the partial counts from the data arrays into a single count value. For example, Fig. 12 shows the proposed reduction unit employed to realize nine possible ways of reading data from the children arrays A, B, C, and D. All of the reduction units at every node of the tree are connected to a shared four-bit *mode* register. The nodes are programmed to appropriate operational modes prior to solving a clustering problem. The nodes at the same tree level are programmed to the same mode value—e.g., m_0 in Fig. 12. It is now possible to read the individual arrays that are used for serving ordinary read requests or to read the sum of values provided by every two or four adjacent



m_1 m_0	Operation
01 01	Read A
01 10	Read B
01 11	Read A+B
10 01	Read C
10 10	Read D
10 11	Read C+D
11 01	Read A+C
11 10	Read B+D
11 11	Read A+B+C+D

Fig. 12. Proposed configurable reduction tree for banks.

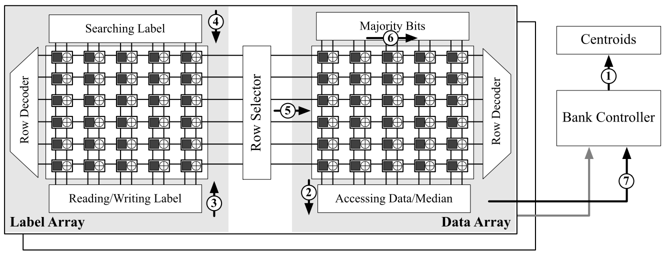


Fig. 13. Illustrative example of the proposed array comprising the cells and peripheral circuits.

arrays, which is used for computing the count value. This flexibility is essential to achieve significant energy efficiency in computing those problems that partially occupy the accelerator banks. In the proposed data clustering platform, software is responsible for computing the mode bits for a given problem size. The mode bits are then streamed into the accelerator during the chip initialization phase.

B. Array Organization

As shown in Fig. 11, every bank includes a controller consisting of buffers for maintaining local data (e.g., centroids), serial adders, comparators, and logic for controlling iterative tasks, such as partitioning and recomputing the centroids. Moreover, the bank controller makes it possible to read, write, and compute a set of selected data arrays efficiently. For energy-efficient k-medians clustering, two subarrays are used to build an array that maintains the data points and the corresponding labels (see Fig. 13). On every clustering iteration, seven major steps are followed by the bank controller to cluster the data points.

1) Initializing Centroids: Every k-medians clustering task begins with randomly initializing the centroids, which are maintained by the bank controller in local buffers (1). The index of each centroid in this table is used as a label for the corresponding cluster.

2) Forming New Clusters: The bank controller forms new partitions by reading the data points from the data subarrays and comparing them with the centroids (2). The index of the closest centroid to every data point is used as the new cluster label for that data and will be written to the label array (3). This is accomplished through a set of serial comparators at the bank controller. As the data points are read out, the serial comparator determines the index of the closest centroid to the data.

3) Computing Medians: The centroid of every cluster must be recomputed by applying the bit-serial median algorithm

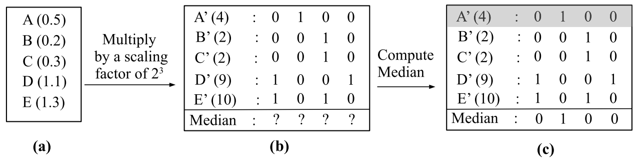


Fig. 14. Proposed mechanism for clustering real numbers. (a) Floating point data set. (b) Fixed-point numbers. (c) Median of fixed-point numbers.

to all the elements of every cluster. This requires the bank controller to keep track of the cluster members at all time. The label array uses the same structure as the data array to carry out the required book keeping for all of the data points. At the beginning of every median computation, the label arrays are searched for matching entries using the cluster labels one after another (4). The outcome of every search operation is the matching lines in the label subarray connected to a *row selector* unit to determine the I and P values for the data array (5). Next, the median's bits are computed by iteratively performing the vertical majority vote computation followed by the horizontal minority propagation (6). As being serially computed, the median's bits are streamed to the bank controller for updating the centroids (7). This process will end after a certain number of iterations defined by software or when all of the newly computed centroids are the same as the old ones (i.e., convergence is reached).

C. Data Representation

Due to the limitations of the bit-serial median algorithm for negative or real numbers, the proposed accelerator requires the data points to be represented in a fixed-point positive format. The necessary data conversion and preprocessing for clustering real numbers and negative values are performed by software prior to loading the data points into the accelerator chips.

1) Clustering Real Numbers: Our experiments indicates that the energy and delay overheads of this required preprocessing are negligible compared with the energy and delay of transferring data to/from the accelerator and performing the actual clustering computations. Moreover, we observed that a 64-bit fixed-point format for the evaluated applications and data sets achieves virtually the same results obtained with a double precision IEEE floating point format. Nevertheless, for sensitive applications, the proposed accelerator is flexible enough to compute the medians of wider bit representations by increasing the number of vertical majority vote computation and applying minimal changes to the control logic. Fig. 14 shows an example of floating point to fixed-point conversion. The input floating point data are scaled by a factor of 2^3 and then are converted to fixed-point data. We now apply an ROF to compute the median of the fixed-point data.

2) Handling Negative Numbers: The median computation by an ROF algorithm assumes that the input data are positive integers. However, for real-time applications, the input data set need not be positive integers. The proposed accelerator addresses this concern by representing the input data set in biased notation, i.e., a bias value 2^n is added to all of the negative and positive elements. Fig. 15 shows an example of computing the median of four-bit integer data. A bias of 2^3 is

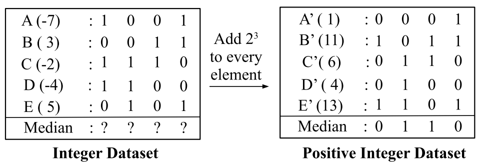


Fig. 15. Illustrative example of handling negative numbers in the proposed clustering framework.

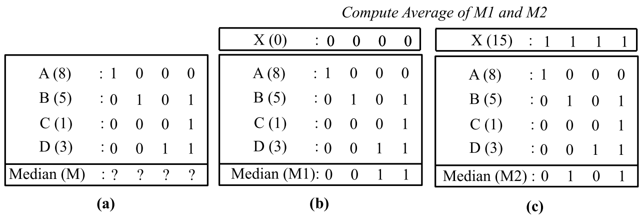


Fig. 16. Illustrative example of even number of data points. (a) Original data. (b) Append 0 and compute Median. (c) Append 15 ($0 \times f$) and compute Median.

added to each of the data points to make them positive, and the median is then computed by applying the ROF algorithm.

D. Handling Even Number of Data Points

The ROF algorithm can compute the median of an odd number of data points only. The proposed accelerator, however, is capable of computing the median of both even and odd numbers of input data. The members of a given cluster may be spread across different arrays and banks of the accelerator. The number of cluster elements in each cluster may change per iteration due to cluster reformations. In order to find the median for a given cluster, we first send the cluster label to the label array. Similar analog bit counting is used to find the number of matches in all label arrays. As shown in Fig. 16, if the cluster contains an even number of elements, the median of the cluster is computed in two steps. First, we append a new data point with value **0** to the cluster elements so that the number of data points becomes odd; then, we find the median $M1$ using the median ROF algorithm. In the second step, we append an all **1**s data point to the cluster elements and compute the median $M2$. The average of $M1$ and $M2$ gives the true median M for the cluster.

VI. Experimental Setup

We evaluate the system performance and energy consumption of the proposed hardware accelerator using a modified version of the enhanced superscalar simulator (ESESC) [54]. A library with real data sets [55]–[57] and two applications pertaining to k-means clustering is used to assess the hardware accelerator against a baseline CPU and a PIM accelerator. We employ the cache access and cycle time models [58] with the multicore power, area, and timing tool [59] to calculate the overall system energy for every configuration.

A. Methodology

K-means computation in CPU involves excessive data movement between the core and the main memory. As the input size and the number of clusters increase, system performance deteriorates further. Moreover, in the case of k-medians clustering, significant computations are required to calculate the median. The baseline CPU configuration implements the k-means algorithm for floating point real data sets. As mentioned before, the quality of k-medians is higher than k-means, because the latter is susceptible to outliers [21], [22]. However, since k-means does not need data sorting, it is much faster than the k-medians. To better evaluate the performance and energy of the proposed accelerator as compared with CPU, we consider a k-means implementation that provides the best performance and energy on CPUs. [Notice that the proposed memristive hardware accelerator achieves higher clustering quality (using k-medians) having performance and energy numbers significantly better than the k-means.]

Present day, energy-efficient big data processing techniques use parallel compute substrates, such as FPGA, GPU, and many-core systems, to better utilize the memory bandwidth and computational resources. For example, advanced GPU architectures [60] employ in-package DRAM to achieve TBps memory bandwidths for data intensive applications. To provide a comparison against such compute systems, we develop a PIM specifically designed and optimized for k-means data clustering. By virtue of being specifically designed, the PIM accelerator eliminates some of the limitations in the general purpose CPU, GPU, and FPGA baselines. PIM places the functional units next to the data arrays on memory dice to alleviate the significant cost of data movement between the processor and the memory in every iteration of the algorithm. Despite a higher bandwidth provided by in-package DRAM, notice that the bandwidth of such system is still limited by the through silicon vias and microbumps used for die-to-die communication, whereas a PIM architecture can directly access data in the memory arrays without such limitations. To build a strong PIM baseline, we consider many dense memristive arrays connected to CMOS functional units that implement the k-means algorithm.

Notice that k-medians requires sorting of data before median computation and majority vote if calculated by the ROF method. These prerequisites for median computation are both time-consuming and demand a large hardware setup for a PIM architecture. In contrast, the *in situ* accelerator addresses these issues and achieves significantly higher performance and energy efficiency due to: 1) *in situ* computations that eliminates unnecessary data movement at the data arrays and simplifies the additional logic on the memory dice and 2) median computation that enables massively parallel processing at the memory cells for finding centroids through rank order median filtering. For fair comparison, we explore the design space of both PIM and the proposed accelerators to find instances that achieve the best energy efficiency within similar area consumption.

B. Architecture

We model a single core system configuration on ESESC [54] that runs at a clock frequency of 3.2 GHz and employs a two-level cache hierarchy with a four-way

TABLE I
BASELINE CPU CONFIGURATION

	Baseline Configuration
Core	a 4-issue OoO core,128 ROB entries, 3.2 GHz
IL1/DL1 cache	32KB, 4-way, LRU,64B block,
L2 cache	2MB, 8-way, LRU, 64B block, MESI protocol
Memory	2 DDR4-3200, FR-FCFS

TABLE II
APPLICATION AND DATA SETS

	Label	Description	Data Source	Number of Points (M)
Application	GEA	Gene- Expression Analysis	Cluster 3.0	549 Genes, 191 experiments 1098 Genes, 191 experiments 2196 Genes, 191 experiments 4392 Genes, 191 experiments
	TF-IDF	Term Frequency Inverse Document Frequency	BBCSport Website	150 Documents, 737 words 300 Documents, 737 words 600 Documents, 737 words 1100 Documents, 737 words
Library	ASA136	Applied Statistics Algorithm 136	Breast Cancer Samples	85490 datapoints 170980 datapoints 338640 datapoints 676450 datapoints
			Indoor Localization Records	260000 datapoints 521040 datapoints 1066000 datapoints 2083120 datapoints
			US Census Data	544000 datapoints 1088000 datapoints 2108000 datapoints 4080000 datapoints

32-KB L1 cache and an eight-way 2-MB last-level cache. The single core system interfaces to two DDR4-3200 DRAM channels, as shown in Table I. For the CPU baseline, both DDR4 channels are plugged with DRAM and serve as a main memory, whereas in the PIM and the proposed *in situ* baselines, one of the channels is employed for communicating with the accelerator. For every clustering problem, the data points and the configuration parameters are first transferred to the accelerator. For transferring data points, a direct memory access mechanism is implemented in ESESC that is configured by the software and simulates the process in cycle-accurate fashion.

C. Applications

We evaluate the accelerator for a k-means library with three real data sets belonging to different domains. First, data set profiles breast cancer samples [55] containing nearly 12 000 proteins per sample. Second, an indoor localization data set [56] with nearly 20 000 training records validated with more than 1000 points, and the last data set contains nearly 1% sample of the total U.S. census from 1990 [57]. We further assess the proposed hardware on two k-means-specific applications—GEA and text mining using TF–IDF. Table II summarizes the data sets used for the library and applications.

The data set used for GEA is a 2-D input file, wherein the rows show various genes, while every column represents experimental results of these genes over time. For evaluating TF–IDF-based document clustering, we use BBC data sets consisting of 737 documents from the BBC Sport website corresponding to sports news articles in five topical areas (athletics, cricket, football, rugby, and tennis) from 2004 to 2005 [61]. The input data set for this algorithm includes a corpus and a collection of words that exists across all the documents as mentioned in Table II. The product of the number of word occurrences in every document divided by the total number of words (TF) and the logarithmic ratio of the total number of documents to the number of documents in which the word occurs (IDF) forms a 2-D TF–IDF matrix that represents the frequency of occurrence of each word in every document present in the corpus. Finally, the words are clustered into multiple groups based on their frequency of occurrence in the corpus.

We study all of the applications and input data sets to determine appropriate ways of representing real and negative numbers. The floating point to fixed-point conversion may induce a rounding error that impacts the final cluster assignments. The choice of scaling factor depends on the range of the input data set. We have evaluated the k-means clustering for various scaling factors ranging from 2^{18} to 2^{24} and observed no change in the final cluster assignments. As discussed in Section V-C, a fixed bias value is added to every data point in order to convert the data set to positive integers. We have evaluated the k-means clustering for various bias values ranging from 2^{24} to 2^{32} and observed no change in the final cluster assignments.

D. Circuits

SPICE predictive technology models [62] of the CMOS transistors at a 22-nm technology node are used to evaluate the proposed RRAM arrays. Area, delay, and energy of the data arrays are evaluated with memristive parameters $R_{\text{LO}} =$ 315 K and $R_{\text{HI}} = 1.1$G based on prior work [46] using NVSim [63]. The RRAM element is simulated using the Verilog-A model provided by prior work on the threshold adaptive memristor model [64] that is tuned for a switching time of 50 ns. The parasitic resistance and capacitance of the wordlines and bitlines are modeled based on the interconnect projections from international technology roadmap for semiconductor [45]. Area, timing, dynamic energy, and leakage power are computed by performing circuit simulations and hardware synthesis on the controller logic at the 45-nm technology node [65]. The results are scaled to 16 nm using the scaling parameters provided by prior work [66]. A charge pump circuit [67] is used to provide higher voltage, as RRAM cells require a write voltage more than supply V_{dd}.

VII. EVALUATION

This section presents potential performance improvements and system energy savings that are attainable for the proposed accelerator compared with the CPU and PIM baselines.

A. Energy

We evaluate the energy potentials of the proposed *in situ* accelerator by comparing its total system energy with that

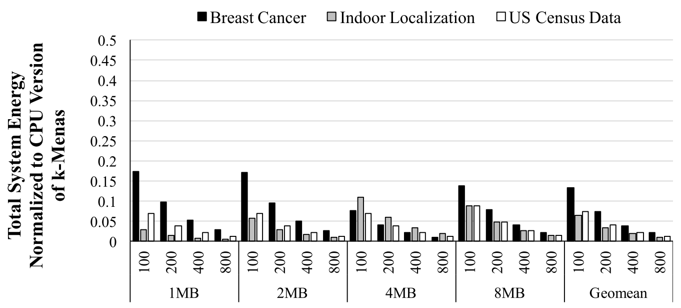


Fig. 17. ASA Library—system energy consumption normalized to the CPU baseline.

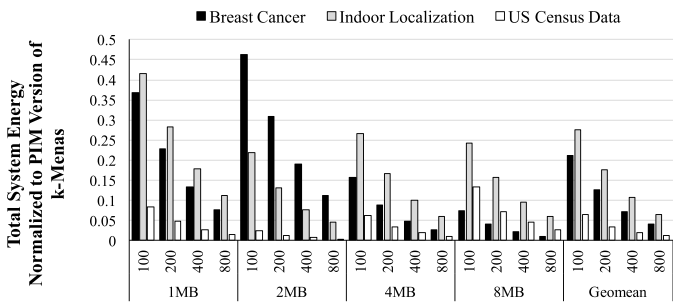


Fig. 18. ASA Library—system energy consumption normalized to the PIM baseline.

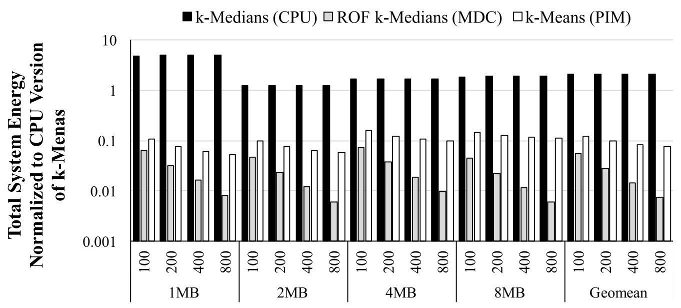


Fig. 19. GEA—system energy consumption normalized to the CPU baseline.

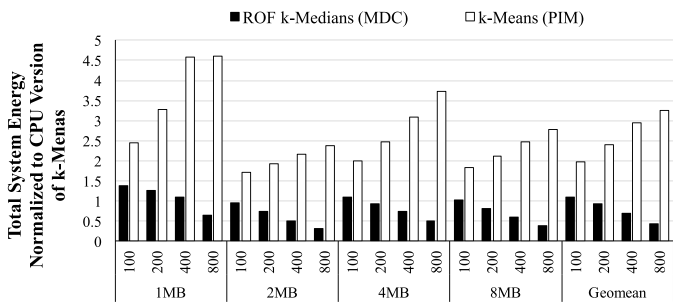


Fig. 20. TF–IDF—system energy consumption normalized to the CPU baseline.

of two baseline implementations of *k*-means on a general purpose CPU and a PIM-like hardware accelerator. Fig. 17 shows the overall system energy consumption normalized to the CPU baseline when accelerating the applied statistics algorithm (ASA) library for clustering various data sets of sizes 1, 2, 4, and 8 MB with 100, 200, 400, and 800 iterations to form four clusters. The data sets are selected from breast cancer, indoor localization, and U.S. census data explained in Section VI. The results indicate that the proposed *in situ* accelerator can significantly reduce the average system energy over the CPU baseline: 18.7×, 38.4×, and 36.2× for the breast cancer, indoor localization, and U.S. census data sets, respectively.

Fig. 18 provides a similar energy comparison between the proposed *in situ* accelerator and a PIM-like baseline. We observe a significant energy reduction achieved by the proposed *in situ* accelerator due to eliminating data accesses to the memory arrays for fetching data prior to data clustering. The results indicate averages of 18.7×, 7.3×, and 32.1× energy savings for the proposed *in situ* accelerator as compared with the PIM-like baseline. Also, we observe more significant energy improvements that are achieved as the number of iterations of the clustering algorithm increases.

We further evaluate the system energy improvements by accelerating two different applications of data clustering. Fig. 19 shows the overall system energy normalized to the CPU baseline when accelerating the GEA application. To better evaluate the systems, we vary the size of data sets and the number of iterations to form four clusters on the gene data. The results indicate that the PIM-like baseline achieves an average of 10.7× improvement over the CPU *k*-means baseline, whereas the proposed accelerator improves the system energy by more than 49×. For further evaluations, we include a *k*-medians version of GEA implemented on CPU, which results in consuming 2.1× system energy compared with the *k*-means CPU baseline.

Most of energy and execution time of the GEA application are consumed by clustering the data point. Unlike GEA, document clustering with TF–IDF comprises multiple significant components, including text-to-number conversion and data classification. Therefore, TF–IDF represents a group of data clustering applications with moderate potentials for hardware acceleration. Fig. 20 shows the system energy of the *in situ* accelerator and PIM normalized to that of the *k*-means implementation on CPU.[6] Similar to previous experiments, the data points are processed to form four clusters. The results indicate that PIM consumes more energy than the CPU, while the *in situ* accelerator improves energy an average of 1.3×. As shown in Fig. 20, the energy improvements by the *in situ* accelerator increases to 2.2×, as the number of clustering iterations varies from 100 to 800.

B. Performance

Fig. 21 shows performance improvements gained by the *in situ* accelerator over the CPU baseline for accelerating the ASA library for clustering various data sets (1, 2, 4, and 8 MB) with 100, 200, 400, and 800 iterations to form four clusters. The results indicate average speedups of 11×, 8.3×, and 41.2× over the CPU baseline for the breast cancer, indoor localization, and U.S. census data sets, respectively. Similarly, Fig. 22 shows significant speedup over the

[6]We notice that this increase is mainly due to the significant static energy consumed by the additional functional units used in PIM. The PIM-like accelerator is optimized for performance.

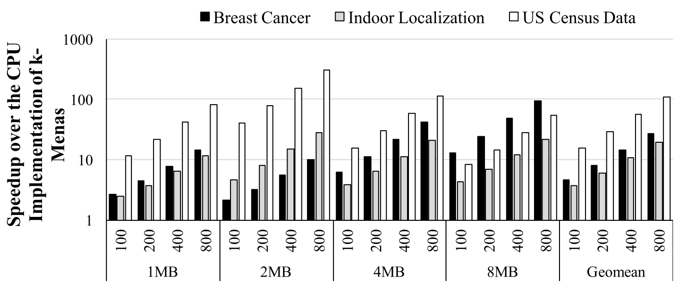


Fig. 21. ASA Library—performance normalized to the CPU baseline.

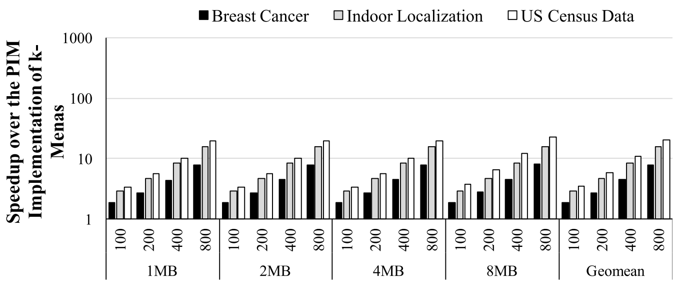


Fig. 22. ASA Library—performance normalized to the PIM baseline.

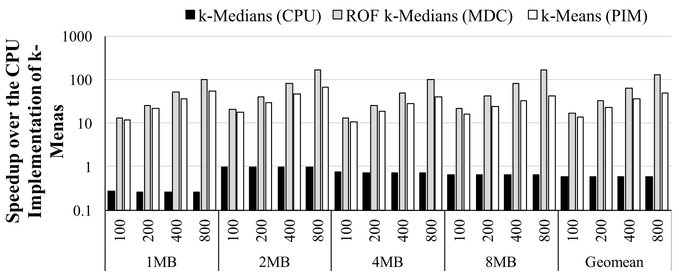


Fig. 23. GEA—performance normalized to the CPU baseline.

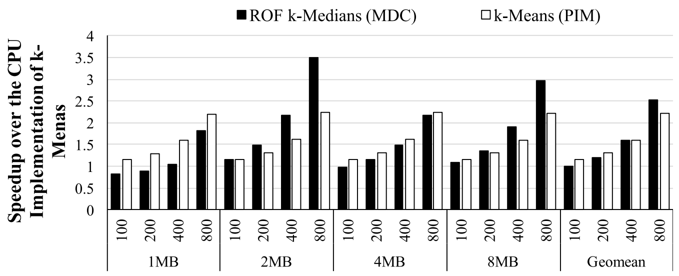


Fig. 24. TF–IDF—performance normalized to the CPU baseline.

PIM hardware: averages of 3.7×, 6.5×, and 8.1× for the breast cancer, indoor localization, and U.S. census data sets, respectively.

As shown in Fig. 23, the proposed *in situ* accelerator and PIM baseline achieve respective average speedups of 45.7× and 27.2× over the baseline CPU implementation. However, the CPU version of k-medians increases the execution time by 70% on average. For the TF–IDF document clustering, we observe an average drop in performance by 10% for small data sets. As the input size increases, the performance rise of 1.5× is observed (see Fig. 24). Overall, we observe more performance improvements, as the number of iterations increases.

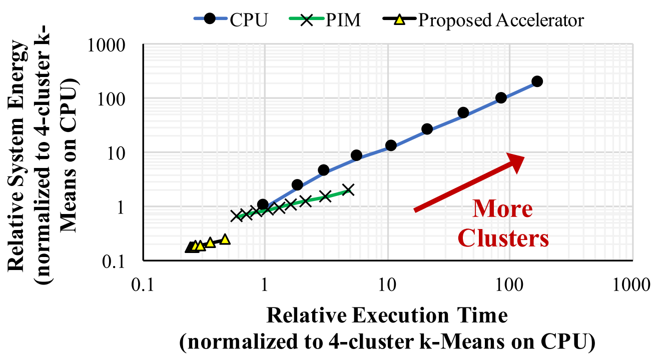


Fig. 25. Execution time and total system energy with increase in the number of clusters.

C. Discussion

1) Finite Switching Endurance: The switching endurance exhibited by RRAM cells varies between 10^6 and 10^{12} writes [33], [46], [68]. The number of possible writes affects the lifetime of RRAM cells. Therefore, the number of writes to RRAM cells is monitored to estimate the lifetime of the proposed hardware accelerator. It is important to know that the input data set is written only once into the accelerator and the memory cells maintain content during clustering iterations. To find the minimum lifetime of the accelerator dual in-line memory module, we conservatively consider the hardware being used constantly to solve clustering problems that require the full capacity of the accelerator.[7] We assume that every new problem requires all of the memory cells to switch when loading the data points; therefore, we find the time between the switching of the memory cells in this setup, which is $T_s = 2.258\Lambda$ s. Λ is the product of the number of clusters and the number of iterations that are defined by the application. Notice that T_s increases as the number of clusters and iterations increase. Assuming 10^8 for write cycles and 1 for Λ, the minimum lifetime of the proposed hardware accelerator is estimated to be seven years while constantly solving clustering problems.

2) Increase in the Number of Clusters: Fig. 25 shows the impact of increase in the number of clusters on the overall system energy and execution time of the CPU, the PIM, and the memristive accelerator. This sensitivity analysis is performed on the ASA 136 library with 400 iterations while increasing the number of clusters from 4 to 1024 by a step of 2^n. Each design point represents the relative execution time and system energy averaged on three runs of the library for 8-MB data from breast cancer, indoor localization, and U.S. census data sets. The results indicate that the energy and execution time of data clustering increase as the number of clusters grows; however, such increase is much more significant for the PIM and CPU baselines. Overall, the proposed *in situ* accelerator achieves 22–290k× and 8–81× better energy-delay-products compared with the CPU and PIM baselines, respectively.

[7]In the case of clustering small problems, memory allocation techniques may be employed to distribute write across memory cells to alleviate the endurance problem.

VIII. Conclusion

K-means and *k*-medians clustering are widely used techniques for data clustering in scientific research and engineering disciplines. Most of the available solutions suffer in performance and energy due to excessive data movement involved throughout the clustering process. The proposed RRAM-based *in situ* data clustering accelerator successfully addresses these concerns by implementing bit-serial ROF algorithm for median calculation and performing *in situ* computations within novel RRAM memory cells, thereby eliminating unnecessary data movement between the core and the main memory. Based on our simulation results, the proposed accelerator achieves significantly better energy and performance improvements as compared with CPU and PIM-like accelerators. In conclusion, the proposed hardware accelerator significantly improves the performance and energy for clustering applications involving processing of very large data sets.

Acknowledgment

The authors would like to thank anonymous reviewers for useful feedback.

References

[1] V. Gligorijević, N. Malod-Dognin, and N. Pržulj, "Integrative methods for analyzing big data in precision medicine," *Proteomics*, vol. 16, no. 5, pp. 741–758, 2016.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," *HotCloud*, vol. 10, no. 10, pp. 1–7, 2010.

[3] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[4] B. Huval *et al.* (Apr. 2015). "An empirical evaluation of deep learning on highway driving." [Online]. Available: https://arxiv.org/abs/1504.01716

[5] A. Seret, T. Verbraken, and B. Baesens, "A new knowledge-based constrained clustering approach: Theory and application in direct marketing," *Appl. Soft Comput.*, vol. 24, pp. 316–327, Nov. 2014.

[6] P. Liu, "Special issue 'Intuitionistic fuzzy theory and its application in economy, technology and management,'" *Technol. Econ. Develop. Econ.*, vol. 22, no. 3, pp. 327–335, 2016.

[7] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, Oakland, CA, USA, 1967, vol. 1. no. 14, pp. 281–297.

[8] R. T. Ionescu and M. Popescu, "Learning based on similarity," in *Knowledge Transfer Between Computer Vision and Text Mining* (Advances in Computer Vision and Pattern Recognition). Cham, Switzerland: Springer, 2016.

[9] H. T. Nguyen *et al.*, "Prediction of chemotherapeutic response in bladder cancer using *k*-means clustering of dynamic contrast-enhanced (DCE)-MRI pharmacokinetic parameters," *J. Magn. Reson. Imag.*, vol. 41, no. 5, pp. 1374–1382, 2015.

[10] S. Krishnan *et al.*, "Unsupervised surgical task segmentation with milestone learning," in *Proc. Int. Symp. Robot. Res. (ISRR)*, 2015, pp. 1–16.

[11] L. Kaufman and P. J. Rousseeuw, *Clustering by Means of Medoids*. Amsterdam, The Netherlands: North Holland, 1987.

[12] C. Elkan, "Using the triangle inequality to accelerate *k*-means," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, 2003, pp. 147–153.

[13] D. Sculley, "Web-scale *k*-means clustering," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 1177–1178.

[14] A. Vedaldi and B. Fulkerson, "Vlfeat: An open and portable library of computer vision algorithms," in *Proc. 18th ACM Int. Conf. Multimedia*, 2010, pp. 1469–1472.

[15] R. Mueller, J. Teubner, and G. Alonso, "Data processing on FPGAs," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 910–921, 2009.

[16] P. Wendt, E. Coyle, and N. Gallagher, "Stack filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 4, pp. 898–911, Aug. 1986.

[17] G. E. Blelloch, A. Gupta, and K. Tangwongsan, "Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design," in *Proc. 24th Annu. ACM Symp. Parallelism Algorithms Archit.*, 2012, pp. 205–213.

[18] M. Zechner and M. Granitzer, "Accelerating *k*-means on the graphics processor via CUDA," in *Proc. 1st Int. Conf. Intensive Appl. Ser. (INTENSIVE)*, Apr. 2009, pp. 7–15.

[19] D. Liu *et al.*, "PuDianNao: A polyvalent machine learning accelerator," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 369–381, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2775054.2694358

[20] I. Hatirnaz, F. K. Gurkaynak, and Y. Leblebici, "Realization of a programmable rank-order filter architecture using capacitive threshold logic gates," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 1. May 1999, pp. 435–438.

[21] A. Vattani. (2009). *The Hardness of k-Means Clustering in the Plane*. [Online]. Available: http://cseweb.ucsd.edu/avattani/papers/kmeans_hardness.pdf

[22] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, 1988.

[23] Y. Lu and J. Han, "Cancer classification using gene expression data," *Inf. Syst.*, vol. 28, no. 4, pp. 243–268, 2003.

[24] P. G. Anick and S. Vaithyanathan, "Exploiting clustering and phrases for context-based information retrieval," *ACM SIGIR Forum*, vol. 31, pp. 314–323, Jul. 1997.

[25] P. V. Tymoshchuk and S. V. Shatny, "Hardware implementation design of analog neural rank-order filter," in *Proc. 11th Int. Conf. Perspective Technol. Methods MEMS Design (MEMSTECH)*, Sep. 2015, pp. 88–91.

[26] A. Gasteratos, I. Andreadis, and P. Tsalides, "A new hardware structure for implementation of soft morphological filters," in *Proc. Int. Conf. Comput. Anal. Images Patterns*, 1997, pp. 488–494.

[27] T. Yamamoto and V. G. Moshnyaga, "A new bit-serial architecture of rank-order filter," in *Proc. 52nd IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2009, pp. 511–514.

[28] P.-E. Danielsson, "Getting the median faster," *Comput. Graph. Image Process.*, vol. 17, no. 1, pp. 71–78, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0146664X8180010X

[29] G. Szedo, "Two-dimensional rank order filter," XililL, Appl. Note XAPP953, 2006, pp. 15–26, accessed: Nov. 22, 2017. [Online]. Available: https://pdfs.semanticscholar.org/790d/aac5bbe4caf8cb7c2ed8394cbb69e0753a00.pdf

[30] V. G. Moshnyaga and K. Hashimoto, "An efficient implementation of 1-D median filter," in *Proc. 52nd IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2009, pp. 451–454.

[31] B.-Y. Tsui, K.-C. Chang, B.-Y. Shew, H.-Y. Lee, and M.-J. Tsai, "Investigation of radiation hardness of HfO_2 resistive random access memory," in *Proc. Int. Symp. VLSI Technol., Syst. Appl. (VLSI-TSA)*, Apr. 2014, pp. 1–2.

[32] *Resistive Memory Devices for Radiation Resistant Non-Volatile Memory*. Accessed: Nov. 22, 2017. [Online]. Available: https://www.nasa.gov/content/resistive-memory-devices-for-radiation-resistant-non-volatile-memory/

[33] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proc. IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec. 2010.

[34] M. Zangeneh and A. Joshi, "Design and optimization of non-volatile multibit 1T1R resistive RAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 8, pp. 1815–1828, Aug. 2014.

[35] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnol.*, vol. 3, no. 7, pp. 429–433, 2008.

[36] P.-F. Chiu and B. Nikolić, "A differential 2R crosspoint RRAM array with zero standby current," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 5, pp. 461–465, May 2015.

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

[38] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational RAM: Implementing processors in memory," *IEEE Design Test Comput.*, vol. 16, no. 1, pp. 32–41, Jan. 1999.

[39] M. Oskin, F. T. Chong, and T. Sherwood, "Active pages: A computation model for intelligent memory," in *Proc. 25th Annu. Int. Symp. Comput. Archit.*, Jun. 1998, pp. 192–203.

[40] Y. Kang *et al.*, "Flexram: Toward an advanced intelligent memory system," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Sep. 2012, pp. 5–14.

[41] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, Mar. 1997. [Online]. Available: http://dx.doi.org/10.1109/40.592312

[42] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "FPGA implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *Proc. NASA/ESA Conf. Adaptive Hardw. Syst. (AHS)*, 2011, pp. 248–255.

[43] A. G. S. Filho *et al.*, "Hyperspectral images clustering on reconfigurable hardware using the k-means algorithm," in *Proc. 16th Symp. Integr. Circuits Syst. Design (SBCCI)*, Sep. 2003, pp. 99–104.

[44] Micron Technology, Inc. (2009). *8Gb DDR3 SDRAM*. [Online]. Available: http://www.micron.com//get-document/documentId=416

[45] ITRS. *International Technology Roadmap for Semiconductors: 2013 Edition*. Accessed: Nov. 22, 2017. [Online]. Available: http://www.itrs.net/Links/2013ITRS/Home2013.html

[46] C. H. Cheng, A. Chin, and F. S. Yeh, "Novel ultra-low power RRAM with good endurance and retention," in *Proc. Symp. VLSI Technol.*, Jun. 2010, pp. 85–86.

[47] B. Razavi, *Principles of Data Conversion System Design*. Hoboken, NJ, USA: Wiley, 1995.

[48] P. J. Crawley and G. W. Roberts, "High-swing MOS current mirror with arbitrarily high output resistance," *Electron. Lett.*, vol. 28, no. 4, pp. 361–363, Feb. 1992.

[49] R. L. Geiger, P. E. Allen, and N. R. Strader, *VLSI Design Techniques for Analog and Digital Circuits* (The McGraw-Hill Series in Electrical Engineering). New York, NY, USA: McGraw-Hill, 1990,

[50] W. A. Kester, *Data Conversion Handbook*. Newnes, 2005.

[51] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016, pp. 1–13.

[52] *IA-32 Intel Architecture Optimization Reference Manual*, Intel Corp., Mountain View, CA, USA, 2003.

[53] *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Adv. Micro Devices, Sunnyvale, CA, USA, 2010.

[54] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2013, pp. 448–459.

[55] P. Mertins *et al.*, "Proteogenomics connects somatic mutations to signalling in breast cancer," *Nature*, vol. 534, no. 7605, pp. 55–62, 2016.

[56] J. Torres-Sospedra *et al.*, "UjiindoorLoc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems," in *Proc. Int. Conf. Indoor Positioning Indoor Navigat. (IPIN)*, 2014, pp. 261–270.

[57] M. Lichman. (2013). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[58] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Lab., Palo Alto, CA, USA, Tech. Rep. HPL-2009-85, 2009.

[59] S. Li *et al.*, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. MICRO*, 2009.

[60] M. O'Connor *et al.*, "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 41–54.

[61] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 377–384.

[62] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm design exploration," in *Proc. 7th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2006, pp. 585–590.

[63] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

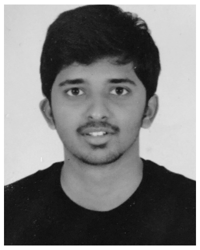
[64] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: Threshold adaptive memristor model," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 1, pp. 211–221, Jan. 2013.

[65] *Free PDK 45 nm Open-Access Based PDK for the 45 nm Technology Node*. Accessed: Nov. 22, 2017. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK

[66] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. ISCA*, 2011, pp. 365–376.

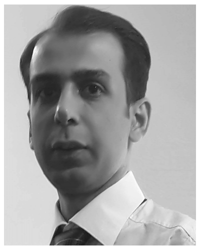
[67] G. Palumbo and D. Pappalardo, "Charge pump circuits: An overview on design strategies and topologies," *IEEE Circuits Syst. Mag.*, vol. 10, no. 1, pp. 31–45, Mar. 2010.

[68] C.-W. Hsu *et al.*, "Self-rectifying bipolar TaO_x/TiO_2 RRAM with superior endurance over 10^{12} cycles for 3D high-density storage-class memory," in *Proc. Symp. VLSI Technol.*, Jun. 2013, pp. T166–T167.



Yomi Karthik Rupesh received the B.Tech. degree in electronics and communication from the Vellore Institute of Technology, Vellore, India, in 2016. He is currently working toward the M.S. degree in electrical and computer engineering at the University of Utah, Salt Lake City, UT, USA.

He is also a Graduate Researcher at the Energy-Efficient Computer Architecture Laboratory, University of Utah. His current research interests include designing energy-efficient architecture for big data processing.



Payman Behnam received the B.S. degree in computer science and engineering from Shiraz University, Shiraz, Iran and the M.S. degree in computer science and engineering from the University of Tehran, Tehran, Iran. He is currently working toward the Ph.D. degree at the School of Computing, University of Utah, Salt Lake City, UT, USA.

He is also a Graduate Researcher at the Energy-Efficient Computer Architecture Laboratory, University of Utah, where he is involved in designing energy-efficient memory-centric accelerators for emerging data intensive applications.

Goverdhan Reddy Pandla received the B.E. degree in electronics and communication engineering from the R. V. College of Engineering, Bengaluru, India, in 2013. He is currently working toward the M.S. degree in computer engineering at the University of Utah, Salt Lake City, UT, USA.

His current research interests include computer architecture and wireless sensing systems.

Manikanth Miryala received the M.S. degree in computer engineering from the University of Utah, Salt Lake City, UT, USA, in 2017.

He was a Graduate Researcher at the Energy-Efficient Computer Architecture Laboratory, University of Utah. He is currently with Intel, Hillsboro, Oregon, where he is involved in system-on-chip power management.

Mahdi Nazm Bojnordi received the Ph.D. degree in electrical and computer engineering from the University of Rochester, Rochester, NY, USA, in 2016.

He is currently an Assistant Professor at the School of Computing, University of Utah, Salt Lake City, UT, USA, where he leads the Energy-Efficient Computer Architecture Laboratory. His current research interests include energy-efficient architectures, low-power memory systems, and the application of emerging memory technologies to computer systems.

Dr. Bojnordi received the two IEEE Micro Top Picks Awards, the HPCA 2016 Distinguished Paper Award, and the Samsung Best Paper Award for his research.