PACT: G: From Near-Data Processing to In-Data Processing-in-Storage

Roman Kaplan
Technion, Israel Institute of Technology
sromanka@campus.technion.ac.il

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 architectures, especially the bandwidth wall.

The goal of this work is to design a storage device that does not suffer from the von Neumann bandwidth bottleneck and can provide high performance on massively parallel big data workloads. To achieve this, a novel processing-in-storage system based on Resistive Content Addressable Memory (ReCAM) is presented. ReCAM functions simultaneously as storage and as a parallel associative processor. ReCAM processing-in-storage resolves the bandwidth wall by keeping computation inside the storage array, thus implementing in-data, rather than near-data, processing.

This work shows that ReCAM based processing-in-storage architecture may outperform existing near-data and accelerator based designs. ReCAM processing-in-storage implementation of Smith-Waterman DNA sequence alignment reaches a speedup of almost five over a GPU cluster. For K-means, ReCAM processing-in-storage achieves speedup of 4.6—68 relative to CPU, GPU and FPGA, and for K-Nearest Neighbors, ReCAM processing-in-storage achieves speedup of 17.9—17,470.

Finally, an implementation of in-storage inline data deduplication is presented and shown to achieve orders of magnitude higher throughput than traditional CPU and DRAM based systems.

Keywords | Content addressable memory, associative processing, memristors, in-storage processing

1. Problem and Motivation

Until the breakthrough of Dennard scaling designers focused on improving performance for a single core by increasing its clock rate. Since 2005, when Dennard scaling diminished but Moore’s law continued, the focus has shifted to increasing the number of cores in multicore processors [12]. However, in the era of big data [14], improved processing elements do not necessarily translate to increased performance. One challenging definition for Big Data is “the amount of data just beyond technology’s capability to store, manage and process efficiently” [21]. Large contributors to the increase in the volume of data are social media, Internet of Things (IoT) and multimedia. All have significantly increased the rate of produced data, whether structured or unstructured. Moreover, big data has the potential to transform a plethora of fields, including science, engineering, healthcare, finance and many others.

Big data datasets do not fit in DRAM of a single machine or even a cluster of machines. Therefore, data is typically fetched to the CPUs and their memory hierarchies from non-volatile storage such as hard disks or Flash SSDs. Consequently, bandwidth and access-time to storage pose a major concern for Big Data applications. The throughput at which data can be fetched from storage to the processing units (CPU or an accelerator) has become a main determinant of system performance. The problem worsens in a datacenter cloud environment, where a dataset can be distributed among multiple nodes across the datacenter. In this case, communicating the stored data adds latency and reduces bandwidth even further, placing low bounds for maximal performance.

This challenge has motivated re-emerging Near-Data Processing (NDP) [2]. The data-centric NDP shifts computing closer to data. NDP seeks to minimize data movement by computing at the most appropriate location in the hierarchy, which can be cache, main memory or persistent storage. With NDP, less data needs to be transferred through levels of hierarchy, thus alleviating the limited bandwidth problem. Since storage bandwidth constitutes the key bottleneck in big data processing, NDP proposals that place computing resources at the cache level [42] or in main memory [18][1][36] (also known as Processing-in-Memory or PiM) do not address the bandwidth bottleneck problem facing big data workloads. In-storage NDP targets the source of the problem, but offers only a partial solution, as reviewed and discussed in section 2.3.

Resistive CAM (ReCAM), a storage device based on emerging resistive materials in the bitcell with a novel non-von Neumann Processing-in-Storage (PRinS) compute paradigm, is proposed in order to mitigate the storage bandwidth bottleneck of big data processing. Section 2 provides background on the basics of ReCAM and PRinS and covers related work. Section 3 presents the ReCAM architecture, explains how processing is performed within ReCAM and establishes its scalability. PRinS implementations for three key algorithms are presented in Section 4 and compared to other approaches: Smith-Waterman DNA sequence alignment, K-means clustering and K-nearest neighbors. In addition, previous work on in-storage data deduplication with ReCAM is concisely presented.

2. Background and Related Work

Three basic concepts underlie the proposed ReCAM: content addressable memories, associative processing and resistive materials. The following two subsections introduce each concept. The third subsection covers related work on NDP and highlights their limitations at addressing the storage bandwidth challenge of big data processing.

2.1 Content Addressable Memory and Associative Processing

Content addressable memory (CAM), also called associative memory, allows the comparison of all data words to a key in...
parallel, tagging the matching words, and possibly reading some or all of the tagged words, one by one. Standard memory read and write operations of a single word at a time can also take place.

In addition to storing information, a CAM array can be modified to function as an associative processor [13][41]. In associative processing, the parallel compare and parallel write operations supported by CAM are used to implement an ‘if condition then value’ expression. Thus, complex Boolean expressions are evaluated in parallel on all data words (CAM rows) by sequential execution of truth table if-then lines. Each (multi-bit) argument of a truth table line is matched with the contents of the appropriate field in the entire CAM array: the matching rows are tagged and the corresponding result values from the truth table line are written into the designated fields of all tagged rows.

For an $m$-bit argument $x$, any Boolean $f(x)$ has $2^m$ possible values, therefore the associative computing operation should incur $O(2^m)$ cycles, regardless of the dataset size. More efficiently, arithmetic operations can be performed on ReCAM in a word-parallel, bit-serial manner, reducing compute time from $O(2^m)$ to $O(m)$. The massive parallelism of each operation compensates in performance for the relatively large number of (parallel execution) cycles of each arithmetic operation.

More complex computations (more than Boolean functions) are decomposed into series of Boolean expressions, as usual [41][43].

2.2 Resistive Memories
Resistive random access memory (ReRAM) stores information by modulating the resistance of nanoscale storage elements. They are nonvolatile, free of leakage power, and emerge as long-term potential alternatives to charge-based memories, including NAND flash [3]. In particular, the storage elements can be placed as a layer above a normal CMOS logic circuit, enabling integrated processing-in-storage (PRinS). A 32GB ReRAM with a CMOS logic layer underneath [29] demonstrated a high density functional chip.

2.3 Related Work
The concept of NDP has been considered for several decades [25]. One approach to implementing NDP employs 3D stacked memories (e.g., [1][16]). DRAM memory is placed on top of a logic die allowing for a high-speed vertical inner interface between the memory and logic. The Hybrid Memory Cube [31] is another memory design (and a product) that exploits the increased bandwidth between DRAM and near-memory logic, and may be exploited for NDP [4].

Another approach for NDP in memory employs custom memory chips with resistive bitcells. In [18] and [19], a resistive associative memory is used as the main memory. Resistive bitcells are also used in analog computing [8][37], proposing PiM structures that allow the memory array to be used as RAM, scratchpad and analog dot product engine. Both trends are still limited to memory NDP.

In-storage NDP, an SSD comprising an array of Flash chips, is amended by an embedded CPU [22] or GPU [9] to perform some processing tasks in the SSD rather than in the host CPU, as demonstrated in Figure 1a. Often called Intelligent (or Smart) SSD, the main idea is increasing effective bandwidth to the in-SSD processors by connecting these processors directly to the flash banks inside the SSD. However, the storage bandwidth inside the SSD is still limited by the read/write ports of the flash devices.

The goal of this work is to design a storage device that does not suffer from von Neumann architecture bandwidth bottleneck and can provide high performance on big data workloads. To achieve this, processing is pushed upon each bit of storage.

3. Approach and Uniqueness
The approach of this work is to design a device for storing big datasets and processing them efficiently. The key properties of this design are scalability and massively parallel processing, possible due to the non-von Neumann architecture. Parallelism is achieved by in-situ processing of the data, in contrast with NDP approaches. In this work, Resistive CAM (ReCAM), a non-volatile and scalable storage device with resistive bitcells and a novel Processing-in-Storage (PRinS) paradigm is presented. The concept is demonstrated in Figure 1b.

3.1 ReCAM Crossbar Array
Figure 2a shows a single integrated circuit (IC) of the resistive CAM crossbar. A bitcell shown in Figure 2b, consists of two transistors and two resistive elements (2T2R). The KEY register contains a data word to be written or compared against. The MASK register defines the active columns for write and read operations, enabling bit selectivity. The TAG register (Figure 2c) marks the rows that are matched by the compare operation and may be affected by a parallel write. The TAG register enables chaining multiple ReCAM ICs.

In a conventional CAM, compare operation is typically followed by a read of the matched data word. When in-storage processing involves arithmetic operations, a compare is usually followed by a parallel write into the unmasked bits of all tagged rows, and additional capabilities, such as read and reduction operations, are included [42].

Any computational expression can be efficiently implemented in ReCAM storage using line-by-line execution of the truth table of...
the expression [41]. Arithmetic operations are typically performed bit-serially. Table 1 lists several operations supported by ReCAM and the number of cycles required by each operation. Addition (in-place or not) is performed in a bit-serial manner using the truth-table approach (32 bits times 8 truth-table rows times 2 for compare and write amounts to 512 cycles). Shifting down a sequence of 32 bits in a consecutive block of rows by one line requires three 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 2c). Third, write ‘1’ copies the shifted TAG to the same bit-column. Row-wise maximum compares two 32-bit numbers in each row in parallel. Max Scalar tags all rows that contain the maximal value in the selected 32 bits.

3.2 System Architecture
Conceptually, the ReCAM comprises hundreds of millions of rows, each serving as a computational unit. Due to power die restrictions, the entire array may be divided into multiple smaller ICs, as in Figure 3a. A row is fully contained within an IC. All ICs are daisy-chained for Shift and Max Scalar operations.

The ReCAM storage system uses a microcontroller (Figure 3b) similar to [19]. It issues instructions, sets the KEY and MASK registers, handles control sequences and executes read requests. In addition, the microcontroller may also perform some baseline processing, such as normalization of the reduction tree results.

ReCAM-based storage is scalable due to its inherent parallelism. It allows for scalability by adding more ICs and increasing storage capacity at no performance cost since compute capability is linearly scalable in the number of ICs. Therefore, processing in-storage of large data sets does not require ReCAM for external communication, in contrast to datacenter-scale storage.

4. Results and Contribution
A cycle-accurate simulator of the ReCAM storage was constructed. Assumed operational frequency is 500MHz. An in-house power simulator was used to evaluate the power consumption of ReCAM. The latency and energy figures used by both the timing and power simulations are obtained using SPICE simulation and are detailed in [41].

4.1 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 bioinformatic tools. The Smith-Waterman algorithm (S-W) [38] provides an optimal solution for the pairwise sequence alignment problem, but requires a number of operations proportional to the product of the two sequences.

S-W identifies the optimal alignment of two sequences by computing a two-dimensional scoring matrix. Matchings base-pairs score positively, while mismatching result in a negative score. The optimal alignment score of two sequences is the highest score in the matrix. 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 this work.

Figure 4a shows a snapshot of the scoring matrix during the algorithm execution. In a parallel implementation, the matrix is filled along the main diagonal and the entire anti-diagonal scores are calculated in parallel. Figure 4b shows the ReCAM memory map of two consecutive 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 [17]. AD[0], AD[1] and AD[2] contain scoring matrix anti-diagonals. Scores are represented by 32-bit integers. Shift operations in the PRinS implementation move data between rows inside an IC and between daisy-chained ICs. Figure 4c shows the ReCAM memory map at the end of an iteration and the mapping between ReCAM and the scoring matrix. A complete description of the S-W PRinS implementation appears in [23].

We simulate the ReCAM with the cycle-accurate simulator. The CUPS metric (Cell Updates per Second) is used to measure S-W performance. Results are compared to other works in Table 2. The in-storage implementation is compared to other implementations in different platforms: a 384-GPU cluster [36], the 128-FPGA RIVYERA platform [39] and a four Xeon Phi implementation [30]. On ReCAM with a total of 8GB in 32 separate ICs, each 256MB and 8M rows, 53 TCUPS are demonstrated, computing a total of 57,2×10^12 scores. 4.7x faster than the best implementation. The table also shows computed GCUPS/Watt ratios; ReCAM is close to twice better than the FPGA solution and 80x better than the GPU system.

4.2 K-Means Clustering
K-means is an unsupervised learning algorithm for clustering unclassified samples. It aims to partition N samples into K clusters, where each sample belongs to the cluster with the nearest center (mean).

An in-storage version of the algorithm was designed and implemented in the cycle-accurate simulator. The algorithm contains two K-iteration loops, assignment and update. The assignment loop first associatively calculates (in parallel) the Euclidean distance between all samples and each mean. Next, the loop assigns each sample with its closest mean in parallel, on all samples. The update loop recalculates the new mean coordinates.
Figure 4: (a) A snapshot of the dynamic programming matrix, shows the direction of progress for the parallel algorithm. (b) and (c) shows an example of organization of data in the ReCAM crossbar array at the beginning (b) and end (c) of an iteration. AD[2] contents in (b) is being replaced with the new result (c). Bottom rows in a crossbar IC are daisy-chained to the next IC in a shift instruction. The cell marked with X contains the global maximum score.

These two loops may be repeated until the mean coordinates converge.

Several evaluations are performed. The ReCAM K-means PRinS implementation is compared with Altera Stratix V [34] and Xilinx ZC706 [28] FPGAs, Intel i7-3770K eight-core CPU [11], NVIDIA K20M GPU [6], and ten NVIDIA K20M GPU cluster [35] implementations. Figure 5 presents the evaluated dataset sizes, the average runtime per iteration (log scale) of each architecture and the relative speedup of ReCAM.

Overall, ReCAM achieved speedups ranging between 4.6 and 67.8. One notable comparison is to Rossbach et al. [35] who used a ten GPU cluster with a large data set of 1 billion samples, occupying roughly 150GB. Their average iteration time is 30.6 seconds, compared with 0.65 seconds on ReCAM, yielding speedup of 47. This comparison highlights the benefit of PRinS insensitivity to dataset size, in contrast to the GPU cluster, which is limited by the inter- and intra-node communication. A complete description of the K-means PRinS implementation and performance comparisons appears in [43].

4.3 K-Nearest Neighbors (KNN)

K-nearest neighbors (KNN) is a method frequently used for classification. It computes the distances between an (unclassified) input query vector and a dataset of classified samples. Query vector classification is usually determined by the majority vote of K nearest database samples, hence the name K-nearest neighbors.

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

Our in-storage implementation has two steps. The first computes the Euclidean distance between the query vector and all dataset samples in parallel. The second step iteratively finds the K dataset samples that are closest to the query vector (the nearest neighbors), one by one.

The ReCAM processing-in-storage implementation is compared with Stratix IV FPGA [32], NVIDIA K20M [20] and NVIDIA Titan X GPUs [27], and Nearest neighbor Content Addressable Memory (NCAM) [27] KNN implementations. The latter is a NDP device with 3D-stacked DRAM on top of a custom logic. Figure 6 plots runtime (log scale) results and relative speedup of ReCAM, in addition to the datasets used in the evaluations. Overall ReCAM achieved speedups ranging between 17.9 – 17,470. A complete description of the KNN PRinS implementation and the performance comparisons appears in [43].

Most notable is the comparison to NCAM. While 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 NDP, the ReCAM 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.
4.4 In-Storage Data Deduplication

Deduplication is a data compression technique for eliminating redundant copies of repeated data, designed to improve storage utilization. Files are split into multiple data blocks. Only unique blocks are meant to be stored.

The ReCAM native compare operation is used to find duplicate data blocks in a fixed number of cycles. In [24], an inline in-storage implementation of deduplication is presented and its performance is compared to a CPU and DRAM implementation. A 256GB ReCAM may achieve on average 100x higher throughput than a typical mass storage appliance with CPU and DRAM based deduplication, while consuming similar energy. The complete in-storage deduplication implementation on ReCAM, including the performance comparisons, appears in [24].

5. Conclusions

This paper introduces ReCAM, a Processing-in-Storage architecture. It combines storage with parallel processing of data within the storage. Each ReCAM IC device may contain hundreds of millions of data rows, each row serving as an associative processing unit. Many ReCAM ICs are chained together constituting a large storage device, suitable for big data applications. ReCAM applies parallel associative processing to the data. This paper presents Smith-Waterman DNA sequence alignment, K-Means, KNN algorithms and in-storage inline data deduplication on ReCAM and shows significant speedup over alternative non-in-storage implementations.

REFERENCES