SuperFri article template

Roman Kaplan[1], Leonid Yavits[1], Ran Ginosar[1]

Keywords: Content Addressable Memory, Associative Processing, in-storage processing, memristors.

Abstract

Near-data in-memory processing research has been gaining momentum in recent years. Typical processing-in-memory architecture places a single or several processing elements next to a volatile memory, enabling processing without transferring data to the host CPU. The processing elements benefit from an increased bandwidth to and from the volatile memory. Since this approach does not place processing next to the lowest hierarchy in the von Neumann architecture, i.e., in storage, it is exposed to the problems faced by von Neumann architectures, namely 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 processing-in-memory 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. 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.

1. Background and Motivation

Until the breakdown 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 [11]. However, in the era of big data [12], 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" [16]. 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.

[1]Israel Institute of Technology, Haifa, Israel

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) [5]. 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 [29] or in main memory [14] [1] [22] (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.

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 **??** provides background on the basic concepts 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 of two algorithms are presented in Section 4 and compared to other approaches: Smith-Waterman DNA sequence alignment and in-storage data deduplication.

¡Review in more depth the in-memory works.¿

¡Review near-storage architectures¿

2. Background and Related Work

Three basic concepts underline 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 [12]âĂŐ[41]. In associative processing, the parallel compare and parallel write operations supported by CAM are used to implement an âĂŸif 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 memories store information by modu-lating the resistance of nanoscale storage elements (memristors). 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 R_{off} maximum resistance (high resistive state, logic '0').

Resistive memories are nonvolatile, free of leakage power, and emerge as long-term potential alternatives to charge-based memories, including NAND flash. The metal-oxide resistive random access memory (ReRAM), employing one resistive device and possibly also one transistor (1R1T) per bit-cell, is considered a potential technology to replace next-generation nonvolatile memories. Its main features are high reliability and fast access speed. A test-chip of 32GB device with two ReRAM-based memory layers and a CMOS logic layer underneath has been developed [2], demonstrating design techniques to achieve a high density functional chip.

2.3. Related Work

The concept of NDP has been considered for several decades [19]. One approach to implementing NDP employs 3D stacked memories (e.g., [1] [13]). 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 [20] 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].

¡Start of in vs. near-data¿

2.3.1. 2D Processing-in-Memory Architectures

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 [60] used large number of processing units positioned in proximity to memory arrays to implement a massively parallel SIMD computer. Gokhale et al. [31] designed TeraSys, a computer architecture comprising a conventional host processor where at least part of its memory was replaced by a PIM array, integrating memory and ALUs in close proximity. Hall et al. [37] developed DIVA, the Data-Intensive Architecture, combining PIM memories with external host processors and performing selected computations in processing elements near memory and reducing the volume of data transferred across the long and slow processor-memory interface. Kogge et al. [41] developed HTMT, a parallel multilevel memory architecture, where

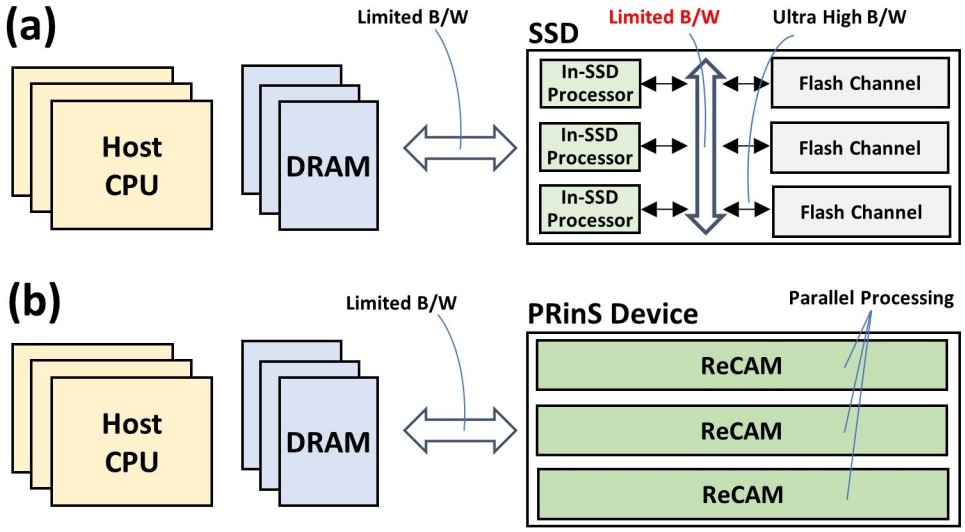

Figure 1. Comparison of (a) **near-data** processing-in-storage (b) and **in-data** processing-in-storage based on ReCAM.

each RAM level is a PIM memory (memory blocks interconnected with ALUs). Suh et al. [68] introduced a SLIIC QL computer featuring a processor integrated on the same die as DRAM. Lipovsky et al. 46] developed a dynamic associative access memory architecture that combined DRAM and a single-bit processing element, capable of associative and conventional arithmetic processing, placed in the sense amplifier area of a DRAM. Yavits et al. [56] 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. Nitin et al. [70] introduced RowCore, a near-memory processing architecture for Big Data Machine Learning. Gao et al. [30] proposed Heterogeneous Reconfigurable Logic, a reconfigurable array for near-data processing systems.

2.3.2. 3D Processing-in-Memory Architectures

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. [69][53] 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. [52] 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. [7] developed Smart Memory Cube and designed a high bandwidth interconnect to serve the bandwidth demand of PIM architecture. Zhang et al. [76] 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. [27] proposed NDA, a near-DRAM acceleration architecture that processes data using accelerators 3D-stacked on DRAM devices.

2.3.3. Processing-in-Memory with Memristors

Recently, emerging memory technologies such as resistive memory have become a focus of PIM research. Somnath et al. âĂŐ[59] developed MBARC, a resistive crossbar in-memory LUT-based processing architecture. Chi et al. âĂŐ[16] introduced PRIME, a PIM accelerator of neural network applications in RRAM based main memory. âĂŐ[57] introduced a resistive CAM based massively parallel accelerator. Shafiee et al. [65] developed an in-situ processing architecture, where memristor crossbar arrays are used to perform dot-product operations in an analog manner.

2.3.4. Near-Data Processing-in-Storage

Boboila et al. [12] 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. [8] 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. [38] studied optimal ways of combining CPU, GPU and SSD for efficient processing of data-intensive algorithms. Cho et al. [17] 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. [40] introduced the Smart SSD model, which combines in-SSD processing with a powerful host system, and constructed a Smart SSD prototype. De et al. [22] introduced the FPGA-based Minerva, which executed application-specific operations in the NVM controller. Jun et al. âĂŐ[39] 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. [18] 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.

¡End of in vs. near-data¿

3. Processing-in-Storage with ReCAM

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

While ReRAM may employ one transistor and one memristor (1T1R) cells, ReCAM uses 2T2R cells, following [3] [4]. Figure 2 shows the resistive CAM crossbar. A bitcell, shown in Figure 2a, 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 2b) 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 [28].

Table 1. Operations included in ReCAM

Integer Instruction (32bit)	Cycles
B ← A+B	256
C ← A+B	512
Shift down by one row	96
Row-wise Max (A,B)	64
Max Scalar (A)	64

Any computational expression can be efficiently implemented in ReCAM storage using line-by-line execution of the truth table of the expression [5]. Arithmetic operations are typically performed bit-serially. Table tab. 1 lists several operations supported by ReCAM and the number of cycles required by each operation. Shifting down a consecutive block of rows by one row position requires three cycles per bit. First, compare-to-âĂŸ1' 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. Shifting 32-bit numbers thus requires 96 cycles. Addition (in-place or not) is performed in a bit-serial manner using a truth table approach [5] (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.

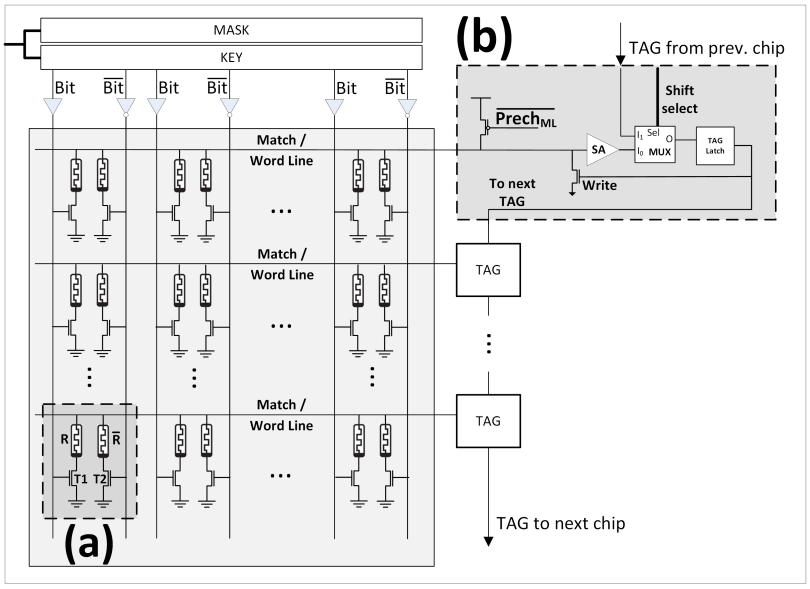

Figure 2. Single ReCAM crossbar integrated circuit. (a) 2T2R ReCAM bitcell. (b) TAG logic.

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 2a. 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 2b) similar to [15]. 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.

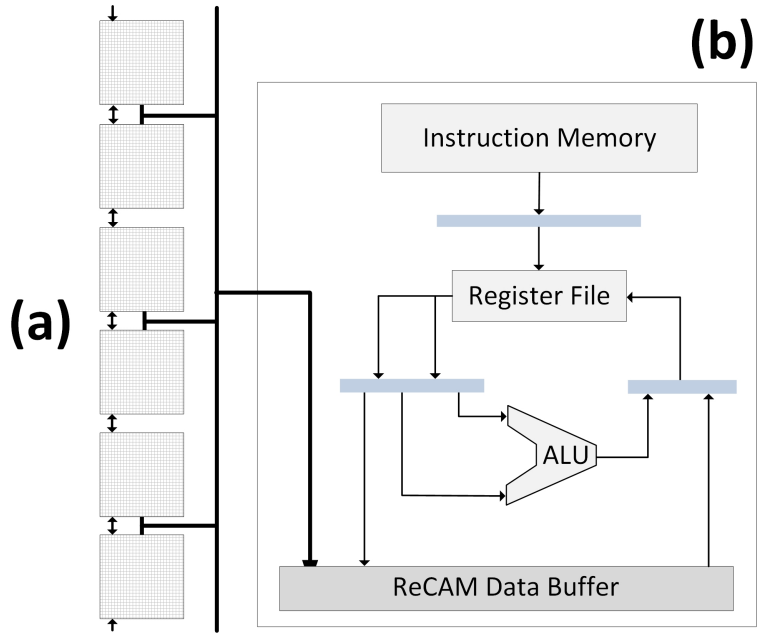

Figure 3. Complete ReCAM-based Storage system, composed of (a) separate multiple daisy-chained ICs and (b) Microcontroller. Connected to the multiple ICs with a reduction tree network.

4. PRinS Application: 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) [25] 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.

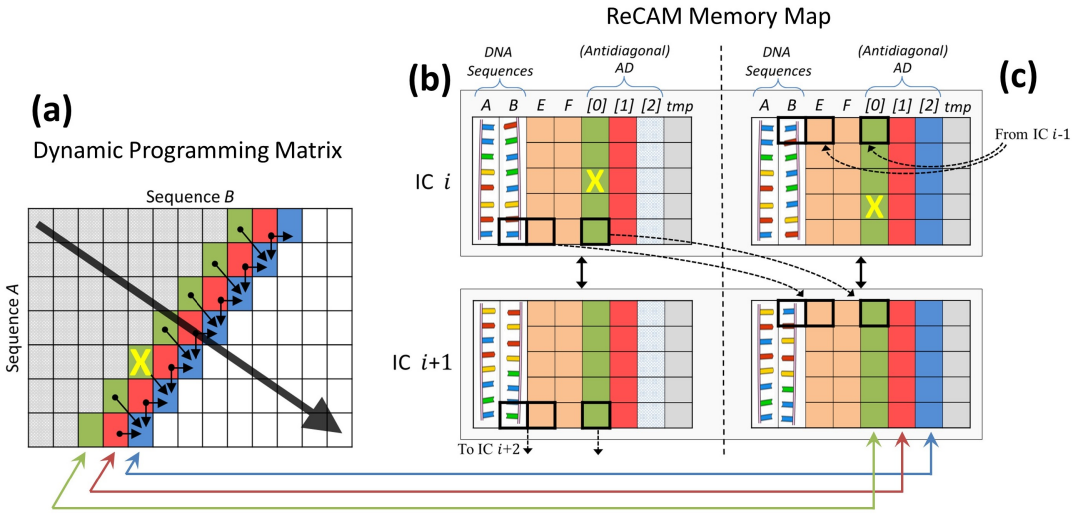

Figure 4. Mapping the dynamic programming matrix on ReCAM. (a) A snapshot of the dynamic programming matrix, shows the direction of progress for the parallel algorithm. (b) and (c) shows an example of organization of data in the ReCAM crossbar array at the beginning (b) and end (c) of an iteration. AD[2] 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 maxmium score.

Figure 2a 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 2b 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 [13]. 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 [18].

4.1. simulation and comparison to state-of-the-art

A cycle-accurate simulator of the ReCAM storage was constructed. Assumed operational frequency is 1GHz. 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].

We simulate the ReCAM with the cycle-accurate simulator. Assumed ReCAM parameters are listed in Table ¡X¿. 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 [22], the 128-FPGA RIVYERA platform âĂŐ [26] and a four Xeon Phi implementation [20]. On ReCAM with a total of 8GB in 32 separate ICs, each 256MB and 8M rows, 53 TCUPS are demonstrated, computing a total of 57×10^{12} scores. 4.7× faster than the best implementation. The table also shows computed GCUPS/Watt ratios; ReCAM is close to twice better than the FPGA solution and 80× better than the GPU system.

Table 2. Simulated ReCAM parameters

ReCAM Parameter	Value
Active storage size	8GB
Frequency	1Ghz
Power per integrated circuit	200W
Number of integrated circuit	32

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

Accelerator	Xeon Phi	FPGA	GPU	ReCAM
Performance (TCUPS)	0.23	6.0	11.1	53
Number of ICs	4	128	384	32
Power (kWatt)	0.8	1.3	100	6.6
GCUPS/Watt	0.3	4.7	0.1	8.0
Reference	[20]	[26]	[22]	

5. PRinS Application: 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. With every new write, a data block is compared against all blocks in the storage. If a match occurs, a pointer to the previously stored block is saved in lieu of the data block. Given that the same data block may occur multiple times (match frequency is also dependent on the block size), storage efficiency can be greatly improved [30].

Deduplication operates on the physical layer of the storage, managing a set of data structures to expose a consecutive logical layer of storage. Each data block has two addresses, physical (PA) and logical (LBA). Only the LBAs are exposed to the outside world, while physical addresses are used internally by the deduplication mechanism.

5.1. Related Work: Conventional Deduplication

In a typical inline storage deduplication system (comprising disk / SSD storage, CPU and DRAM for holding indices and tables), the basic deduplication data unit is termed a chunk. Upon writing a new data chunk to storage, comparing the chunk contents (typically 4-8 KByte) to the entire storage is infeasible. Instead, a much shorter representation, called a fingerprint or hash (e.g., 20-byte SHA-1 hash) is calculated for each chunk, and the fingerprint is looked up in a chunk index. If no entry is found, the chunk is stored and a new entry is added to the chunk index. In addition to the fingerprint, the index entry also holds at least the chunk's PA and the number of references to it (Figure 18). If the fingerprint of the new chunk is found, its number of references is incremented. An additional address translation table holds both the LBA and the PA of each chunk.

Conventional implementations of deduplication require a dedicated computer within the storage appliance. For example, a disk-based deduplication system [30] with usable capacity of 6TB employs 15 SATA drives (connected in RAID6), 500GB each, and two dual-core CPUs with 8GB

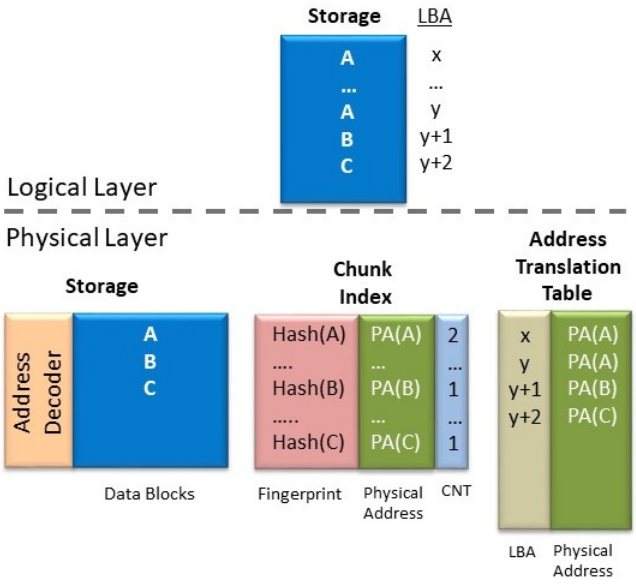

Figure 5. Conventional deduplication scheme after writing the following sequence of (data block, LBA): (A, x), (A, y), (B, y+1), (C, y+2). The storage, chunk index and address translation table reside in the physical layer.

of DRAM. It reaches 90% CPU utilization at peak I/O performance. All chunk metadata is stored on disk, while the DRAM serves as a cache for chunk metadata, to reduce non-I/O storage access. An expansion of that system [10] includes a flash-based SSD serving as fast storage for the entire chunk metadata. The configuration is similar to [30], although smaller, with a RAID4 storage comprising five hard drives, 500GB each, a dual-core CPU and 4GB of DRAM. As in the previous work, DRAM serves as a small cache for chunk metadata. Xtremio's X-brick [27] is an example of an all-flash high-end large-scale contemporary storage appliance. Each of its units contains either 13 or 25 SSDs with an effective capacity of 3.2 or 7.2 TB, respectively. The appliance supports up to 8 units and uses a quad-core processor with 256GB of DRAM.

At the other end of the spectrum, [7] shows an example of an in-SSD deduplication with the purpose of enhancing the device endurance. The authors suggest using the device controller and memory buffer to calculate the chunk fingerprint. Deduplication is implemented with an additional indirection in the flash translation layer and uses the buffer as a small cache (similar to the DRAM in [10] and [30]).

5.2. In-Storage ReCAM Based Deduplication

The proposed ReCAM based inline deduplication requires neither external CPU nor DRAM. The deduplication is accomplished entirely within the ReCAM, using its in-storage processing capabilities. ReCAM based deduplication is illustrated in Figure 19. Each data block in ReCAM storage is divided into $S = (block_size)$âĄĎ$(ReCAM_width)$ row-segments of $ReCAM\_width$ size. For example, for 256-bit wide ReCAM and 4KB blocks, the number of segments is $S = 128$. Data blocks are stored in ReCAM in segment by segment fashion, in S consecutive ReCAM rows. The first segment of each data block is marked by '1' in the block_start bit column. The values of $block\_start$ in all other ReCAM rows of the data block are zero.

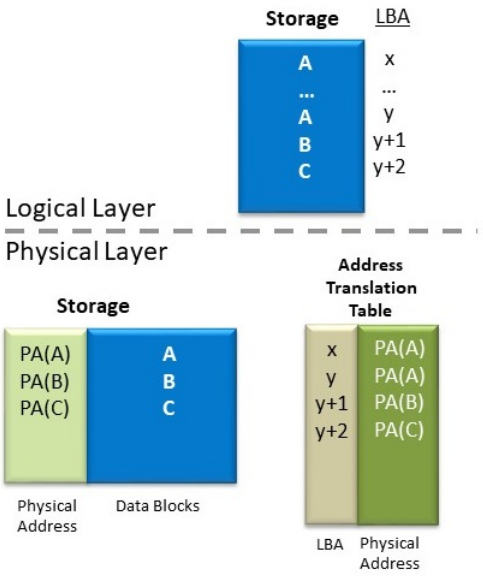

Figure 6. ReCAM based deduplication scheme, following the same sequence of writes as in Figure 5.

Otherwise, the new block is unique. In that case it is written into the ReCAM along with its (arbitrarily assigned, unique) PA. As described above, the block is written segment by segment into S consecutive rows, and the first segment is marked '1' in the start_block bit column.

In both cases (unique and duplicate), the LBA of the data block is placed together with its PA in an associative address translation table, which can be stored in a separate module of the ReCAM storage. The translation table mapping can be optimized to eliminate storing multiple copies of the same PA (of duplicated blocks). Overall, write takes $O(S)$ cycles.

Read is done in two steps. First, the LBA of the data block is searched in the associative address translation table. The corresponding PA is retrieved from the table. Second, the PA is searched in the ReCAM storage (by compare), followed by read of the data block from the matched ReCAM rows. It is accomplished by a series of S read operations, starting with the row marked by '1' in the start_block bit column. Overall, read operation takes $O(S)$ cycles.

Deletion of a data block is performed in three steps. In the first step, the LBA is searched in the address translation table; its PA is retrieved (to be used at the second step), and the entry at the address translation table is deleted (using the delete_data function of Figure 17). In the second step, the PA (retrieved at the first step) is searched again in the address translation table; if MATCH returns '0', it means that the deleted block has no duplicates. In this case, it is deleted from the ReCAM storage, in $O(S)$ cycles.

5.3. In-Storage Deduplication Evaluations

We simulate the ReCAM based deduplication using the cycle-accurate CAM simulator introduced in [28], employing ReCAM performance and power figures obtained by SPICE simulations. During ReCAM execution we record and count all operations (compare, write and delete). The simulated ReCAM size is 256GB, running at 1GHz. External data throughput is assumed non-limiting (contemporary interconnect such as multi-lane PCIe is capable of supporting in excess of 2.2M IOPS).

```
1: function DEDUP_WRITE(address, data_block)
2:     Compare (data_block)              ▷ in parallel for all data block stored in ReCAM
3:     if NNZ_TAG then                                          ▷ no match for data block
4:         (empty_bit)
5:         First_tag()
6:         Write (address, data, empty_bit)                          ▷ empty_bit ← 0
7:     else                                                  ▷ data_block is duplicated
8:         Read(address)          ▷ reads the address from matching data block in ReCAM
9:         Save (new block address)     ▷ Save pointer to an existing block in an associative
   address conversion table. Could be stored in a Se
10:    end if
11: end function

1: function DEDUP_READ(address)
2:     Compare (address)                          ▷ find address in address conversion table
3:     if NNZ_TAG == 0 then ▷ if found, block is deduplicated. Need to fetch its physical
   address.
4:         Read (physical_address)   ▷ read physical_address field from address conversion
   table
5:         Compare (pysical_address)                    ▷ find physical_address in ReCAM
6:     else                          ▷ block is unique, address is its physical address
7:         Compare (address)
8:     end if
9:     Read (data_block)                                          ▷ read a data_block
10: end function

1: function DEDUP_DELETE(address)
2:     Compare (address)                          ▷ find address in address conversion table
3:     if NNZ_TAG == 0 then                                        ▷ match for address.
4:         Remove (address)      ▷ if deduplicated, remove pointer from associative address
   conversion table
5:     else                                                      ▷ data block is unique
6:         Delete (address)                       ▷ delete data block from ReCAM storage
7:     end if
8: end function
```

Figure 7. Associative Write, Read and Delete in ReCAM-based data deduplication.

We compare our ReCAM deduplication implementation with opendedup âĂŐ [24], which supports inline deduplication and runs on top of the local filesystem. It allows for either variable or fixed-size blocks and does not limit the amount of stored data. In our analysis, block sizes of 1KB, 2KB, 4KB and 8KB were used. In addition, we used opendedup on a server with four octa-core Intel Xeon E5-4650 CPUs with 64GB of RAM and 800GB Intel SSD DC P3700 drive.

To evaluate the performance and energy consumption of opendedup,the file system benchmark IOzone was used [21]. IOzone allows writing data chunks with fixed number of duplicate parts,

to control the degree of deduplication. All runs include writing of 50GB of data, with varying percentage of duplicate blocks. Each test was repeated with inline deduplication on and off, to isolate the CPU and DRAM energy consumptions during deduplication. Intel performance counter monitor âĂŐ [9] was used for measurements.

As demonstrated by âĂŐ [30], real-world workloads have high variability in the percentage of duplicate data. Our goal is to exhaustively examine ReCAM performance and energy consumption. Therefore we use a suite of artificial workloads with a varying degree of duplication ratio. It allows us to control both the workload and the mainline system parameters. Both opendedup and ReCAM deduplicate all duplicate blocks.

The simulated write throughput as a function of percentage of deduplicated blocks is presented in Figure 20. The measured throughput of opendedup is also presented in Figure 20 for comparison. The ReCAM throughput increases with the percentage of duplicate blocks, as the number of writes drops. For 8KB data blocks, ReCAM storage reaches 2.2M IOPS for 30% duplicate blocks. For comparison, high-end all-flash X-brick storage appliance reaches 150K IOPS in 30% write, 70% read operation âĂŐ [27], similar to the simulated performance of opendedup.

The simulated energy consumption of ReCAM based deduplication as a function of percentage of deduplicated blocks is presented in Figure 21. To understand the energy benefits of continuous compare, we simulate the energy consumption without deactivating the ReCAM rows that have mismatched in previous compares. This results in much higher simulated energy consumption.

The measured energy consumption of opendedup (including the SSD energy consumption) is also presented in Figure 21 for comparison. The energy consumption of ReCAM based deduplication is in the same range (slightly higher for smaller blocks, lower for larger blocks).

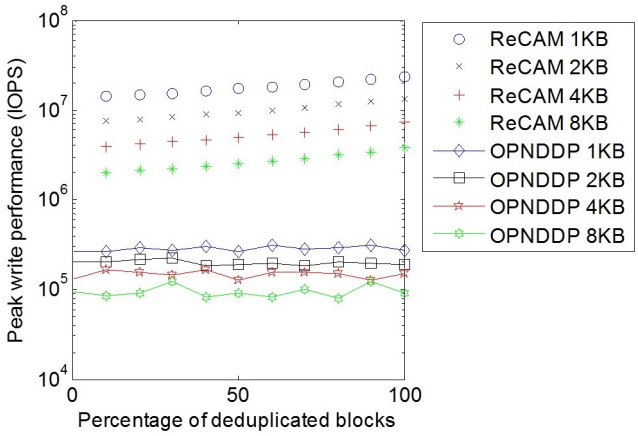

Figure 8. Write performance for different block sizes vs. percentage of deduplicated blocks, for data blocks of 1KB, 2KB, 4KB and 8KB (OPNDDP = Opendedup).

6. Conclusions

This paper introduces ReCAM, a Processing-in-Storage architecture. It combines storage with parallel processing of data within the storage. Each ReCAM integrated circuit may contain hundreds of millions of data rows, each row serving as an associative processing unit. Many ReCAM integrated circuits chained together constitute a large storage device, suitable for big data applications. ReCAM applies parallel associative processing to the data. This paper presents

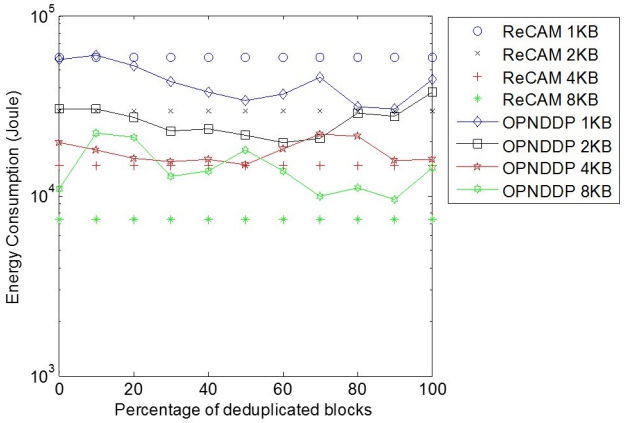

Figure 9. Deduplication energy for different block sizes vs. percentage of deduplicated blocks, for data blocks of 1KB, 2KB, 4KB and 8KB while writing 50GByte of data.

Smith-Waterman DNA sequence alignment and in-storage data deduplication on ReCAM and shows significant speedup over alternative non-in-storage implementations.

References

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

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

3. Akinaga, H., Shima, H.: Resistive random access memory (reram) based on metal oxides. Proceedings of the IEEE 98(12), 2237–2251 (2010)

4. Azarkhish, E., Pfister, C., Rossi, D., Loi, I., 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)

5. 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 (2014)

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

7. Chen, F., Luo, T., Zhang, X.: Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In: FAST. vol. 11, pp. 77–90 (2011)

8. Cho, B.Y., Jeong, W.S., Oh, D., Ro, W.W.: Xsd: Accelerating mapreduce by harnessing the gpu inside an ssd. In: Proceedings of the 1st Workshop on Near-Data Processing (2013)

9. Corporation, I.: Intel performance counter moniter. `www.intel.com/software/pcm`

10. Debnath, B.K., Sengupta, S., Li, J.: Chunkstash: Speeding up inline storage deduplication using flash memory. In: USENIX annual technical conference. pp. 1–16 (2010)

11. Esmaeilzadeh, H., Blem, E., St Amant, R., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: ACM SIGARCH Computer Architecture News. vol. 39, pp. 365–376. ACM (2011)

12. Foster, C.C.: Content addressable parallel processors. John Wiley & Sons, Inc. (1976)

13. Giridhar, B., Cieslak, M., Duggal, D., Dreslinski, R., Chen, H.M., Patti, R., Hold, B., Chakrabarti, C., Mudge, T., Blaauw, D.: Exploring dram organizations for energy-efficient and resilient exascale memories. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. p. 23. ACM (2013)

14. Guo, Q., Guo, X., Bai, Y., Ipek, E.: A resistive tcam accelerator for data-intensive computing. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 339–350. ACM (2011)

15. Guo, Q., Guo, X., Patel, R., Ipek, E., Friedman, E.G.: Ac-dimm: associative computing with stt-mram. ACM SIGARCH Computer Architecture News 41(3), 189–200 (2013)

16. Kaisler, S., Armour, F., Espinosa, J.A., Money, W.: Big data: Issues and challenges moving forward. In: System Sciences (HICSS), 2013 46th Hawaii International Conference on. pp. 995–1004. IEEE (2013)

17. Kang, Y., Kee, Y.s., Miller, E.L., Park, C.: Enabling cost-effective data processing with smart ssd. In: Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on. pp. 1–12. IEEE (2013)

18. Kaplan, R., Yavits, L., Ginosar, R., Weiser, U.: A resistive cam processing-in-storage architecture for dna sequence alignment. arXiv preprint arXiv:1701.04723 (2017)

19. Kogge, P.: A short history of pim at notre dame. July1999 (1999)

20. Liu, Y., Schmidt, B.: Swaphi: Smith-waterman protein database search on xeon phi coprocessors. In: Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference On. pp. 184–185. IEEE (2014)

21. Norcott, W.D., Capps, D.: Iozone filesystem benchmark (2003)

22. 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)

23. Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S., 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. IEEE Press (2016)

24. Silverberg, S.: âĂIJopendedup sdfs (2010)

25. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of molecular biology 147(1), 195–197 (1981)

26. Wienbrandt, L.: The fpga-based high-performance computer rivyera for applications in bioinformatics. In: Conference on Computability in Europe. pp. 383–392. Springer (2014)

27. XtremIO, E.: X-Brick tech spec. https://www.emc.com/collateral/data-sheet/h12451-xtremio-4-system-specifications-ss.pdf, accessed: 2017-07-06

28. Yavits, L., Kvatinsky, S., Morad, A., Ginosar, R.: Resistive associative processor. IEEE Computer Architecture Letters 14(2), 148–151 (2015)

29. Yavits, L., Morad, A., Ginosar, R.: Computer architecture with associative processor replacing last-level cache and simd accelerator. IEEE Transactions on Computers 64(2), 368–381 (2015)

30. Zhu, B., Li, K., Patterson, R.H.: Avoiding the disk bottleneck in the data domain deduplication file system. In: Fast. vol. 8, pp. 1–14 (2008)