PRINS: Processing-in-Storage Acceleration of Machine Learning

Roman Kaplan, Leonid Yavits and Ran Ginosar

***Abstract*—Machine learning algorithms have become a major tool in various applications. The high performance requirements on large-scale datasets pose a challenge for traditional von Neumann architectures.**

We present two machine learning implementations and evaluations on PRINS, a novel processing-in-storage system based on Resistive Content Addressable Memory (ReCAM). PRINS functions simultaneously as a storage and a massively parallel associative processor. PRINS processing-in-storage resolves the bandwidth wall faced by near-data von Neumann architectures, such as 3D DRAM and CPU stack or SSD with embedded CPU, by keeping the computing inside the storage arrays, thus implementing in-data, rather than near-data, processing. We show that PRINS based processing-in-storage architecture may outperform existing in-storage designs and accelerator based designs. Multiple performance comparisons for the ReCAM processing-in-storage implementations of *K*-means and K-nearest neighbors are performed. Compared platforms include CPU, GPU, FPGA and Automata Processor. We show that PRINS may achieve an order-of-magnitude speedup and improved power efficiency relative to all compared platforms.

***Index Terms*— Near-data Processing; Associative Processing; Processing-in-storage; Processing-in-Memory; RRAM; CAM; Memristors.**

I. Introduction

Machine learning algorithms have become a ubiquitous tool in wide range of large-scale applications, from data analytics to pharmaceutical research. The scale and performance requirements of these applications makes machine learning algorithms a prime candidate for accelerator research. Most accelerators are von Neumann machines, which performance is severely limited by memory bandwidth. This limitation is especially acute in machine learning algorithms which are typically data intensive. One approach to mitigate the bandwidth constraint is to bring the processing units closer to the data. That approach is known as *near-data processing* (NDP) [1].

The premise of NDP is reducing memory transfer time by cutting the physical distance and increasing the bandwidth between the processing units and memory. However, NDP architectures such as 3D DRAM and CPU stacks, or SSD with embedded CPU, are still inherently limited because they are based on replicating the von Neumann architecture in memory or storage. Since its inception, NDP has mainly meant processing-in-memory (PiM). More recent approach is to exploit processing elements closer to storage, i.e., near-storage processing. However, both PiM and near-storage processing still suffer from the von Neumann bandwidth bottleneck. PiM has a bottleneck between storage and main memory. Near-storage processing suffers from the bottleneck between the storage chips and processing units. Both approaches are inherently limited because they are largely based on the von Neumann architecture model.

This work presents PRINS, a novel Processing IN Storage architecture that employs a Resistive CAM (ReCAM, [2][3]), and applies it to acceleration of machine learning tasks. PRINS simultaneously functions as a 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. The scalability of the system makes it suitable for storing and *in-situ* processing of high volume data intensive applications such as machine learning. The entire dataset fits in PRINS storage. Furthermore, there is no other storage and PRINS storage is large enough to contain it.

This paper makes the following contributions:

- We present the PRINS architecture and show how it can function as a scalable storage with in-data processing capabilities.
- We develop and evaluate a PRINS based implementation for two machine learning algorithms: *K*-means and KNN.
- We show that PRINS implementations can outperform near-data or other implementations in both performance and power efficiency. Compared platforms include CPU, GPU, FPGA and an extended Automata Processor [4], [5].

The rest of this paper is organized as follows. Section II reviews the background and previous work. Section III presents the PRINS architecture. Sections IV and V describe the ReCAM implementations of *K*-means and KNN, respectively, including performance and power efficiency comparisons. Section VI offers conclusions.

Manuscript received July 31, 2017; revised December 8, 2017; accepted January 20, 2018.

The authors are with the Electrical Engineering Department, Technion, Israel Institute of Technology (email: sromanka@campus.technion.ac.il; leonid.yavits@nububbles.com; ran@ee.technion.ac.il).

II. Background and Previous Work

The following review considers previous work in the areas of processing in memory (PiM) addressing the memory bandwidth and latency limitations, processing near storage and implementations of PiM using resistive logic.

A. Processing in Memory

Memory access latency in conventional computer architectures is often orders of magnitudes longer than processing time, and memory bandwidth is similarly limited. One approach to bridging this gap is processing-in-memory (PiM) [6]–[9], which places memory and processing on the same integrated circuit. Although PiM improves memory latency and bandwidth, the approach is limited by the size of memory and logic that can be integrated on a single chip.

Monolithic 3D integrated circuits [10], together with recent advances in 3D memory (e.g., Hybrid Memory Cube [11]) may help to improve the memory and logic density and to relax the memory latency and bandwidth obstacles. However, separate memory and logic 3D stacks yield near-memory processing rather than PiM.

B. Processing Near Storage

More recently, with the introduction of Flash SSD, near-data processing-in-storage, or near-storage-processing (NSP), research has begun to emerge. In general, this approach either exploits existing processing elements within the storage device (i.e., controller) [12]–[15] or places additional processing elements. The additional processing elements can vary from a multi-core CPU [14], to GPU [16] and FPGA [17], [18]. All of the above works attempt to exploit the higher inner bandwidth of the SSD. However, the bandwidth wall still exists, because the bandwidth of flash arrays is limited, ranging from a few hundred MB/s to a few GB/s, depending on the number of parallel flash channels [19].

C. PiM Using Resistive Logic

Emerging resistive memory technologies have become a focus of PiM research. A 3D monolithic device [20], including a storage layer that consists of more than one million resistive RAM cells, allows for *in-situ* processing as the logic cells are placed in proximity of the storage cells. Another work [21] has demonstrated a test chip of 32Gb device in 24nm technology with two ReRAM-based memory layers and a CMOS logic layer underneath, showing design techniques to achieve a high density functional chip.

Other works aim to exploit the resistive devices as logic. The stateful IMPLY logic [22] employs one resistor and two memristors can be used as input and the state of one of the memristor stores the output. The Memristor Aided Logic (MAGIC) method [23][24] offers additional basic logic functions, implemented without the resistor and with a third memristor to store the output. Li *at al.* [25] experimentally demonstrated a similar approach by implementing 16 Boolean logic functions with memristor NAND logic.

Complete PiM architectures based on resistive materials were also proposed. Somnath *et al*. [26] developed MBARC, a memory-based architecture for reconfigurable computing, combining a resistive crossbar architecture with memory partitions serving as lookup tables with CMOS logic for pre-evaluation. Chi *et al*. [27] introduced PRIME, a PIM accelerator of neural network applications in RRAM based main memory. Shafiee *et al.* [28] developed an *in-situ* processing architecture, where memristor crossbar arrays are used to perform dot-product operations in an analog manner. Guo *et al*. [29]–[31] suggested using resistive and STT-MRAM ternary CAM for data intensive computing. Their works used the associative capabilities of CAM and ternary CAM mainly for search operations, while 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.

D. Our Contribution

In this paper, we demonstrate *K*-means and KNN high performance and power-efficient execution on PRINS, a massively parallel non-von Neumann processing-in-storage architecture. This architecture is different from previous suggestions mainly by its use of associative, rather than logic-based, processing. The inherent parallel performance of the resistive memory array can be utilized to the full extent, enabling very high computation throughput while reducing energy consumption (mainly due to reduced data movement within the device).

III. PRINS Architecture

Resistive memories store information by modulating the resistance of nanoscale storage elements. The metal-oxide resistive random access memory (ReRAM) is considered a potential technology to replace next-generation nonvolatile memories [32]. Recent advances in the technology have demonstrated fast (about 0.3ns) and low energy (about 0.1pJ) [33] switching. While most ReRAM-based designs employ one transistor and one memristor (1T1R) cells, ReCAM uses 2T2R [37] or 2R [38] cells and appropriate peripheral circuits [38] to support associative storage and processing.

A. ReCAM Crossbar Array

The ReCAM module (Figure 1) comprises a ReCAM array, where each memory row is also a baseline processing unit (PU), and a peripheral circuitry. The basic cell contains two memristors, holding complementary values R and $\bar{R}$, and two selector transistors T_1 and T_2 (Figure 1a). The peripheral circuitry includes key and mask registers, TAG logic (Figure 1b), a reduction tree (Figure 1c) and a daisy-chain interconnect of the TAG line.

ReCAM is a scalable 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 '0') and a maximum resistance R_{OFF} (high resistive state, logic '1').

The *KEY* register (Figure 1) contains a key data word to be written or compared against. The *MASK* register defines the

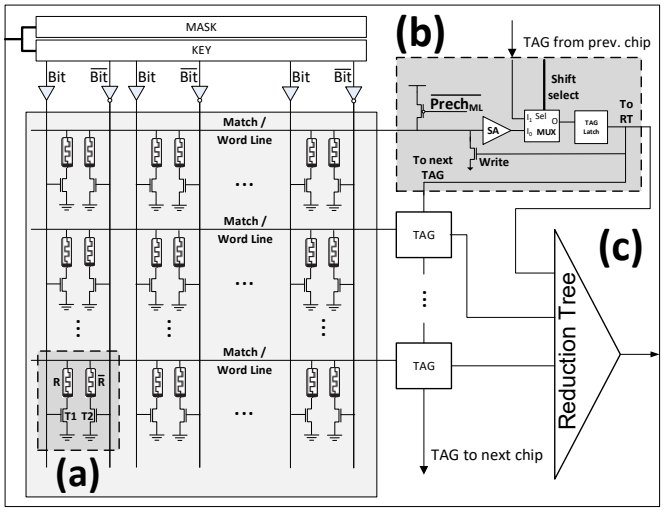

Figure 1: ReCAM module. (a) 2T2R ReCAM bitcell. (b) TAG logic. (c) Reduction tree

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 through their TAGs, all PUs in parallel. The reduction tree enables counting of active tag bits by logarithmic summation. This operation is useful whenever a vector needs to be reduced to a scalar.

The ReCAM compare operation is implemented as follows. The Match/Word lines are pre-charged and the key is asserted to the Bitlines and the inverse key is asserted to the Bit-not lines. In the columns that are ignored during comparison, the Bit and Bit-not lines are kept at '0'. If all unmasked bits in a row match the key (*i.e.*, when Bit line '1' is applied to an R_{OFF} memristor and Bit-not line '0' is applied to an R_{ON} memristor, or vice versa), the Match/Word line remains high and '1' is sampled into the corresponding TAG bit. If at least one unmasked bit is mismatched, the Match/Word line discharges through an R_{ON} memristor and an open selector transistor. A logic '0' is then sampled into the TAG.

The write operation is performed in two phases. First, the $V \geq V_{ON}$ voltage (where V_{ON} is a threshold voltage required to switch memristors to the "on" state) is applied to all tagged Match/Word lines, and applicable Bit lines (for writing '0's) and Bit-not lines (for complementing '1's) are set at '1'. Second, the $V \leq V_{OFF}$ voltage (where V_{OFF} is a threshold voltage to switch memristors to the "off" state) is applied to all tagged Match/Word lines, and applicable Bit-not lines (for complementing the '0's) and Bit lines (for writing '1's) are set at '1'.

Memristor sub-nanosecond switching time [33] allows GHz ReCAM processing-in-storage operation. The energy consumption during compare may be less than 1fJ per bit [38]. The write energy is in the range of tens of fJ [33], which may be prohibitively high for simultaneous parallel writing of multi-bit words in the entire ReCAM storage. Another factor which potentially limits ReCAM processing-in-storage system is endurance (the number of program/write cycles that can be applied to a memristor before it becomes unreliable). Recent endurance figures vary between 10^{10} [34] and 10^{12} cycles [35], [36], which may suffice for only about one month. However, the area is still actively researched. Moreover, with some advanced wear-leveling techniques similar to those presented in [39][40], we believe ReCAM processing-in-storage system life-time can be extended to a number of years.

Table 1: 32-bit arithmetic operations on PRINS

Instruction	Cycles
Shift down one row	192
B ← A + B	256
C ← A + B	512
Row-wise Max (A, B)	64
Max Scalar (A)	64

B. Associative Processing with the ReCAM Crossbar Array

In a conventional CAM, compare operation is typically followed by a read of the matched data word. In associative processing a compare is usually followed by a parallel write of the result value into the unmasked bits of all tagged rows. Additional capabilities, such as read and reduction operations, are included [38][2].

Any computational expression can be efficiently implemented in ReCAM storage using line-by-line execution of the truth table of the expression [38]. Arithmetic operations are typically performed bit-serially. Table 1 lists several operations supported by a ReCAM crossbar array for 32-bit operands and the number of cycles required per each operation. Shifting down a consecutive block of rows by one row position requires six cycles per bit: First, compare-to-'1' copies the source bit-column of all rows into the TAG. Second, shift moves the TAG vector down by setting the shift-select line (Figure 1b). Third, write-'1' copies the shifted TAG to the same bit-column. The fourth-to-sixth cycles repeat the process for '0' values. Shifting 32-bit numbers thus requires 192 cycles. Addition (in-place or not) is performed in a bit-serial manner using a truth table approach [38] (32 bits times 8 truth-table rows times 2 for compare and write amount to 512 cycles). Row-wise maximum compares in parallel two 32-bit numbers in each row. Max Scalar tags all rows that contain the maximal value in the selected element. Additional operations, such as parallel and reduction arithmetic, may be required for other algorithms. Floating point arithmetic operations are also supported. Single precision floating point multiplication requires 4,400 cycles [38].

C. System Architecture

Conceptually, ReCAM may comprise hundreds of millions of rows, each serving as a processing unit (PU). The entire array may be divided into multiple smaller ICs (due to power per die restrictions, Figure 2a). A row is fully contained within a single IC.

The PRINS processing-in-storage system uses a

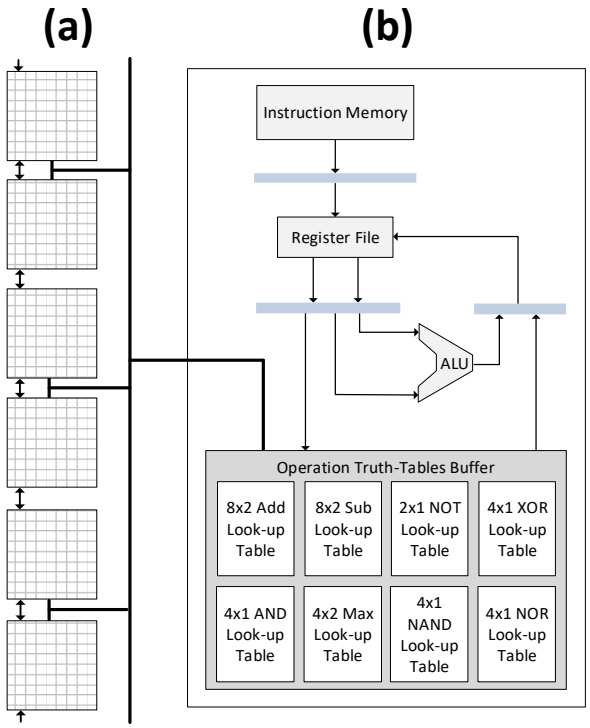

Figure 2: A PRINS system, composed of (a) separate multiple ReCAM modules and (b) a microcontroller.

microcontroller (Figure 2b). It issues instructions, sets the key and mask registers, handles control sequences and executes read requests. The microcontroller may also perform some baseline processing, such as normalization of the reduction tree results. PRINS software, including both associative operations 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 becomes worse. As a result, the performance of processing-in-storage system may saturate or even diminish.

ReCAM processing-in-storage provides 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 the data is never transferred outside the storage arrays through a bandwidth-limited communication interface, the performance limit is pushed further away.

D. Programming and Evaluation Parameters

ReCAM relies on the host to transfer the code, execution parameters (e.g., dataset addresses) and invoke execution. The code intended to run on ReCAM is translated into associative primitives. Presently, ReCAM code is manually encoded at assembly language level.

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

Table 2: Simulated PRINS parameters used for performance and power efficiency evaluations.

ReCAM Parameter	Value
Technology	28nm
Frequency	500MHz
Single row compare energy	1fJ
Single bit write energy	100jJ
Bits per row	256
Single module memory size	256MB

The following two sections present two algorithm implementations and evaluations on ReCAM for performance and power efficiency. Memristor latency and energy figures are obtained using intensive SPICE simulations based on the TEAM model [41]. ReCAM energy and timing figures were obtained using full SPICE simulations of [38] for a single cell and a row, an incorporating the above results for the memristor. ReCAM parameters used for the algorithms performance and energy evaluations are listed in Table 2. Full PRINS array performance and power simulations were conducted using our own cycle-accurate simulator (similar to the one introduced in[38]) for each dataset in the paper.

IV. *K*-MEANS PRINS IMPLEMENTATION

K-means is an unsupervised machine 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 center (mean). Every sample usually consists of multiple attributes (dimensions). The algorithm performs several iterations until reaching convergence, modifying the cluster center coordinates and sample cluster assignments in each iteration.

A. In-Storage K-Means Algorithm

The *K*-means algorithm pseudocode, as implemented in storage, is presented in Figure 3. The algorithm minimizes the Euclidean distances between the samples and the cluster means. 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. During execution, the algorithm consists of two *K*-iteration loops, *assignment* and *update*.

The *assignment* assigns each sample with a cluster. It does so by assigning each sample to the cluster whose mean yields the minimal Euclidean distance. In each iteration of lines 3-6 of Figure 3, the distance over a single attribute is associatively calculated in parallel for all dataset samples. Next, in lines 7-9, after all attribute distances were summed, the minimal Euclidean distance and the cluster assignment are updated in

parallel for all dataset samples. Note that lines 2-9 are always executed in parallel on the entire storage, in a SIMD-like style.

The *update* loop recalculates the new mean coordinates. First, 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 (line 13), followed by counting the number of samples assigned to the mean (line 14), both using the reduction tree. Finally, the new mean coordinates are calculated by the microcontroller (line 15).

The *assignment* and *update* loops may be repeated until the mean coordinates convergence. Note that the key computational steps are parallelized, and in the *update* loop parallelism (over all samples of the same cluster) is made possible by associativity. Convergence is checked at the microcontroller by comparing the mean coordinates of each two consecutive *assignment* iterations, then checking if the difference exceeds the predefined threshold. If all mean coordinates change within the threshold, the execution ends.

B. K-Means Performance Evaluations

Several evaluations are performed. We compare our PRINS implementation with a multicore CPU [42], FPGA [43], [44], single- [45] and 10-GPU cluster [46] *K*-means implementations. Table 4 lists all compared datasets from each work. Table 3 presents the average runtime per iteration of each architecture, PRINS simulated runtime, relative speedup and power efficiency ratio of PRINS relative to the other architectures (based on specified power figures for each architecture).

Ding *et al.* [42] used a high-end eight-core Intel i7-3770K CPU. We used two of their largest evaluated datasets for comparison, the first containing 2.5M samples (*US Census Data* in Table 4) and 68 attributes. The second contains 1M samples and 384 attributes. The attributes were extracted from "tiny" images (30×30 pixels each), hence the name *Tiny*. Li *et al.* [43] showed a simplified MapReduce implementation on Xilinx ZC706 FPGA and about 2 million samples and 4 attributes. Ramanathan *et al.* [44] presented an implementation with work-stealing method of run-time load balancing on an Altera Stratix V FPGA. Bhimani *et al*. [45] presented GPU implementation of *K*-means, using NVIDIA K20M and executing on a 1164×1200 pixel RGB image. Each pixel has five attributes, three for color and two for coordinates. Rossbach *et al.* [46] used a ten-node cluster. Every node with a NVIDIA Tesla K20M GPU and two Intel Xeon E5-2620 CPU, which dissipate total of 225W+2×95W=415W. Evaluations were performed on a very large data set of 1 billion samples and 40 dimensions, occupying roughly 150GB. Their average iteration time is 49.4 seconds, compared with simulated 0.16 seconds on PRINS, yielding a speedup of 302. The large speedup over the big dataset is attributed to the insensitivity of PRINS to dataset size, unlike the GPU cluster, which is limited by the communication bandwidth of each GPU.

Table 4: *K*-Means datasets used for performance comparisons.

Work Ref.	Dataset				Clusters (K)
	Name	Samples	Attributes (Dimensions)	Size on disk	
[42]	US Census Data [52]	2.5M	68	318.8MB	10000
	Tiny	1M	384	384MB	10000
[43]	Electric Power [52]	2M	4	31.6MB	4
[44]	N/A (synthetic)	~1M	1	4MB	128
[45]	N/A (1.4MP RGB Image)	1.4M	5	21.3MB	240
[46]	N/A (synthetic)	1B	40	157.2GB	120

V. K-Nearest Neighbors PRINS Implementation

K-nearest neighbors (KNN) is another common machine learning algorithm frequently used for classification. The algorithm computes the distances between an (unclassified) input query sample and a dataset of classified samples. Each sample consists of multiple attributes and also stores its class. 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, as the two latter options are less computationally

Table 3: K-Means compared performance to different architectures.

Platform	Work Ref.	Device/s	TDP (W)	Dataset Name	Avg. Time per Iteration (sec)	PRINS Time per Iteration (sec)	PRINS Average Power (W)	PRINS Speedup	PRINS Power Efficiency Ratio
CPU	[42]	Intel i7-3770K	77	US Census Data	76	6	63.7	12.6×	15.2×
				Tiny	48.5	9.3	81.3	5.2×	5×
FPGA	[43]	Xilinx ZC706	25	Electric Power	8.5 msec	0.55 msec	7.3	15.5×	52.6×
	[44]	Altera Stratix V	25	N/A (synthetic)	22 msec	4.5 msec	4	4.9×	30.4×
GPU	[45]	NVIDIA K20M	225	1.4MP RGB Image	1.77	43 msec	5	86.2×	1900×
	[46]	Cluster of ten nodes	4150	N/A (synthetic)	49.4	0.16	27.7kW	302×	45×

Algorithm 2 KNN Implementation in PRINS

//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 $attr \in \{sample\ attributes\}$:
Do-all $x \in X$: //all samples in parallel
2: $dist_{attr} \leftarrow Query_{attr} - x_{attr}$
3: $sqDist_{attr} \leftarrow (dist_{attr})^2$
4: $sqDist \leftarrow sqDist_{partial} + sqDist_{attr}$

//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 4: KNN Pseudocode.

demanding. For example, the Automata Processor (compared to PRINS in subsection VB) does not have an arithmetic logic unit and is therefore using a variation of Hamming distance for efficient implementation of KNN.

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.

A. In-Storage K-Nearest Neighbors Algorithm

KNN algorithm pseudocode on PRINS is presented in Figure 4. 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 4), 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. Adapting the implementation for a different distance calculation (e.g. Hamming) can be done by changing the computation in lines 2-4 appropriately.

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. 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 microcontroller). Overall the loop is iterated K times.

Table 5: KNN compared datasets and platforms with PRINS.

Work Ref.	Dataset				K
	Name	Samples	Attributes	Size on Disk	
[47]	KDD-Cup 2004	20.5k	64	5.2MB	20
[48]	KDD-Cup 99	4.9M	42	800MB	1000
[5]	WordEmbed [53]	~1M	64	8MB	2
	SIFT [54]		128	16MB	4
	TagSpace [55]		256	32MB	16

B. K-Nearest Neighbors Performance Evaluations

We compare our PRINS implementation with FPGA [43][5], GPU [48][5], multicore CPU [5], and an Automata Processor [5] KNN implementations. Table 5 lists the compared datasets from each work. Table 6 shows the runtime results and of each compared platform, the simulated runtime results of PRINS, and the relative speedup and power efficiency of PRINS relative to the other architectures. In [47], Pu *et al*. presented a FPGA implementation of KNN using Stratix IV 4SGX530 and the KDD-CUP 2004 quantum physics dataset, with 20,480 samples and 64 attributes. For K=20, runtime was 69ms. On PRINS, the runtime for the same dataset, regardless of the number of samples, is 1.1ms, resulting in speedup of 62.8.

Gutierez *et al*. [48] analyzed a GPU-based KNN on NVIDIA K20M. They applied it to KDD-CUP 1999 dataset of 4.9 million samples and 42 attributes, and achieved runtime of 2msec for K=1000. On PRINS, the same task completes in 0.21ms, showing speedup of almost 10. Energy efficiency figures for [47] and [48] were calculated based on the device power and reported execution time.

Lee *et al.* [5] presented KNN simulation on the Micron Automata Processor (AP) [4]. AP is a non-von Neumann near-data processing architecture for high speed automata evaluation. Automata processors do not have an arithmetic logic unit (ALU), therefore the authors could not use Euclidean distance. Instead, a variation of Hamming distance on quantized binary code [49] of the input was used. The authors compared AP performance with several existing platforms. We use the authors' projected performance of the AP with their proposed automata optimizations and architectural extensions, which show the best performance and energy figures.

In addition, we select the three best-performing compared platforms from [5] to be included in our comparisons: an Intel Xeon multicore CPU, a Xilinx Kintex-7 FPGA and an NVIDIA GTX Titan X GPU. Several datasets were used, each of $2^{20} \approx 1$M samples, binary coded to 64-256 bit. For CPU, the authors used the FLANN implementation [50] of Hamming distance. For GPU, a CUDA implementation of KNN [51] was modified to use XOR and population count, instead of Euclidean distance. Energy efficiency figures for all comparisons with [5] are reported in [5] and are used as-is in this work. For the other two comparisons, device published TDP were used (25W for the Stratix IV and 225W for the NVIDIA K20M). On PRINS, the

Table 6: KNN compared platforms with PRINS.

Architecture	Device	Work Ref.	Dataset	Avg. Time per Query (ms)	Energy Eff. (query/Joule)	PRINS Time per Query (ms)	PRINS Energy eff. (query/Joule)	PRINS Speedup	PRINS Power Efficiency Ratio
FPGA	Stratix IV FPGA	[47]	KDD-Cup 2004	69.1	0.6	2.2	473	31.4×	817×
	Xilinx Kintex-7	[5]	WordEmbed	0.45	593.9	3.7 µsec	11841	122×	19.9×
			SIFT	0.9	296.9	9 µsec	4829	100.7×	16.3×
			TagSpace	1.8	148.4	21.5 µsec	1018	83.7×	6.9×
GPU	NVIDIA K20M GPU	[48]	KDD-Cup 99	2	2.1	0.42	4.2	5×	2×
	NVIDIA GTX Titan X	[5]	WordEmbed	0.24	83.8	3.7 µsec	11841	65.3×	141×
			SIFT	0.25	81.9	9 µsec	4829	27.8×	59×
			TagSpace	0.25	81	21.5 µsec	1018	11.7×	12.6×
CPU	Intel Xeon E5-2620	[5]	WordEmbed	4.8	3.9	3.7 µsec	11841	1312×	3036×
			SIFT	8.1	2.3	9 µsec	4829	905×	2054×
			TagSpace	16.7	1.3	21.5 µsec	1018	682×	782.9×
Automata	Micron Automata Processor	[5]	WordEmbed	9 µsec	1738	3.7 µsec	11841	2.6×	6.8×
			SIFT	15 µsec	1092	9 µsec	4829	1.7×	4.4×
			TagSpace	56 µsec	236.3	21.5 µsec	1018	2.6×	4.3×

lack of hardened arithmetic units allows for flexibility of the compute operations. The binary Hamming distance calculation can be performed efficiently with a series of 1-bit compare and 8-bit counter increment operations on the entire dataset in parallel, resulting in considerable speedups and equal or better power efficiency compared with all platforms.

VI. Conclusions

This paper presents an evaluation of K-means and KNN compute intensive machine learning algorithms on PRINS, a non-von Neumann processing-in-storage system based on resistive content addressable memory (ReCAM). PRINS is a massively parallel and scalable SIMD architecture with *in-situ* processing capabilities. It uses associative processing instead of Boolean logic which allows executing any logic or arithmetic operation in a fixed number of cycles. Every ReCAM integrated circuit can contain hundreds of millions of data rows, each row serving as an associative processing unit.

PRINS performance and power-efficiency while executing these two algorithms have been compared to several other works. The evaluation shows that PRINS improves performance by up to three orders of magnitude and achieves better power efficiency for both algorithms, when compared to host based and near-data processing architectures.

References

[1] R. Balasubramonian, J. Chang, and T. Manning, "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro*, 2014.

[2] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A Resistive CAM Processing-in-Storage Architecture for DNA Sequence Alignment," *IEEE Micro*, vol. 37, no. 4, pp. 20–28, 2017.

[3] R. Kaplan *et al.*, "Deduplication in Resistive Content Addressable Memory Based Solid State Drive," pp. 100–106, 2016.

[4] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3088–3098, 2014.

[5] V. T. Lee, J. Kotalik, C. C. del Mundo, A. Alaghi, L. Ceze, and M. Oskin, "Similarity Search on Automata Processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 523–534.

[6] J. L. Potter and W. C. Meilander, "Array processor supercomputers," *Proc. IEEE*, vol. 77, no. 12, pp. 1896–1914, 1989.

[7] M. Hall, P. Kogge, J. Koller, P. Diniz, and J. Chame, "Mapping irregular applications to DIVA, a PIM-based data-intensive architecture," *Proc.*, 1999.

[8] T. L. Sterling and H. P. Zima, "Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing," in *ACM/IEEE SC 2002 Conference (SC'02)*, 2002, pp. 48–48.

[9] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the terasys massively parallel PIM array," *Computer (Long. Beach. Calif).*, vol. 28, no. 4, pp. 23–31, 1995.

[10] S. Wong, a. El-Gamal, P. Griffin, Y. Nishi, F. Pease, and J. Plummer, "Monolithic 3D Integrated Circuits," *2007 Int. Symp. VLSI Technol. Syst. Appl.*, no. c, pp. 5–8, 2007.

[11] Hybrid Memory Cube Consortium, *Hybrid Memory Cube Specification 1.0*. 2013.

[12] D.-H. Bae *et al.*, "Intelligent SSD," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*, 2013, pp. 1573–1576.

[13] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman, "Active Flash: Out-of-core data analytics on flash storage," in *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, 2012, pp. 1–12.

[14] Y.-Y. Jo, S. Cho, S.-W. Kim, and H. Oh, "Collaborative processing of data-intensive algorithms with CPU, intelligent SSD, and GPU," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, 2016, pp. 1865–1870.

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

[16] B. Y. Cho, S. Jeong, D. Oh, and W. W. Ro, "XSD: Accelerating

MapReduce by Harnessing the GPU inside an SSD."
[17] A. De, M. Gokhale, R. Gupta, and S. Swanson, "Minerva: Accelerating Data Analysis in Next-Generation SSDs," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, 2013, pp. 9–16.
[18] S.-W. Jun *et al.*, "BlueDBM," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA '15*, 2015, pp. 1–13.
[19] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart SSDs: opportunities and challenges," in *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, 2013, pp. 1221–1230.
[20] M. M. Shulaker *et al.*, "Three-dimensional integration of nanotechnologies for computing and data storage on a single chip," *Nat. Publ. Gr.*, vol. 547, no. 7661, pp. 74–78, 2017.
[21] T. Liu *et al.*, "A 130.7-mm^2 2-Layer 32-Gb ReRAM Memory Device in 24-nm Technology," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 140–153, 2014.
[22] K. C. Rahman, D. Hammerstrom, Y. Li, H. Castagnaro, and M. A. Perkowski, "Methodology and Design of a Massively Parallel Memristive Stateful IMPLY Logic based Reconfigurable Architecture," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 675–686, Jul. 2016.
[23] S. Kvatinsky *et al.*, "MAGIC—Memristor-Aided Logic," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.
[24] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, 2016.
[25] Y. Li *et al.*, "Realization of Functional Complete Stateful Boolean Logic in Memristive Crossbar," *ACS Appl. Mater. Interfaces*, vol. 8, no. 50, pp. 34559–34567, 2016.
[26] S. Paul and S. Bhunia, "A Scalable Memory-Based Reconfigurable Computing Framework for Nanoscale Crossbar," *IEEE Trans. Nanotechnol.*, vol. 11, no. 3, pp. 451–462, May 2012.
[27] P. Chi *et al.*, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, vol. 44, no. 3, pp. 27–39.
[28] A. Shafiee, A. Nag, and N. Muralimanohar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *Proc. 43rd*, 2016.
[29] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive TCAM accelerator for data-intensive computing," *Proc. 44th Annu. IEEE/ACM*, 2011.
[30] Q. Guo, X. Guo, R. Patel, and E. Ipek, "Ac-dimm: associative computing with stt-mram," *ACM SIGARCH Comput.*, 2013.
[31] Q. Guo, X. Guo, Y. Bai, R. Patel, and E. Ipek, "Resistive ternary content addressable memory systems for data-intensive computing," *IEEE Micro*, 2015.
[32] 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.
[33] A. Torrezan, J. Strachan, and G. Medeiros-Ribeiro, "Sub-nanosecond switching of a tantalum oxide memristor," 2011.
[34] C. Nail *et al.*, "Understanding RRAM endurance, retention and window margin trade-off using experimental results and simulations," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, p. 4.5.1-4.5.4.
[35] H.-S. P. Wong *et al.*, "Metal–Oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.
[36] J. Joshua Yang *et al.*, "Memristive devices for computing.," *Nat. Nanotechnol.*, vol. 8, no. 1, pp. 13–24, 2013.
[37] F. Alibart, T. Sherwood, and D. B. Strukov, "Hybrid CMOS/nanodevice circuits for high throughput pattern matching applications," in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 279–286.
[38] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive Associative Processor," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 148–151, Jul. 2015.
[39] S. Schechter *et al.*, "Use ECP, not ECC, for hard failures in resistive memories," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 141, Jun. 2010.
[40] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, 2011, pp. 466–477.
[41] S. Kvatinsky, E. Friedman, and A. Kolodny, "TEAM: Threshold adaptive memristor model," *IEEE Trans.*, 2013.
[42] Y. Ding, Y. Zhao, N. Xipeng Shen, M. Musuvathi, and M. Todd Mytkowicz, "Yinyang K-Means: A Drop-In Replacement of the Classic K-Means with Consistent Speedup."
[43] Z. Li, J. Jin, and L. Wang, "High-performance K-means Implementation based on a Simplified Map-Reduce Architecture," Oct. 2016.
[44] N. Ramanathan, J. Wickerson, F. Winterstein, and G. A. Constantinides, "A case for work-stealing on FPGAs with OpenCL atomics," *Proc. 2016 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 48–53, 2016.
[45] J. Bhimani, M. Leeser, and N. Mi, "Accelerating K-Means clustering with parallel implementations and GPU computing," *High Perform. Extrem.*, 2015.
[46] C. J. Rossbach, Y. Yu, J. Currey, J.-P. Martin, and D. Fetterly, "Dandelion: a Compiler and Runtime for Heterogeneous Systems," pp. 3–613, 2013.
[47] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl," *Field-Programmable Cust.*, 2015.
[48] P. Gutiérrez, M. Lastra, J. Bacardit, and J. Benítez, "GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs," *Information*, 2016.
[49] Y. Gong, S. Lazebnik, C. Science, and U. N. C. C. Hill, "Iterative Quantization : A Procrustean Approach to Learning Binary Codes," pp. 1–15.
[50] M. Muja, M. Muja, and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," *VISAPP Int. Conf. Comput. Vis. THEORY Appl.*, pp. 331--340, 2009.
[51] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–6.
[52] "UCI Machine Learning Repository." [Online]. Available: https://archive.ics.uci.edu/ml/index.php. [Accessed: 19-Jul-2017].
[53] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From Word Embeddings To Document Distances," *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, pp. 957–966, 2015.
[54] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, 2004.
[55] J. Weston and K. Adams, "#TagSpace : Semantic Embeddings from Hashtags," pp. 1822–1827, 2014.

Roman Kaplan (M'17) is a PhD candidate in the Viterbi faculty of Electrical Engineering, Technion, under the supervision of Prof. Ran Ginosar. Kaplan's research interests are parallel computer architectures, in-data accelerators for machine learning and bioinformatics.

Leonid Yavits Leonid is a postdoc fellow in Electrical Engineering in the Technion. He co-authored a number of patents and research papers on SoC and ASIC. His research interests include non von Neumann computer architectures and processing in memory.

Ran Ginosar is a Professor at the Department of Electrical Engineering and serves as Head of the VLSI Systems Research Center at the Technion. His research interests include VLSI architecture, manycore computers, asynchronous logic and synchronization, networks on chip and biologic implant chips.