# In-Data vs. Near-Data Processing: The Case for Processing in Resistive CAM

Leonid Yavits, Roman Kaplan and Ran Ginosar

## ABSTRACT

Near-data in-storage processing research has been gaining momentum in recent years. Typical processing-in-storage architecture places a single or several processing cores inside the storage and allows data processing without transferring it to the host CPU. Since this approach replicates von Neumann architecture inside storage, it is exposed to the problems faced by von Neumann architecture, especially the bandwidth wall. We present a novel processing-in-storage system based on Resistive Content Addressable Memory (RCAM). RCAM functions simultaneously as a storage and a massively parallel associative processor. RCAM processing-in-storage resolves the bandwidth wall faced by conventional processing-in-storage architectures by keeping the computing inside the storage arrays, thus implementing in-data, rather than near-data, processing. We show that RCAM based processing-in-storage architecture may outperform existing in-storage designs and accelerator based designs. RCAM processing-in-storage implementation of *K*-means achieves speedup of 4.6—68 relative to CPU, GPU and FPGA based solutions. For *K*-Nearest Neighbors, RCAM processing-in-storage achieves speedup of 17.9—17,470 and for Smith-Waterman sequence alignment it reaches speedup of almost 5 over a GPU cluster based solution.

## Keywords

Near-data Processing; Associative Processing; Processing-in-storage; Processing-in-Memory; RRAM; CAM; Memristors.

## 1. INTRODUCTION

In von Neumann architecture, execution time comprises data processing time $T_{CPU}$ (divided by scaling factor $S_{CPU}$) and data transfer time $T_{MEM}$ which is a function of memory bandwidth $BW$:

$$T_{EXEC} = \frac{T_{CPU}}{S_{CPU}} + T_{MEM}(BW) \quad (1)$$

Historically, $T_{CPU}$ scales much faster than $T_{MEM}$. Further scaling of $T_{CPU}$ by increasing $S_{CPU}$ through improving the instruction level parallelism or adding more cores has a diminishing effect on overall execution time.

The premise of near-data processing is reducing $T_{MEM}$ by cutting the physical distance and increasing the bandwidth between CPU and memory. Since inception, near-data processing mainly meant processing in memory (PIM). To process datasets larger than memory footprint, processing units are placed near storage, achieving "processing-in-storage."

We believe that near-data processing-in-storage is inherently limited because it is largely based on replicating the von Neumann architecture near data storage. Hence it potentially faces some of von Neumann architecture problems, such as the bandwidth wall.

This work presents a novel resistive CAM (RCAM)-based processing-in-storage architecture with *in-data* rather than near-data processing-in-storage. The RCAM processing-in-storage system simultaneously functions as data storage and a massively parallel SIMD accelerator that performs the computations *in-situ*, resulting in increased performance through more complete utilization of the internal storage bandwidth, and reduced energy consumption.

RCAM processing can be implemented in different hierarchies of resistive storage and memory. While it can be implemented in the mass storage, cost-wise a better place for RCAM processing-in-storage could be an intermediate storage hierarchy between the main memory and mass storage, for example a storage class memory.

This paper makes the following contributions:

- We present a RCAM architecture offering storage with in-data processing capabilities.
- We develop a RCAM processing-in-storage based implementation of several algorithms in the fields of machine learning and bioinformatics.
- We show that RCAM processing-in-storage implementations can outperform near-data or other (CPU, GPU or FPGA based) implementations both in performance and in power efficiency.

The rest of this paper is organized as follows. Section 2 presents the motivation and related work. Section 3 introduces the architecture of RCAM processing-in-storage system. Section 4 reviews the principles of associative processing. Section 5 explores RCAM programming, and applications. Section 6 presents simulation setup and comparative performance of several big data algorithms. Section 7 offers conclusions.

## 2. Background and Motivation

Near-data processing research has gained momentum recently. Typical processing-in-storage architecture places a single or several processing cores inside the storage and allows data processing without transferring it to the host processor. The concept of near-data processing-in-storage is

illustrated in Figure 1a. A comprehensive review of near-data processing can be found in [8].

Processing-in-storage research mainly focuses on processing data in NAND flash based solid state disk (SSD). Boboila *et al*. [10] proposed Active Flash, a processing in solid-state storage that expedites data analysis by migrating the data to the flash device. The authors explored energy and performance trade-offs of their processing-in-storage architecture. Bae *et al*. [7] introduced the notion of Intelligent SSDs, exploring the design considerations and examining their potential benefits in data mining applications. Continuing the work on Intelligent SSD, Jo *et al*. [31] studied optimal ways of combining CPU, GPU and SSD for efficient processing of data-intensive algorithms. Cho *et al*. [12] cited the lack of parallel processing abilities in earlier in-SSD processing architectures and proposed integrating a GPU, providing API sets based on the MapReduce framework. Kang *et al*. [33] introduced the Smart SSD model, which combines in-SSD processing with a powerful host system, and constructed a Smart SSD prototype. De *et al*. [17] introduced the FPGA-based Minerva, which executed application-specific operations in the NVM controller. Jun *et al*. [32] introduced and constructed BlueDBM, combining a flash based storage with in-store processing capability and a low latency high-throughput inter-controller network, and explored its performance benefits. Cho *et al*. [13] explored some of the questions which are also addressed by this paper. The authors made a case for Intelligent SSD by discussing the bandwidth trends and quantifying the potential benefits of processing-in-storage across a range of applications.

While processing-in-storage research is relatively young, the wider concept of near-data processing, focusing mainly on processing in memory (PIM) has been thoroughly researched. The concept of mixing memory and logic has been around since 1960s. The DAPP, STARAN, CM-2, and GAPP computer architectures [51] used large number of PUs positioned in proximity to memory arrays to implement a massively parallel SIMD computer.

[47] suggested replacing the last level cache and the vector co-processor of a conventional high-performance CPU by an associative processor, which is a PIM accelerator, combining data storage and massively parallel SIMD processing capabilities.

While embedding processing with conventional 2D DRAM chips is less practical, recent advancement in 3D memory and logic stacking technology may remove this obstacle. Citing severe bandwidth limitations in conventional computer architecture as datasets continue to grow, Ahn *et al*. [1] introduced Tesseract, a 3D Processing in Memory accelerator for large-scale graph processing. In another work, Ahn *et al*. [2] developed a hybrid memory cube based framework that automatically decides whether to execute PIM operations in memory or processors depending on the locality of data. Nair, Sura *et al*. [58][44] introduced the Active Memory Cube, a heterogeneous computing system including general-purpose host processors and specially designed in-memory processors that would be integrated in a logic layer within 3D DRAM memory. In another work, Gao *et al*. [43] developed hardware and software of a 3D stack memory and near-data processing architecture for in-memory analytics frameworks, including MapReduce, graph processing, and deep neural networks. Azarkhish *et al.* [6] developed Smart Memory Cube and designed a high bandwidth interconnect to serve the bandwidth demand of PIM architecture. Zhang *et al*. [64] explored PIM implemented via 3D die stacking. Akin *et al*. [3] addressed the issue of data reorganization in 3D stacked near-data processing architecture, introducing HAMLeT, a mechanism for host interference, bandwidth allocation, and in-memory coherence. Farmahini-Farahani *et al*. [22] proposed NDA, a near-DRAM acceleration architecture that processes data using accelerators 3D-stacked on DRAM devices.

Recently, emerging memory technologies such as resistive memory have become a focus of PIM research. Somnath *et al*. [50] developed MBARC, a resistive crossbar in-memory LUT-based processing architecture. Chi *et al*. [11] introduced PRIME, a PIM accelerator of neural network applications. [48] introduced a resistive CAM based massively parallel accelerator. Shafiee et al. [56] developed ISAAC, an in-situ accelerator of neural network, where memristor crossbar arrays are used to perform dot-product operations in an analog manner.

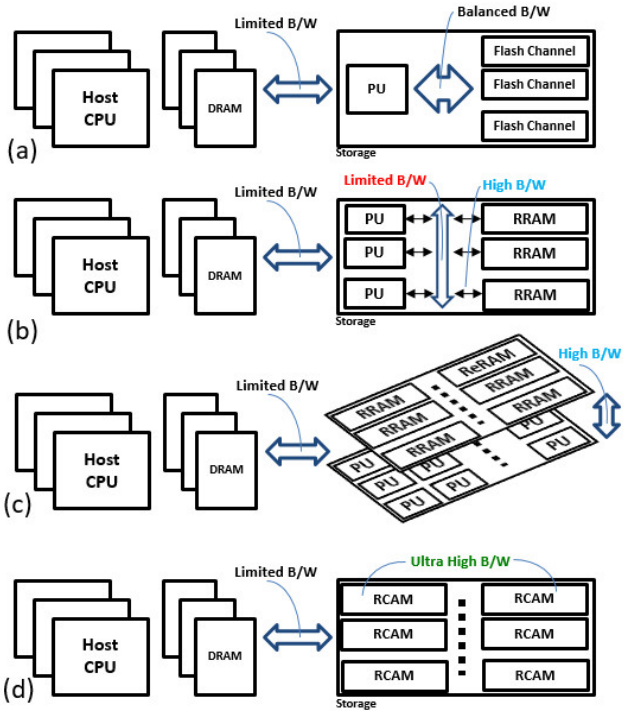


**Figure 1: (a) Near-Data Processing in Flash Based Storage; (b) 2D Near-Data Processing in RRAM Based Storage; (b) 3D Near-Data Processing; (d) In-Data Processing in RCAM Based Storage.**

We believe that near-data processing-in-storage is inherently limited because it is based on replicating the von Neumann architecture in a storage. Hence it potentially faces some of von Neumann architecture problems, such as the bandwidth wall. We define the computation throughput of an in-storage processor as follows:

$$Throughput_{Computation} = \frac{Size_{Dataset}\ [Byte]}{Runtime\ [sec]} \qquad (2)$$

For processing-in-storage systems to reach optimal performance, the peak computation throughput of an in-storage processor should match the internal bandwidth of that storage. The upper bound of such bandwidth is defined by the maximum bandwidth of flash arrays, and ranges from few hundred MB/s to few GB/s depending on the number of parallel flash channels [49].

Early works on in-SSD processing report the computation throughput of several MB/s to a few hundred MB/s depending on workload (for example, 7MB/s to 350MB/s in [10]). However, as the number of flash channels in SSD grows, so does the effective internal SSD bandwidth. A conventional response to the growing internal bandwidth is increasing parallelism by adding more in-SSD processing cores. One example of such increased parallelism is placing a processing core in each flash channel [31]. However, with the advancement of non-charge based memory technologies, there is a growing consensus that resistive memory has a potential to replace flash in future SSDs [4]. With bandwidth and latency characteristics similar to DRAM [14], resistive memory may significantly increase the upper bound of the internal SSD bandwidth. This may lead to the following two scenarios. First, increasing the parallelism by adding more in-SSD processing cores will become inefficient and may eventually cause a reduction in performance [63]. Second, internal storage bandwidth is likely to become limited by the internal communication bus/network (Figure 1b) due to the surge in inter-core communication [63]. Both scenarios repeat the problems faced by manycore von Neumann architectures in the "macro" world.

As suggested in [8], the compute throughput to internal SSD bandwidth balance can be regained through new system-on-chip and die stacking technologies that enable network-on-chip integration, a more efficient network software stack, and potentially new opportunities for near-data processing-customized interconnect designs.

The concept of 3D near-data processing architecture is illustrated in Figure 1c. 3D stacking of RRAM and a parallel in-SSD processor, with some ultra-wide vertical communication capabilities, has the potential to realize the bandwidth upside of the future NVM. This is certainly a valid potential direction of the near-data processing architecture development.

In this paper, we propose a new processing-in-storage architecture that increases the compute throughput to match the potentially ultra-high internal bandwidth of the storage arrays. This architecture progresses from random addressable to content addressable (associative) storage (Figure 1d). This architecture enables massively parallel SIMD processing of the data *inside* the storage arrays. The processing is associative, making the dedicated in-storage processors redundant. There is no data transfer outside the storage arrays through a bandwidth limited internal SSD communication bus/network. We refer to the RCAM processing-in-storage as *in-data* rather than *near-data* processing architecture. The inherent performance (read/write access time and bandwidth) of the resistive memory can be utilized to the full extent, enabling very high computation throughput while reducing the energy consumption (mainly due to the lack of data movement inside the SSD).

The main reason to prefer in-data RCAM processing-in-storage over 3D stacked near-data processing is the per-bit connectivity of memory and processing: In RCAM, each memory bit is directly connected to processing transistors, whereas in 3D stacked near-data processing, the data must pass through memory interface circuits and through 3D vertical interconnects, typically much fewer in numbers than the number of bits. In RCAM processing-in-storage, the bulk of data ideally never leaves the memory. The computation is performed within the confines of the memory array. This potentially holds a significant performance and energy efficiency advantage: Using DRAM as an example, there is typically a reduction in available bandwidth of six orders of magnitude between the sense amplifiers and the CPU edge [8]. In addition, the cost of access in terms of energy increases from hundreds of femtojoules to tens of picojoules over a span of the same distance [8].

The use of STT-MRAM and Resistive Ternary CAM for data intensive computing was pioneered by Guo *et al*. [27][28][29]. Guo *et al*. used the associative capabilities of CAM and Ternary CAM mainly for search operations, while the computing is largely done in a CPU. Their work targeted a different architecture, replacing RAM by resistive CAM or ternary CAM in NVDIMM rather than in mass storage. Adopting associative processing architectures such as Goodyear Aerospace's STARAN or MPP to processing-in-storage is also suggested in [8].

## 3. Architecture

Resistive memories store information by modulating the resistance of nanoscale storage elements. They are nonvolatile, free of leakage power, and emerge as potential alternatives to charge-based memories, including NAND flash. The metal-oxide resistive random access memory (RRAM) is considered as one of the potential technologies to replace next-generation nonvolatile memories [4]. Its main features are high reliability and fast access speed. A test-chip of 32GB device with two RRAM-based memory layers and a CMOS logic layer underneath has been demonstrated [38]. While RRAM [4] employs one transistor and one memristor (1T1R) cell, RCAM processing-in-storage uses 2T2R cells [36] and appropriate peripheral circuits [48] to support associative storage and processing. A number of alternative

resistive CAM and ternary CAM cell designs have been proposed [5][21][41][42][61].

### 3.1 RCAM processing-in-storage system

The top-level view of RCAM processing-in-storage system and its possible positions within memory hierarchy is presented in Figure 2.

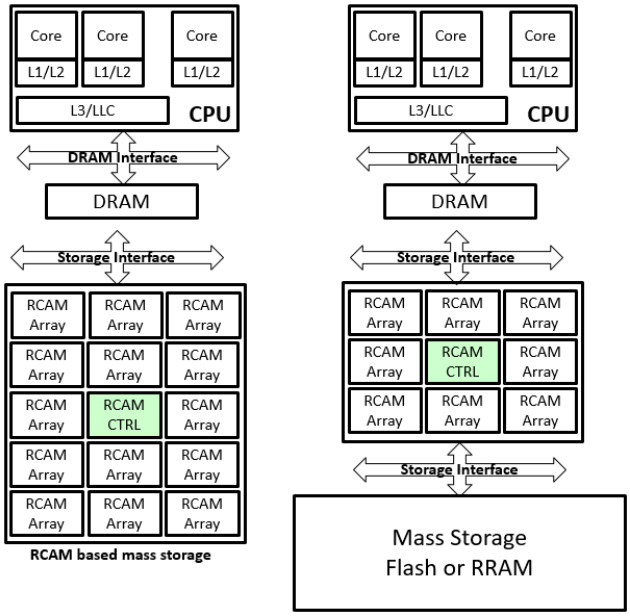


**Figure 2: RCAM Position in Memory Hierarchy.**

RCAM processing-in-storage comprises a multitude of RCAM arrays, possible divided into multiple ICs, with a central microcontroller. The mass storage may be implemented by RCAM rather than RRAM or flash. This will enable massively parallel in-mass storage processing however this option comes at relatively high cost since RCAM is less dense than RRAM. Another candidate is an additional memory hierarchy between the main memory and mass flash or RRAM storage, similarly to a storage class memory. Such option could provide a better performance-cost trade-off.

### 3.2 RCAM Array

RCAM array is the heart of RCAM processing-in-storage architecture, presented in Figure 3. It comprises a resistive memory crossbar, in which each memory line is also a baseline processing unit (PU), and a peripheral circuitry. The latter includes a microcontroller, key and mask registers, tag logic, and two optional circuits: a tag counter or reduction tree and a daisy-chain interconnect. The basic RCAM cell is created by virtually pairing two RRAM cells (memristors), holding complementary values $R$ and $\bar{R}$.

The resistive (memristor based) CAM is a scalable and highly dense alternative to CMOS CAM. Memristors are two-terminal devices, where the resistance of the device is changed by the electrical current or voltage. The resistance of the memristor is bounded by a minimum resistance $R_{ON}$ (low resistive state, logic '1') and a maximum resistance $R_{OFF}$ (high resistive state, logic '0').

The *key* register (Figure 3a) contains a key data word to be written or compared against. The *mask* register defines the active fields for write, compare and read operations, enabling bit selectivity. The *tag* marks the rows that are matched by the compare operation and are to be affected by the successive parallel write. A daisy-chain like bitwise interconnect allows PUs to intercommunicate, all PUs in parallel. The tag counter is a reduction (adding) tree, enabling logarithmic summation of tag bits. This operation is useful whenever a vector needs to be reduced to a scalar.

The RCAM compare operation is implemented as follows. The Match/Word line is precharged and the key is set on Bit and Bit-not lines. In the columns that are ignored during comparison, the Bit and Bit-not lines are kept floating. If all unmasked bits in a row match the key (*i.e.*, when Bit line '1' is applied to an $R_{ON}$ memristor and Bit-not line '0' is applied to an $R_{OFF}$ memristor, or vice versa), the Match/Word line remains high and '1' is sampled into the corresponding TAG bit. If at least one bit is mismatched, the Match/Word line discharges through an $R_{ON}$ memristor and '0' is sampled into the TAG.

Write operation is performed in two phases. First, the $V \geq V_{ON}$ voltage (where $V_{ON}$ is a threshold voltage required to switch to the "on" state) is asserted to applicable Bit lines (to write '1's) and Bit-not lines (to write '0's). Second, the $V \leq V_{OFF}$ voltage (where $V_{OFF}$ is a threshold voltage to switch to the "off" state) is asserted to Bit-not lines (to complement the '1's) and Bit lines (to complement '0's). The write affects only the tagged rows.

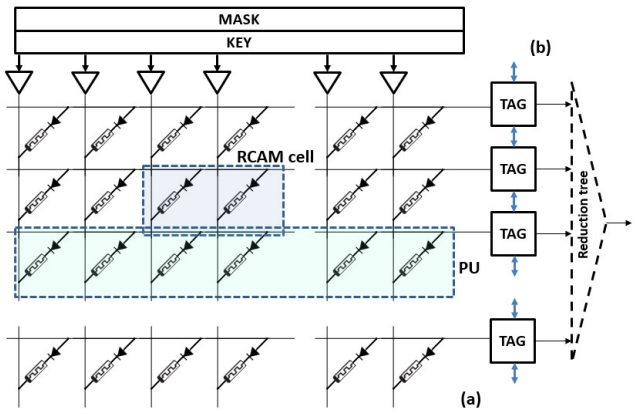


**Figure 3: RCAM Array: (a) Resistive Crossbar and (b) Peripheral Circuitry**

Memristor sub-nanosecond switching time [59] allows GHz RCAM processing-in-storage operation. The energy consumption during compare may be less than 1fJ per bit. The write energy is in the range of 0.1pJ to 3pJ per bit [62], which may be prohibitively high for simultaneous parallel writing of the entire RCAM storage; the energy consumption is addressed in Section 6.

Another factor which potentially limits RCAM processing-in-storage system is endurance (the number of program/write cycles that can be applied to a memristor before it becomes unreliable). Resistive memory endurance is shown at about $10^{12}$ [62], which may suffice for only about one month. However, studies predict that the endurance of resistive memories may grow to the $10^{14} - 10^{15}$ range [21][45], extending RCAM processing-in-storage system endurance to a number of years.

### 3.3 Tag and Match Circuits

The tag logic is presented in Figure 4. It comprises a pre-charge circuit, a Match line sense amplifier, a tag flip-flop, a multiplexor (implementing the daisy-chain TAG connectivity), a first_match circuit and an if_match circuit. The Match line is pre-charged during compare. The tag register latches the result of compare. The First_match circuit implements 'match first,' a frequent associative operation, by keeping only first match and resetting the remaining tags. If_match, another frequent associative operation, returns '1' if a parallel compare operation results in at least one match.

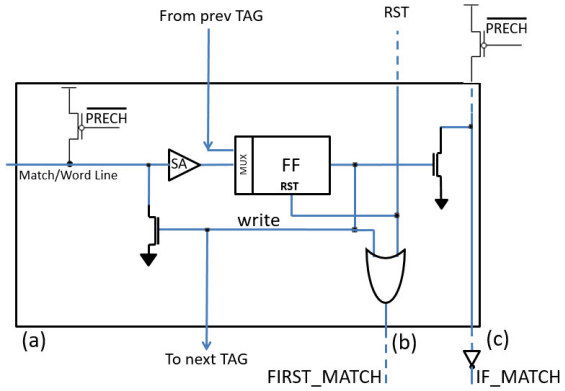


**Figure 4. TAG Logic: (a) TAG, (b) First_match, (c) If_match.**

### 3.4 System Architecture and RCAM scaling

Conceptually, RCAM may comprise hundreds of millions of rows, each serving as a processing unit (PU). Due to thermal limitations, the entire array may be divided into multiple ICs (Figure 5a).

The RCAM processing-in-storage system uses a *microcontroller* (Figure 5b). It issues instructions, sets the key and mask registers, handles control sequences and executes read requests. In addition, the microcontroller contains the RCAM buffer, which stores the reduction tree outputs. The microcontroller may also perform some baseline processing, such as normalization of the reduction tree results. Presently, RCAM software, including both associative operations (SIMD array instructions) and sequential instructions executed on the microcontroller, is manually encoded at assembly language level.

The scaling of conventional near-data processing architectures may be limited, similarly to high-performance parallel von Neumann architectures. When growing internal bandwidth of the storage arrays is met by increasing number of in-storage processing cores, the storage array to in-storage processor communication bottleneck worsens. As a result, the performance of processing-in-storage system may saturate or even diminish.

RCAM processing-in-storage provides much better scalability. Its inherent parallelism allows increasing the performance of many workloads almost linearly as the datasets grow along with storage size. Since the bulk of data is never transferred outside the storage arrays through a bandwidth-limited communication interface, the performance limit is pushed further away.

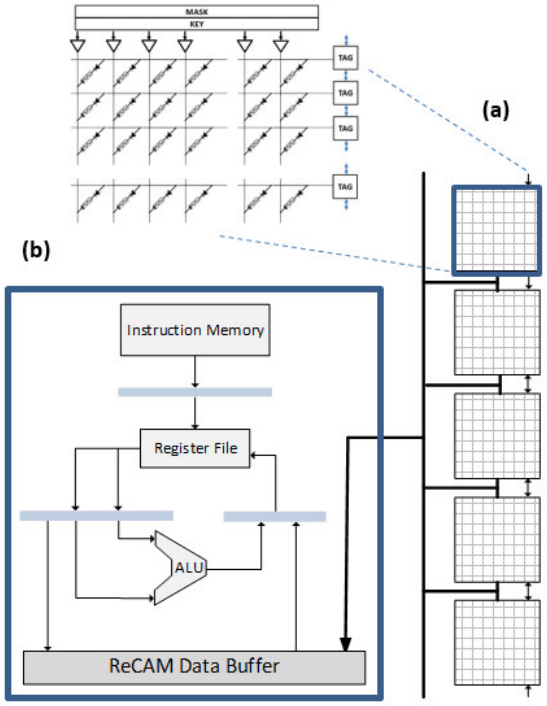


**Figure 5: RCAM processing-in-storage system is composed (a) of separate multiple ICs and (b) a microcontroller.**

## 4. Associative Processing

RCAM is a non-von Neumann associative in-storage processor. Most computations may be structured as series of Boolean functions, and Boolean functions can be implemented on RCAM using truth table executions. The data are stored in the RCAM array, one data element per RCAM row (PU). The truth table entries, embedded in the microcode, are broadcast entry-by-entry by the RCAM microcontroller.

The input part of each truth table entry is matched against the entire RCAM content (the entire data set). The matching RCAM rows are tagged, and the corresponding truth table output values are written into the designated fields of the tagged rows. For an $m$-bit argument $x$, any Boolean function $b(x)$ has $2^m$ possible output values. Therefore, a naïve associative computing operation would incur $O(2^m)$ cycles,

regardless of the data set size. More efficiently, arithmetic operations can be performed on RCAM in a word-parallel, bit-serial manner, reducing time complexity from $O(2^m)$ to $O(m)$. For instance, vector addition may be performed as follows [24]. Suppose that two $m$-bit RCAM columns hold vectors A and B. The sum of A+B is written onto another $m$-bit column S (Figure 6a). A one-bit column C holds the carry bit. The operation is carried out as $m$ single-bit additions (3):

$$c[:] \mid s[:]_i = a[:]_i + b[:]_i + c[:], \quad i = 0, \ldots, m-1 \quad (3)$$

where $i$ is the bit index, ':' means all elements of the vector, and $c$ and $s$ are, respectively, the carry and sum bits. The single-bit addition is carried out in a series of steps. In each step, one entry of the truth table (a three bit input pattern, Figure 6c) is matched against the contents of the $a[:]_i, b[:]_i, c[:]$ bit columns and the matching rows (PUs) are tagged; the logic result (two-bit output of the truth table, Figure 6c) is written into the $c[:]$ and $s_i[:]$ bits of all tagged rows. During that operation, all but three input bit columns and two output bit columns of the associative array are masked out in each step. Overall, eight steps of one compare and one write operation are performed to complete a single-bit addition over all rows, regardless of the number of rows.

A snapshot of such vector addition, for $m = 4$, for the zero bit of the vector elements and the 2nd entry of the truth table is shown in Figure 6. During compare (Figure 6a), the input pattern '001' is compared against bit columns c, $a_0$ and $b_0$, for all vector elements in parallel. The matching rows (two in this example) are tagged. During write (Figure 6b), the output pattern '01' is written in bit columns c and $s_0$ accordingly. Only the tagged rows are affected by write.

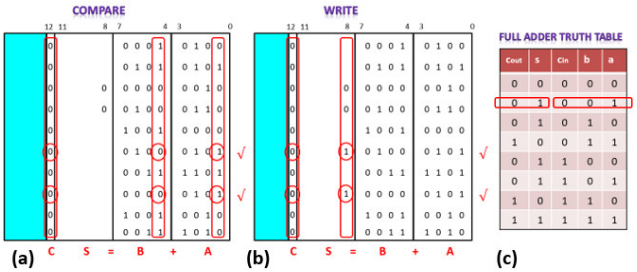


**Figure 6: Vector addition in RCAM example, for two 4-bit vectors, snapshot at zero bit, 2nd entry of the truth table: (a) Compare, (b) Write, (c) Full Adder Truth Table.**

A fixed-point $m$ bit addition and subtraction take $O(m)$ cycles. Fixed point multiplication and division in RCAM processing-in-storage architecture require $O(m^2)$ cycles. Single precision floating point multiplication takes 4,400 cycles [47], regardless of the data set size.

# 5. Programming and applications

## 5.1 Programming RCAM processing-in-storage

In RCAM processing-in-storage, the host is responsible for running the OS and sequential code, and RCAM implements parallel SIMD kernels. The host transfers execution parameters such as dataset addresses, and triggers RCAM kernel execution.

We analyze applications to find highly parallelizable data intensive SIMD phases. We divide the application into sequential (run by the host) and in-storage kernels (executed on RCAM). The code intended to run on RCAM is translated into associative primitives.

The host invokes the RCAM to perform its code fraction. It sends the workload parameters to RCAM and starts execution. Once RCAM execution completes, the host can access the RCAM output.

There is no hardware support for data coherence between the host CPU and RCAM storage. RCAM has no access to the host main memory or on-chip cache. Therefore, the datasets on which RCAM operates must reside in RCAM and should not be left in the host memory. To avoid inconsistencies between the RCAM and host CPU memory, RCAM storage is inaccessible to the host CPU during the RCAM operation.

## 5.2 Applications

In this section, we discuss the implementation of several compute-intensive workloads from different application fields. The first is sparse matrix multiplication, frequently used in machine learning, for example in linear Support Vector Machine classification and regression. Another is *K*-means, a clustering algorithm for classification. The third algorithm is K-Nearest Neighbors (KNN), another classification and regression kernel. Last, we present the Smith-Waterman sequence alignment, a basic tool in bioinformatics. All these algorithms, performed in RCAM processing-in-storage architecture in our work, are compared with CPU and accelerator-based implementations.

Somewhat less ambitious applications include data intensive searches such as *string matching,* addressed by several near-storage or in-SSD architectures [33][49]. Clearly, whereas the complexity of reading data out of storage for performing search by in-SSD cores is of linear time complexity, performing search in RCAM processing-in-storage architecture is closer to constant time complexity, and is not shown here.

### *5.2.1 Sparse Matrix-Vector Multiplication*

Sparse matrix by dense vector multiplication (SpMDV or SpMV) is typically constrained by memory bandwidth limitations, and hence the efficient implementation of SpMV is critical to large scale linear algebra applications.

We propose a fully associative algorithm for SpMV execution in RCAM processing-in-storage architecture. Revised versions of this algorithm can be used for dense matrix multiplication and sparse matrix by sparse matrix [46] or sparse vector multiplication.

Figure 7 presents the algorithm of RCAM SpMV. Matrix A is assumed to be stored in RCAM in Compressed Sparse Row (CSR) format, where each nonzero element $e_A$ is stored alongside its column index $i_A$.

The algorithm includes three parts. The first part, broadcast, consists of a loop going over the elements of vector $B$. In a first cycle, the index of an element of $B$, $i_B$, is compared against the column index field of the entire matrix $A$ (in parallel for all nonzero elements of $A$, using the compare command). All index-matching rows holding nonzero elements of matrix $A$ are tagged.

**Algorithm 1** SpMV

```
//Let A, B, C denote matrix A and vectors B and C.
//Each RCAM row holds a non-zero element of A (e_A, i_A)
    // Broadcast
1:  For each e_B ∈ {elements of B}:
        // Compare i_B with all column indices of A, i_A
2:      Compare i_B to all i_A
        // Write e_B into all matching rows
3:      Write e_B
    // Associatively multiply the entire A by B
4:  PR ← e_B * e_A      // PR is a matrix
    // Reduction: all rows of A in parallel, each row is tallied
5:  For each (non-zero) row k of A:
6:      C_k ← Reduction(PR_k)
    // C has non-zero elements where A has non-zero rows
```

**Figure 7: RCAM based SpMV pseudocode.**

In the second cycle, $e_B$ is written simultaneously into all tagged rows, alongside the index-matched elements of matrix $A$. The loop is repeated for all elements of vector $B$. Upon completion, each nonzero pair of elements of $A$ and $B$ required to calculate the product vector C is aligned (stored in the same row) in the RCAM.

The second part (step 4) is the associative multiplication of the $e_A, e_B$ pairs, performed in parallel for all pairs. The number of multiplications performed simultaneously equals the number of nonzero elements in $A$.

The third part sums the products along each row of $A$ (steps 5, 6) using the reduction tree.

RCAM SpMV has the computational complexity of $O(nc_A + nr_A)$ where $nc$ and $nr$ are the number of columns and rows, respectively.

### 5.2.2 K-Means

*K-means* is an unsupervised learning algorithm for clustering unclassified samples. It aims to partition $N$ samples into $K$ clusters, where each observation belongs to the cluster with the nearest mean.

The $K$-means algorithm pseudocode, as implemented in storage, is presented in Figure 8. The algorithm minimizes the Euclidean distances between the samples and the cluster centers (the means), as follows. Prior to execution, the means are initialized by randomly choosing $K$ samples and the minimum Euclidean distance of all samples is initialized to the highest possible value.

The algorithm consists of two $K$ iterations loops, *assignment* and *update*. The *assignment* loop finds the closest mean of each sample. The *update* loop recalculates the new mean coordinates. These two loops may be repeated until the mean coordinates convergence.

In the *assignment* loop, each sample is assigned to the cluster whose mean yields the minimal Euclidean distance. In each iteration of lines 3-6 of Figure 8, the distance over a single attribute is associatively calculated in parallel for all dataset samples. Next, in lines 7-9, the minimal Euclidean distance and the cluster assignment are updated in parallel for all samples $x \in X$. Note that lines 2-9 are always executed in parallel on the entire storage, in a SIMD-like style.

In the *update* loop, for each mean index $i_{mean}$, all samples assigned to this mean are tagged (line 11). Then, for each attribute and for all tagged rows, the sum of coordinates is calculated in parallel using the Reduction Tree (line 13), followed by counting the number of samples assigned to the mean (line 14) and finally calculating the new mean coordinates by the microcontroller (line 15).

Note that the key computational steps are parallelized, and in the *update* loop parallelism is made possible by associativity.

**Algorithm 2** *K*-Means Implementation in RCAM

```
// X: the group of samples
// Every x ∈ X is stored in a separate RCAM row

//Assignment: assign each sample with a cluster
// Each of the k means is a tuple: (i_mean, Mean)
1:  For each i_mean ∈ [1,K]:
        Do-all x ∈ X:
2:          Write mean coordinates to temp columns
3:          For each attr ∈ {sample attributes}:
4:              dist_attr ← x_attr − mean_attr
5:              sqDist_attr ← (dist_attr)^2
6:              sqDist_to_mean ← sqDist_to_mean + sqDist_attr
7:          Tag rows with sqDist_to_mean < min_sqDist
8:          Write min_sqDist ← sqDist_to_mean
9:          Write i_assigned_mean ← i_mean

// Update: calculate new mean coordinates
10: For each i_mean ∈ [1,K]:
11:     Tag rows with i_assigned_mean == i_mean
12:     For each attr ∈ {sample attributes}:
13:         Sum_attr ← Reduction(x_attr)
14:         Cluster_size ← Reduction(match_lines)
15:         mean_attr ← Sum_attr/Cluster_size
```

**Figure 8: *K*-Means Pseudocode for a single iteration of the algorithm.**

### 5.2.3 K-Nearest Neighbors (KNN)

*K*-nearest neighbors (KNN) is frequently used for classification. It computes the distances between an (unclassified) input query vector and a dataset of classified

samples. Each sample consists of multiple *attributes* (features or dimensions). The query vector classification is usually determined by the majority vote of $K$ nearest database samples, hence the name *K-nearest neighbors*. Distance is most commonly Euclidean, although Manhattan or Hamming distance might occasionally be used.

In a von Neumann machine, the required computational effort is proportional to dataset size and is the main cause for limited performance on large datasets. In contrast, in-data implementation of KNN is not limited by dataset size and can therefore provide high performance on very large datasets.

KNN algorithm pseudocode on RCAM is presented in Figure 9. This implementation calculates the Euclidean distance between the query vector and the dataset samples, followed by serially selecting the $K$ closest samples. The algorithm comprises two steps. The first step computes the Euclidean distance (squared) between the query vector and each dataset sample. In each iteration of the first step (lines 1-4 in Figure 9), deltas of one attribute are calculated in parallel for all samples (line 2), squared (line 3) and added to the final distance $sqDist$ (line 4). The number of iterations in the first loop equals the number of attributes $M$.

The second step (lines 5-9) iteratively finds the $K$ dataset samples that are closest to the query vector (the nearest neighbors), one by one.

**Algorithm 3** KNN Implementation in RCAM

//$K$ denotes the number of nearest neighbors.
//Every sample $x \in X$ may be stored in several consecutive RCAM rows; the code assumes one row per sample for simplicity.
//Each sample is characterized by $M$ attributes
//Calculate distance of each dataset sample from query
1: For each $attr \in [1, M]$
  Do-all $x \in X$:
2: $dist_{attr} \leftarrow Query_{attr} - x_{attr}$
3: $sqDist_{attr} \leftarrow (dist_{attr})^2$
4: $sqDist \leftarrow sqDist_{partial} + sqDist_{attr}$
//Assume unique $sqDist$ values

//Find $K$ closest samples
//Histogram of all classes maintained by microcontroller
//Start with all samples unmarked
5: Loop $K$ times
6: Tag all unmarked samples
7: Tag and mark first row with min value of $sqDist$
8: Retrieve $class$ of tagged row to microcontroller
9: On microcontroller: Histogram[$class$]++
//Classification: Class with highest histogram

**Figure 9: KNN Pseudocode.**

Every iteration tags the minimal unmarked Euclidean distance (lines 6-7), reads the tagged sample class (line 8) and increments a histogram counter for that class (line 9, performed by the mirocontroller). Overall the loop is iterated $K$ times.

### *5.2.4 Smith-Waterman DNA Sequence Alignment*

Searching for similarities in pairs of protein and DNA sequences (also called Pairwise Alignment) has become a routine procedure in Molecular Biology and it is a crucial operation in many bioinformatics tools. The Smith-Waterman algorithm (S-W) [57] provides an optimal solution for the pairwise sequence alignment problem. However, the optimality comes with a high computational cost, requiring a number of operations proportional to the product of the two sequences. The algorithm allows for some parallelism, but requires serial steps proportional to the length of the longest sequence of the two compared.

S-W identifies the optimal alignment of two sequences by computing a two-dimensional scoring matrix. Matchings base-pairs score positively (e.g., +2), while mismatching result in negative score (e.g., -1). The optimal alignment score between two sequences is the highest score in the matrix. The alignment may contain gaps in both sequences which are penalized in the score calculation (negative scores). According to the affine gap model [26], opening a gap is harder than extending it, therefore the penalty for opening a gap is larger. The S-W has two steps, scoring (to find the maximal alignment score) and trace-back to construct the alignment. The first step is the most computationally demanding and is the focus of our work.

Figure 10a shows snapshot of the scoring matrix during algorithm execution. Scores are represented by 32-bit integers. In a parallel implementation, the matrix is filled along the main diagonal and the entire anti-diagonal scores are calculated in parallel, as the figure illustrates. Two anti-diagonals are required to calculate the score of a new anti-diagonal, therefore in each iteration only three anti-diagonals are stored in memory. The data set may be distributed over a large number of ICs, as in Figure 5. Figure 10b shows the RCAM memory map of two neighboring ICs at the beginning of an iteration. $A$ and $B$ contain the sequences, where each base-pair takes 2-bit and resides in a separate row. $E$ and $F$ are partial score results of the affine gap model. $AD[0], AD[1]$ and $AD[2]$ contain scoring matrix anti-diagonals. Shift operations move data between rows inside a RCAM IC and between daisy-chained ICs. Figure 10c shows the RCAM memory map at the end of the iteration and the mapping between RCAM and the scoring matrix.

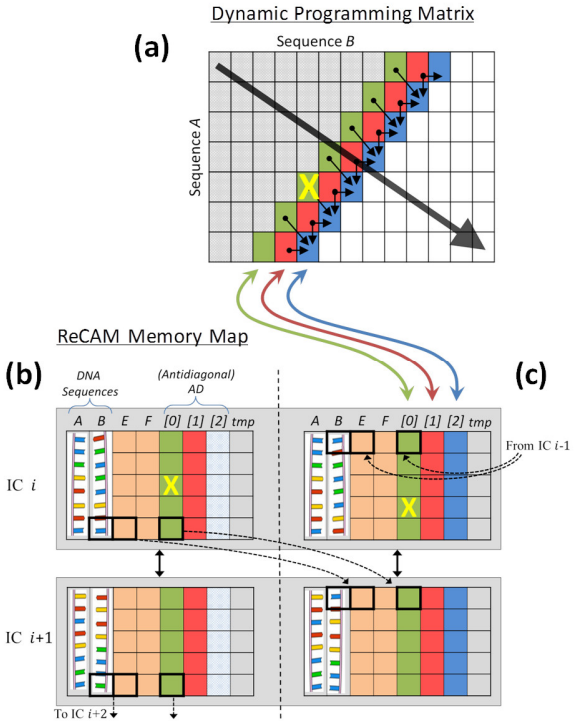


**Figure 10: (a) Snapshot of the dynamic programming matrix, showing the direction of progress for the parallel algorithm. (b),(c) The matching organization of data in the RCAM array at the beginning of an iteration (b) and its end (c). AD[2] contents in (b) is being replaced with the new result (c). Bottom rows in a RCAM IC are daisy-chained to the next IC in a shift instruction.**

# 6. Evaluation

## 6.1 Simulation Platform

We assume that RCAM is implemented in 28nm technology. We simulate RCAM using the associative processor simulator [47], with operating frequency of 500MHz. We have developed an in-house power simulator to evaluate the power consumption of the RCAM. The latency and energy figures used by both the timing and power simulations are obtained using SPICE simulation and are detailed in [48].

## 6.2 Sparse Matrix Vector Multiplication

To simulate sparse matrix multiplication, we have used the 21 square matrices from the UFL Sparse Matrix Collection [15] (listed in Figure 11), having 327,000 - 37 million nonzero elements. These matrices are also used for performance study by Saule *et al.* [55]. Performance of SpMV is presented in Figure 11(a), together with performance on Intel Xeon Phi SE10P and NVidia K20 [55]. These Xeon Phi SE10P and K20 SpMV implementations have $O(nnz_A)$ computational complexity ($nnz_A$ is the number of nonzero elements).

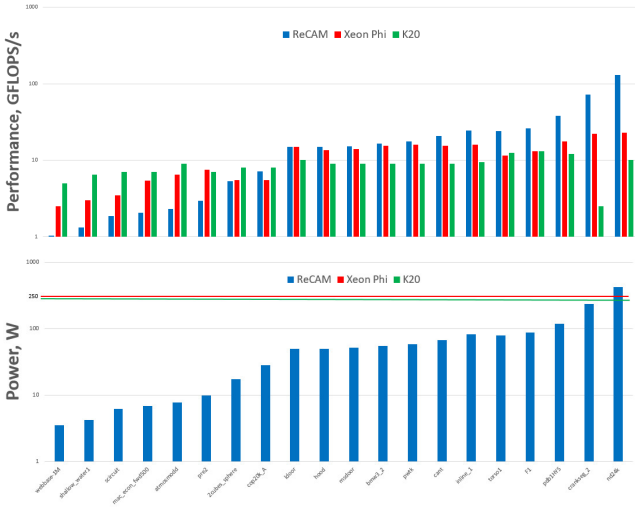


**Figure 11: SpMV (a) Performance, (b) Power Consumption**

It appears that these Xeon Phi SE10P and K20 implementations assume that matrices are preloaded to main memory (DRAM). Therefore, the time and energy spent on fetching those matrices from storage to main memory are not included in the evaluation [55]. In contrast, RCAM processing-in-storage architecture implements in-data SpMV, thus saving latency and energy consumption of the data transfer, and freeing the CPU and GPU to other tasks.

Power simulation of RCAM SpMV is based on upper-bound energy figures from [62], applicable to 28nm, same technology node as K20. The simulated power consumption of the SpMV is presented in Figure 11(b). The SpMV power efficiency of GPUs such as the 28nm NVidia GTX Titan is around 0.1 GFLOP/s/W [19] (we assume that K20 power efficiency is similar). The simulated SpMV RCAM power efficiency is in the similar low range of 0.3-0.4 GFLOP/s/W. The reason for low power efficiency lies in memristor write energy, which dominates RCAM energy consumption during arithmetic operations.

Figure 12 shows the GPU/CPU to RCAM computational complexity ratio $\frac{nnz_A}{nc_A+nr_A}$ as a function of $nnz_A$ (calculated for 2,740 matrices of the UFL Sparse Matrix Collection). It shows the trend of potential RCAM speedup (over GPU and CPU implementations) with the growing dataset size (the number of nonzero elements in a sparse matrix).

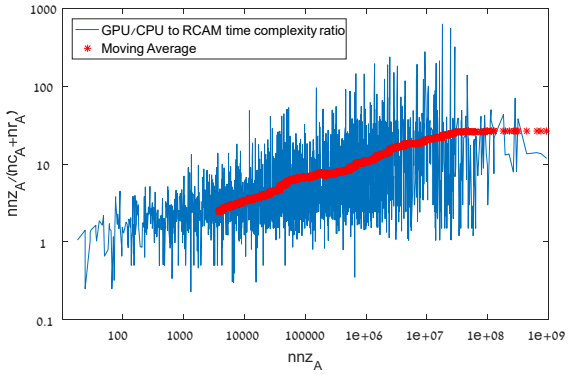


**Figure 12: GPU/CPU to RCAM complexity ratio $\frac{nnz_A}{nc_A+nr_A}$**

## 6.3 K-Means

Several evaluations are performed. We compare our RCAM processing-in-storage architecture with an FPGA [37][53], Multicore CPU [18], single GPU [9], and a 10-GPU cluster [54] *K*-means implementations. Table 1 provides a summary of the platforms and datasets. Figure 13 presents the average runtime per iteration (log scale) of each architecture, including the relative speedup and power efficiency of RCAM. Li *et al.* [37] showed a simplified MapReduce implementation on Xilinx ZC706 FPGA and about 2 million samples, each with 4 attributes. Their average time per iteration was 8.5ms, compared to 0.57ms on RCAM, yielding speedup of 15 and 1.5× better power efficiency. Ramanathan *et al.* [53] presented an implementation with work-stealing method of run-time load balancing on an Altera Stratix V FPGA. The total runtime is about 350ms with 16 iterations, while RCAM completes the same task in 75ms, resulting in a 4.6 speedup. The relatively small speedup is attributed mainly to the high clusters-to-samples ratio. The small dataset also leads to 2.1× lower power efficiency of ReCAM. More data samples may lead to lower FPGA performance and power efficiency, while keeping RCAM's the same. Ding *et al.* [18] used a high-end eight-core Intel i7-3770K CPU. We used their largest evaluated dataset for comparison, containing 2.5M samples, each with 68 dimensions. An iteration with 10,000 clusters has taken 432.9 seconds on the CPU on average, compared to 6.38 seconds on RCAM, leading to a 58.4× speedup and 20.8× higher power efficiency of RCAM. Bhimani *et al.* [9] presented GPU implementation of *K*-means, using NVIDIA K20M with 225W TDP and a 1164×1200 pixel RGB image. The total runtime in case of 240 clusters is 294 seconds in 166 iterations, while on RCAM it takes only 5 seconds, showing 58.4 speedup. Compared with 250W dissipated by a single RCAM chip, RCAM has 52.9× higher power efficiency. Rossbach *et al.* [54] used a ten NVIDIA Tesla K20M GPU cluster with a very large data set of 1 billion samples, occupying roughly 150GB. Their average iteration time is 30.6 seconds, compared with 0.65 seconds on RCAM, yielding speedup of 47. Each of the machines in the cluster has two Intel Xeon E5-2630 CPUs, which leads to a 415W TDP per machine. In total, the ten GPU cluster has 4.9× lower power efficiency than RCAM. The large speedup over the big dataset is attributed to the insensitivity of RCAM to dataset size, unlike the GPU cluster, which is limited by the communication bandwidth of each GPU.

**Table 1: *K*-Means compared datasets and platforms with RCAM.**

| Work Ref. | Platform | Dataset | | | |
|---|---|---|---|---|---|
| | | Samples | Attributes | Size on disk | Clusters |
| [53] | FPGA | 1M | 1 | 4MB | 128 |
| [37] | FPGA | 2M | 4 | 31.6MB | 4 |
| [18] | Intel i7 | 2.5M | 68 | 318.8MB | 10000 |
| [9] | GPU | 1.4M | 5 | 21.3MB | 240 |
| [54] | 10-GPU Cluster | 1B | 40 | 157.2GB | 120 |

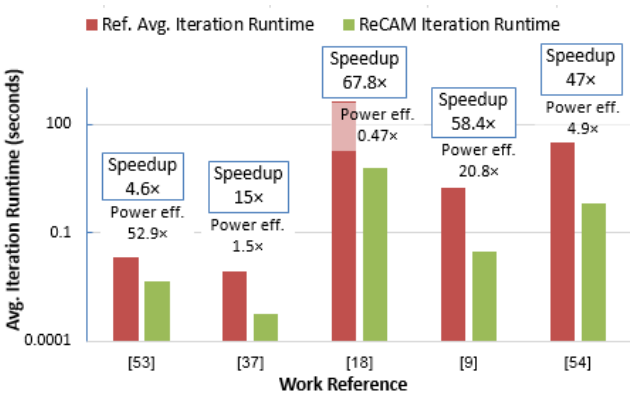


**Figure 13: Speedup and power efficiency of RCAM K-Means vs. existing solutions.**

## 6.4 KNN

We compare RCAM processing-in-storage implementation with FPGA [52], GPU [30] and Nearest neighbor Content Addressable Memory (NCAM) [35] *KNN* implementations. Table 2 summarizes the platforms and datasets used in these works. Figure 14 plots runtime (log scale) results, relative speedup and power efficiency of RCAM. Pu *et al.* [52] presented a FPGA implementation of *KNN* using Stratix IV 4SGX530 and the KDD-CUP 2004 quantum physics dataset, with 20,480 samples out of the total 50,000, each sample with 5 attributes. For *K*=20, runtime was 69ms. On RCAM, the runtime for the same dataset, regardless of the number of samples, is 2.3ms, resulting in speedup of 30 and 3× improved power efficiency.

Gutierez *et al.* [30] proposed a GPU-based *KNN* on NVIDIA K20M. They applied it to KDD-CUP 1999 dataset of 4.9 million instances, each having 41 attributes, and

achieved runtime of 33.9 sec. On RCAM, the same task completes in 1.9ms, showing speedup of 17,470 and improved power efficiency of 15.7k×.

Lee *et a.* [35] presented a NCAM processing-in-memory architecture, combining custom logic and HMC (hybrid memory cube). They compared NVIDIA Titan X GPU performance to NCAM on two image classification datasets, SIFT [40] and ImageNet [34]. The data samples are quite large (up to 16 KB). While the NCAM improves runtime over the GPU by an order of magnitude (17× and 5.2× on SIFT and ImageNet, respectively) and shows a significant benefit of a non-von Neumann concept, the RCAM processing-in-storage architecture demonstrates additional 1-2 orders of magnitude speedup (17.9 and 67.9, respectively) relative to NCAM, thanks to in-data rather than near-data processing. Compared with NCAM, ReCAM shows lower power efficiency of 0.9× and 0.2× for SIFT and AlexNet datasets, respectively. The lower power efficiency can be attributed to the highly specialized design of NCAM, which targets to accelerate KNN, in contrast to the larger scope of applications with high performance on RCAM.

**Table 2: KNN compared datasets and platforms with RCAM.**

| Work Ref. | Platform | Dataset | | | |
|---|---|---|---|---|---|
| | | Name | Samples | Attributes | *K* |
| [52] | FPGA | KDD-Cup 2004 | 20.5k | 5 | 240 |
| [30] | GPU | KDD-Cup 99 | 4.9M | 41 | 1000 |
| [35] | CPU | SIFT [40] | 1M | 128 | 16 |
| | NCAM | | | | |
| | GPU | AlexNet [34] | 1M | 4096 | 32 |
| | NCAM | | | | |

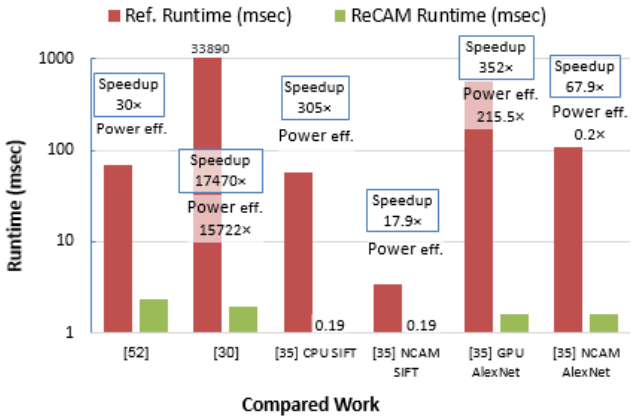


**Figure 14: Speedup and power efficiency of RCAM KNN vs. existing solutions.**

## 6.5 Smith-Waterman

The CUPS metric (Cell Updates per Second) is used to measure S-W performance. Performance results are compared to other works in Table 3. A four Xeon Phi implementation achieves 0.23 TCUPS [39]. A FPGA implementation of S-W reaches 6.0 TCUPS on the RIVYERA platform [60] having 128 Xilinx Spartan-6 LX150 FPGAs. A multi-GPU implementation reached 11.1 TCUPS on a cluster of 128 compute nodes with a total of 384 Tesla M2090 GPUs [16]. On RCAM with a total of 8GB in 32 separate ICs, each 256MB and 8M rows, we demonstrate 53 TCUPS, computing a total of $57.2\times10^{12}$ scores, achieving 4.7 times higher throughput than the GPU version. Table 3 also shows computed GCUPS/Watt ratios; RCAM is close to twice better power efficiency than the FPGA solution and 80× better than the GPU system.

**Table 3: Summary of state-of-the-art performance for S-W scoring step in previous works and in RCAM.**

| Accelerator | Xeon Phi | FPGA | GPU | RCAM |
|---|---|---|---|---|
| Performance (TCUPS) | 0.23 | 6.0 | 11.1 | **53** |
| Number of ICs | 4 | 128 | 384 | 32 |
| Power (kW) | 0.8 | 1.3 | 100.0 | 6.6 |
| GCUPS/W | 0.3 | 4.7 | 0.1 | 8.0 |
| Reference | [39] | [60] | [16] | |

## 7. Conclusions

Near-data processing-in-storage is inherently limited because it is based on replicating von Neumann processors near storage. Therefore, it potentially faces some of von Neumann architecture problems, such as the bandwidth wall. To resolve this problem and allow for full utilization of ultra-high internal bandwidth of future resistive memory based SSD, we propose a novel in-data processing-in-storage architecture based on Resistive Content Addressable Memory (RCAM). Unlike near-data in-SSD processing, RCAM enables storage with *in-data* associative processing capabilities. It can contain hundreds of millions of data rows, each row serving as an associative processing unit. RCAM requires no in-storage processing cores external to the storage arrays. There is no data transfer outside the storage arrays. Therefore, the internal bandwidth of the resistive memory based storage can be utilized to its fullest extent, considerably improving computation throughput of processing-in-storage system.

The RCAM architecture, capable of general purpose associative processing, has been applied to a variety of challenging data and compute intensive problems, such as various machine learning and bioinformatics algorithms. The paper investigated SpMV, *K*-Means, *KNN* and Smith-Waterman sequence alignment algorithms and compared RCAM to other published analyses.

## 8. REFERENCES


[1] Ahn, J., Hong, S. , Yoo, S., Mutlu, O., and Choi, K. , "A scalable processing-in-memory accelerator for parallel graph processing," Computer Architecture (ISCA), ACM/IEEE 42nd Annual International Symposium on, , pp. 105–117 2015.

[2] Ahn, J., Yoo, S., Mutlu, O., & Choi, K. "PIM-enabled Instructions: A Low-overhead, Locality-aware Processing-in-memory Architecture." Computer Architecture (ISCA) ACM/IEEE 42nd Annual International Symposium on (pp. 336-348) , 2015.

[3] Akin, B., Franchetti, F., & Hoe, J. C.. "HAMLeT architecture for parallel data reorganization in memory". IEEE Micro, 36(1), 14-23, 2016.

[4] Akinaga, H., and Hisashi Shima. "Resistive random access memory (RRAM) based on metal oxides." Proceedings of the IEEE 98.12: 2237-2251, 2010.

[5] Alibart, F., T. Sherwood, D. Strukov. "Hybrid CMOS/nanodevice circuits for high throughput pattern matching applications", IEEE Conference on Adaptive Hardware and Systems, 2011.

[6] Azarkhish, E., Pfister, C., Rossi, D., Loi, I. and Benini, L. "Logic-Base Interconnect Design for Near Memory Computing in the Smart Memory Cube". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(1), 210-223, 2017.

[7] Bae, D.-H., J.-H. Kim, S.-W. Kim, H. Oh, and C. Park, "Intelligent SSD: a turbo for big data mining," in Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, pp. 1573–1576, ACM, 2013.

[8] Balasubramonian, R., Chang, J., Manning, T., Moreno, J. H., Murphy, R., Nair, R., & Swanson, S. "Near-data processing: Insights from a MICRO-46 workshop", IEEE Micro, 34(4), 36-42, 2013.

[9] Bhimani, J., M. Leeser, and N. Mi. "Accelerating K-Means clustering with parallel implementations and GPU computing." High Performance Extreme Computing Conference (HPEC), 2015.

[10] Boboila, S., Kim, Y., Vazhkudai, S. S., Desnoyers, P., & Shipman, G. M. (2012, April). Active flash: Out-of-core data analytics on flash storage. In Mass Storage Systems and Technologies (MSST), IEEE 28th Symposium on (pp. 1-12), 2012.

[11] Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., and Xie, Y., "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," Proceedings of the 43rd International Symposium on Computer Architecture, pp. 27–39, IEEE Press, 2016.

[12] Cho, B. Y., W. S. Jeong, D. Oh and W. W. Ro, "Xsd: Accelerating mapreduce by harnessing the gpu inside an ssd," Proceedings of the 1st Workshop on Near-Data Processing, 2013.

[13] Cho, S., Park, C., Oh, H., Kim, S., Yi, Y., and Ganger, G. R. "Active disk meets flash: A case for intelligent ssds," in Proceedings of the 27th international ACM conference on International conference on supercomputing, pp. 91–102, , 2013.

[14] Coburn, J., Bunker, T., Schwarz, M., Gupta, R., & Swanson, S. From ARIES to MARS: Transaction support for next-generation, solid-state drives. In Proceedings of the twenty-fourth ACM symposium on operating systems principles, pp. 197-212, 2013.

[15] Davis, T., Hu, Y., "The University of Florida sparse matrix collection," ACM Transactions on Mathematical Software (TOMS), 38, no. 1, 2011.

[16] de Oliveira Sandes, E. F., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., & Melo, A. C. M. CUDAlign 4.0: Incremental Speculative Traceback for Exact Chromosome-Wide Alignment in GPU Clusters. IEEE Transactions on Parallel and Distributed Systems, 27(10), 2838-2850, 2016.

[17] De, A., M. Gokhale, R. Gupta, and S. Swanson, "Minerva: Accelerating data analysis in next-generation ssds," in Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on, pp. 9–16, IEEE, 2013.

[18] Ding, Y. , Zhao, Y., Shen, X., Musuvathi, M., and Mytkowicz, T. "Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup". In Proceedings of the 32nd International Conference on Machine Learning, ICML,pp. 579–587, 2015.

[19] Dorrance, R., Ren, F., and Marković, D. "A scalable sparse matrix-vector multiplication kernel for energy-efficient sparse-blas on FPGAs". In Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays, pp. 161-170, 2014.

[20] Duck-Ho, B., Jin-Hyung, K., Yong-Yeon, J., Sang-Wook, K., Hyunok, O., & Chanik, P. Intelligent SSD: A turbo for big data mining. Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, 2013, pp. 1573-1576, 2016.

[21] Eshraghian, K., Cho, K. R., Kavehei, O., Kang, S. K., Abbott, D., & Kang, S. M. S.. Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 19(8), 1407-1417, 2011.

[22] Farmahini-Farahani, A., Ahn, J. H., Morrow, K., & Kim, N. S. "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules." IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 283-295, 2015.

[23] Fengwei, A. N., Tetsushi Koide, and Hans Juergen Mattausch. "A K-means-based multi-prototype high-speed learning system with FPGA-implemented coprocessor for 1-NN searching." IEICE TRANSACTIONS on Information and Systems 95, no. 9, pp. 2327-2338, 2015.

[24] Foster, C., "Content Addressable Parallel Processors", Van Nostrand Reinhold Company, NY, 1976.

[25] Gao, M., and Kozyrakis, C. (2016, March). "HRL: efficient and flexible reconfigurable logic for near-data processing." IEEE International Symposium on High Performance Computer Architecture (HPCA), (pp. 126-137).

[26] Gotoh, O. "An improved algorithm for matching biological sequences." Journal of molecular biology 162.3, pp. 705-708, 1982.

[27] Guo, Q., Guo, X., Bai, Y., and Ipek, E. A resistive TCAM accelerator for data-intensive computing. Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 339-350, 2011.

[28] Guo, Q., Guo, X., Bai, Y., Patel, R., Ipek, E., and Friedman, E. G. "Resistive ternary content addressable memory systems for data-intensive computing." IEEE Micro, vol. 35, no. 5, pp. 62-71, 2015.

[29] Guo, Q., Guo, X., Patel, R., Ipek, E., & Friedman, E. G. "AC-DIMM: associative computing with STT-MRAM". ACM SIGARCH Computer Architecture News, vol. 41, no. 3, pp. 189-200, 2013.

[30] Gutiérrez, P. D., Lastra, M., Bacardit, J., Benítez, J. M., & Herrera, F. "GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs." Information Sciences, 373, 165-182, 2016.

[31] Jo, Y. Y., Cho, S., Kim, S. W., & Oh, H. "Collaborative processing of data-intensive algorithms with CPU, intelligent SSD, and GPU." Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1865-1870, 2016.

[32] Jun, S. W., Liu, M., Lee, S., Hicks, J., Ankcorn, J., King, M., Xu, S, Arvind, "BlueDBM: an appliance for big data analytics", ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), 2015.

[33] Kang, Y., Kee, Y. S., Miller, E. L., & Park, C. (2013, May). Enabling cost-effective data processing with smart SSD. IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1-12, 2013.

[34] Krizhevsky, A., I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems 25 (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[35] Lee, V. T., del Mundo, C. C., Alaghi, A., Ceze, L., Oskin, M., & Farhadi, A. (2016). NCAM: Near-Data Processing for Nearest Neighbor Search. arXiv preprint arXiv:1606.03742.

[36] Li, J., Montoye, R. K., Ishii, M., & Chang, L. 1 Mb 0.41 μm² 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing. IEEE Journal of Solid-State Circuits, 49(4), 896-907, 2014.

[37] Li, Z., Jifang J., and Lingli W. "High-performance K-means Implementation based on a Coarse-grained Map-Reduce Architecture." arXiv preprint arXiv:1610.05601, 2016.

[38] Liu, T. Y., Yan, T. H., Scheuerlein, R., Chen, Y., Lee, J. K., Balakrishnan, G. and Sasaki, T. A 130.7mm$^2$ 32-Gb ReRAM Memory Device in 24-nm Technology. IEEE Journal of Solid-State Circuits, 49(1), 140-153, 2014.

[39] Liu, Y. and Schmidt, B.. SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors. In IEEE ASAP. pp. 184–185, 2014.

[40] Lowe, D. G. "Distinctive Image Features from Scale-Invariant Keypoints," Int. J. Comput. Vision, vol. 60, pp. 91–110, Nov. 2004.

[41] Matsunaga, S., Hiyama, K, Matsumoto, A, Ikeda, S, Hasegawa, H, Miura, K, Hayakawa, J., Endoh, T., Ohno, H., and Hanyu, T. "Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices." Applied Physics Express 2, no. 2, 2009.

[42] Matsunaga, S., Katsumata, A., Natsui, M., Fukami, S., Endoh, T., Ohno, H., and Hanyu, T., "Fully Parallel 6T-2MTJ Nonvolatile TCAM with Single-Transistor-

Based Self Match-Line Discharge Control," Symposium on VLSI Circuits Digest of Technical Papers, pp. 298-299, 2011.

[43] Mingyu, G., Ayers, G., and Kozyrakis, C. "Practical near-data processing for in-memory analytics frameworks." International Conference on Parallel Architecture and Compilation (PACT), 2015.

[44] Nair, R., Antao, S.F., Bertolli, C., Bose, P., Brunheroto, J.R., Chen, T., Cher, C.Y., Costa, C.H., Doi, J., Evangelinos, C. and Fleischer, B.M. "Active memory cube: A processing-in-memory architecture for exascale systems. IBM Journal of Research and Development," 59(2/3), pp.17-1, 2015.

[45] Nickel, J., "Memristor Materials Engineering: From Flash Replacement Towards a Universal Memory," Proceedings of the IEEE International Electron Devices Meeting, 2011.

[46] Omitted for blind review

[47] Omitted for blind review.

[48] Omitted for blind review.

[49] Park, K., Kee, Y. S., Patel, J. M., Do, J., Park, C., & Dewitt, D. J. "Query Processing on Smart SSDs". IEEE Data Eng. Bull., 37(2), 19-26, 2014.

[50] Paul, S., & Bhunia, S. A scalable memory-based reconfigurable computing framework for nanoscale crossbar. IEEE transactions on Nanotechnology, 11(3), 451-462, 2012.

[51] Potter J., and Meilander, W. "Array processor supercomputers," Proceedings of the IEEE, vol. 77, no. 12, pp. 1896–1914, 1989.

[52] Pu, Y., Peng, J., Huang, L., & Chen, J. (2015, May). An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 167-170, 2015.

[53] Ramanathan, N., Wickerson, J., Winterstein, F., & Constantinides, G. A. A case for work-stealing on fpgas with opencl atomics. Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 48-53), 2016.

[54] Rossbach, C. J., Yuan Yu, Jon, C., Jean-Philippe M., and Dennis F. "Dandelion: a compiler and runtime for heterogeneous systems." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 49-68. 2013.

[55] Saule, E., Kaya, K., & Çatalyürek, Ü. V. Performance evaluation of sparse matrix multiplication kernels on intel xeon phi. In International Conference on Parallel Processing and Applied Mathematics (pp. 559-570), 2013.

[56] Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S. and Srikumar, V., ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In Proceedings of the 43rd International Symposium on Computer Architecture (pp. 14-26)., 2016.

[57] Smith, T. F., and Waterman M. S. "Identification of common molecular subsequences." Journal of molecular biology 147.1 pp. 195-197, 1981.

[58] Sura, Z., Jacob, A., Chen, T., Rosenburg, B., Sallenave, O., Bertolli, C., Antao, S., Brunheroto, J., Park, Y., O'Brien, K. and Nair, R., May. Data access optimization in a processing-in-memory system. In Proceedings of the 12th ACM International Conference on Computing Frontiers, p. 6, 2015.

[59] Torrezan, A. C., Strachan, J. P., Medeiros-Ribeiro, G., & Williams, R. S. "Sub-nanosecond switching of a tantalum oxide memristor." Nanotechnology, vol. 22 no. 48, 2011.

[60] Wienbrandt., L."The FPGA-based High-Performance Computer RIVYERA for Applications in Bioinformatics." Language, Life, Limits: 10th CiE, pp. 383-392. Springer 2014.

[61] Xu, W., Zhang, T., Chen, Y., "Design of spin-torque transfer magnetoresistive RAM and CAM/TCAM with high sensing and search speed", IEEE Transactions VLSI Systems, 18.1 pp. 66-74, 2010.

[62] Yang, J., Strukov, D., and Stewart, D. "Memristive devices for computing." Nature nanotechnology, vol. 8, no. 1, 13-24, 2013.

[63] Yavits L, Morad, A., & Ginosar, R. "The effect of communication and synchronization on Amdahl's law in multicore systems". Parallel Computing, vol. 40, no. 1, pp. 1-16, 2014.

[64] Zhang, D., Jayasena, N., Lyashevsky, A., Greathouse, J. L., Xu, L., & Ignatowski, M. TOP-PIM: Throughput-oriented programmable processing in memory. Proceedings of the 23rd international symposium on High-performance parallel and distributed computing, pp. 85-98, 2014.