Deduplication in Resistive Content Addressable Memory Based Solid State Drive

R. Kaplan, L. Yavits, A. Morad, R. Ginosar
Dept. of Electrical Engineering, Technion IIT
Haifa, Israel
sromanka@tx.technion.ac.il

Abstract—An in-storage deduplication based on a resistive content addressable memory (ReCAM) is proposed. The ReCAM native compare operation is used to find duplicate data blocks in a fixed number of cycles. The performance of ReCAM based in-storage deduplication is compared to Solid State Drive (SSD) based in-line deduplication performed in CPU and DRAM, showing an average 100x higher throughput at roughly the same energy consumption.

Keywords—SSD, Deduplication, In-Memory Computing, Memristor, Resistive RAM, Resistive CAM.

I. Introduction

Solid state disk (SSD) based on resistive memory (ReRAM), offering the potential for higher density, longer retention and higher endurance than NAND have been explored recently [6][10]. "Intelligent" or "smart" SSD, where some processing is implemented *in-situ*, by an embedded CPU or even GPU, is also an active field of research [3][5][13]. The motivation is reduction of latency and energy associated with data transfer between SSD and main CPU or server.

In this paper, we propose a solid state storage based on *Resistive Content Addressable Memory* (ReCAM) [18], and present in-storage implementation of deduplication. ReCAM is targeted due to its capability to combine processing with storage. This technology suits for application in enterprise high-speed mass storage systems such as high-end storage appliances [17] rather than in smaller SSDs dedicated for personal use.

We show that 256GByte ReCAM with inline in-storage deduplication may achieve on average 100x higher throughput than typical mass storage appliance with CPU and DRAM based deduplication [20][17], while consuming similar energy. Such ReCAM can be used as a basis for a ultra-high speed storage, or cache in a hybrid mass storage systems [4][8]. Another potential application for the proposed scheme is in-memory deduplication [15].

The rest of this paper is organized as follows. Section II presents the architecture of ReCAM based storage. Section III explores ReCAM based in-storage deduplication and compares it to related work of conventional deduplication. Section IV discusses simulation results and Section V offers conclusions.

II. ReCAM Based Storage

As CMOS feature scaling slows down, conventional memory technology experiences scalability problems. In response, resistive memory (ReRAM) technologies emerge as scalable, long-term potential alternatives to charge-based memories, including NAND flash [16]. Resistive memories such as memristors store information by modulating the resistance of nanoscale storage elements. Memristors are two-terminal devices, where the resistance of the device is changed by the electrical current or voltage. The resistance of the memristor is bounded by a minimum resistance R_{ON} (low resistive state, logic '1') and a maximum resistance R_{OFF} (high resistive state, logic '0').

A block diagram of a conventional all FLASH storage appliance is presented in Fig 1. Such appliance typically comprises a high-performance multicore processor, a large DRAM and a multitude of FLASH modules, containing FLASH control and NAND array. The appliance usually connects to outside world through a dedicated interface such as SCSI or SAS.

The block diagram of the proposed ReCAM based storage is shown in Fig 2. It contains a number of ReCAM modules, consisting of an interface unit (for example high speed Ethernet), ReCAM controller and ReCAM array.

The proposed storage employs the resistive crossbar of [12] as applied to CAM [1] (Fig 3(a)). The ReCAM array comprises of bit cells (Fig 3(b)) organized in bit-columns and word-rows. Several special registers are appended to the ReCAM array. The KEY register contains a data block to be written or compared against. The MASK register defines the active fields for write and read operations ("masks" them "on"), enabling bit selectivity. The TAG register marks the rows that are matched by the compare operation and may be affected by write.

A bitcell (Fig 3(b)) consists of two nonlinear bipolar memristors, storing complementary bits. Diodes are added to terminate sneak paths. This crossbar constitutes a ReCAM [18], operating either as a CAM or as a conventional

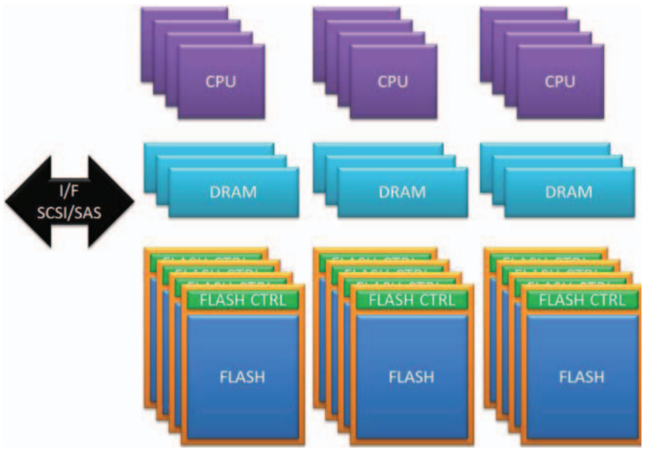

Fig 1. Conventional All FLASH Storage Appliance

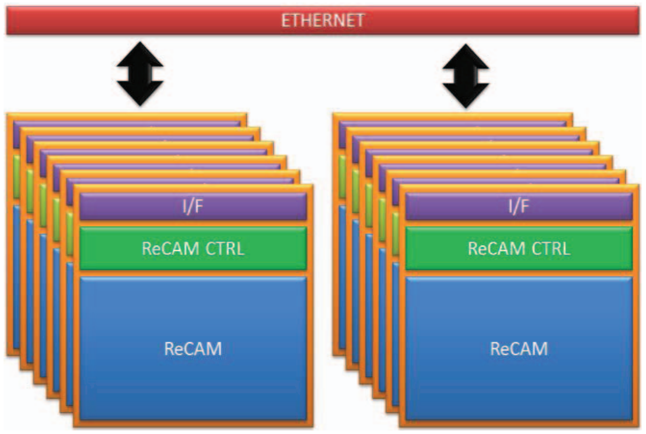

Fig 2. ReCAM based storage appliance

ReRAM. The bitcell footprint is $8F^2/k$, where k is the number of vertically integrated memristor layers [1] (unlike ReRAM which have a cell size of $4F^2$).

When operating as a CAM, the ReCAM can perform Compare and Write operations. The TAG column marks the rows that are matched by Compare.

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

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

Compare followed by a write operation are illustrated in Fig 4, which shows a fragment of ReCAM storing '0110' in the first row and '0101' in the second row; The ReCAM content is compared with the '011x' key and a new '1xxx' key is written in the tagged (first) row.

In ReCAM, sneak currents affect the compare operation

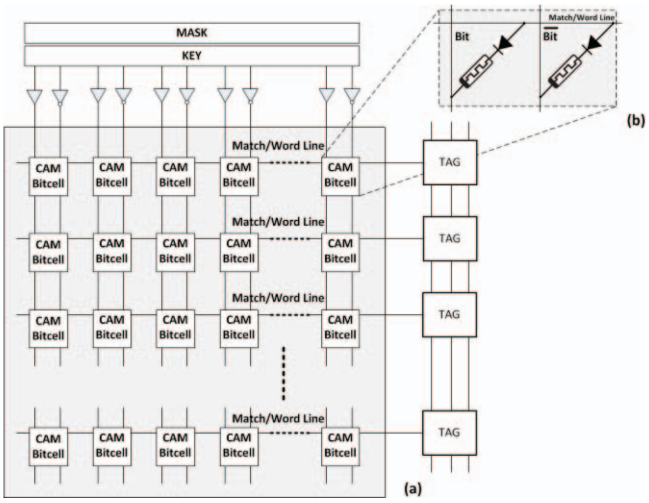

Fig 3. (a) Resistive crossbar, (b) ReCAM bitcell

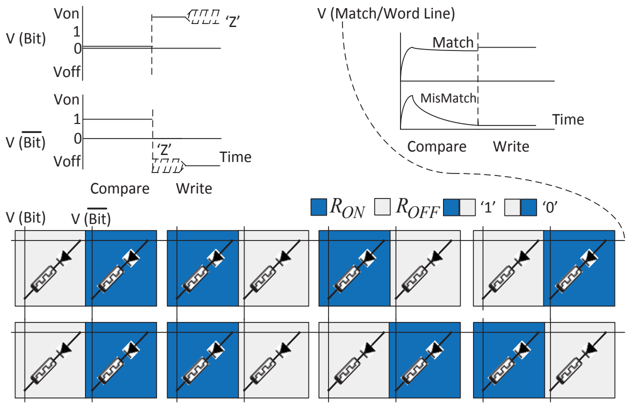

Fig 4. Compare and Write in ReCAM

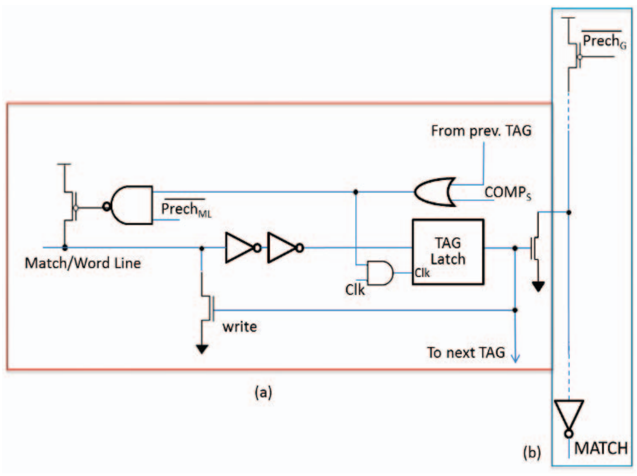

Fig 5. (a) TAG Logic; (b) MATCH logic

```
Init {                              // called once when storage is formatted
    Empty_bit column is set
}

Write_data (address, data) {
    Compare (empty_bit_col==1)  // tag topmost row having
                                // empty_bit=1
    Write (address, data, empty_bit=0)    // write into masked-on
                                // address, data, empty_bit fields
                                // of the tagged row
}

Delete_data (address) {
    Compare (address_field==address)
    Write (empty_bit=1)         // write into masked-on empty_bit
                                // field of the tagged row
}
```

Fig 6. Associative Write and Delete in ReCAM

(rather than read operation as in a standard ReRAM crossbar [18]). More specifically, there are sneak paths leading from a matching Match/Word Line (which is supposed to retain '1') through neighboring mismatching Match/Word Lines to the ground. The purpose of per-cell diode [1] is to terminate such sneak paths, so that current can only flow from a Match/Word Line to the ground (through a Bit Line) in one direction.

ReCAM behavior is verified and its performance and energy figures are obtained by SPICE simulations using the memristor TEAM model [9].

Two compare operations are defined: A *single* compare, where each TAG samples the match line of an individual row, and a *continuous* compare, where the match result of an individual row is ANDed with the TAG of the previous row.

TAG logic is illustrated in Fig 5(a). The AND, OR and NAND gates are used for continuous compare, as described in Section III. MATCH logic is illustrated in Fig 5 (b). The MATCH output (at the bottom of the array) is '1' if there is a match resulting from Compare in at least one of the ReCAM rows.

ReCAM behavior, performance and energy dissipation are further described in [18].

Data addressing in ReCAM based storage is achieved by storing the physical address alongside each data block, creating the *address field*. Read operation in ReCAM storage is performed in two cycles. First, the address is compared with the ReCAM address field. Second, the data is read from the row tagged by the Compare.

Associative addressing provides simple means for both writing new records and deleting existing records, as shown in Fig 6.

III. In-Storage ReCAM Based Deduplication

A. *Background*

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 [20].

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.

B. *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 (Fig 7). 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 [20] with usable capacity of 6TB employs 15 SATA drives (connected in RAID6), 500GB each, and two dual-core CPUs with 8GB 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 [4] includes a flash-based SSD serving as fast storage for the entire chunk metadata. The configuration is similar to [20], 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. Both papers provide detailed lists of parameters to tune for optimal system performance, such as the number of hash functions used in a variant of cuckoo hashing for the chunk index data structure, the size of a smaller fingerprint to be stored in DRAM, or the Bloom filter vector length (referred to as *summary vector* in [4]) for fast identification of non-stored chunks.

Xtremio's X-brick [17] is an example of an all-flash high-end large-scale contemporary storage appliance, according to Fig 1. 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, [2] 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 [4] and [20]). The proposed system uses two types of hash fingerprints and additional data structures to maintain the additional level of indirection.

We see that while greatly improving storage efficiency and reducing cumulative number of writes, typical inline deduplication may increase system cost and energy consumption, and may limit the data throughput (number of input-output operations, or IOPS). The same conclusion was noted in [20].

C. In-ReCAM 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 Fig 8. Each data block in ReCAM storage is divided into $S=block\_size/ReCAM\_width$ 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.

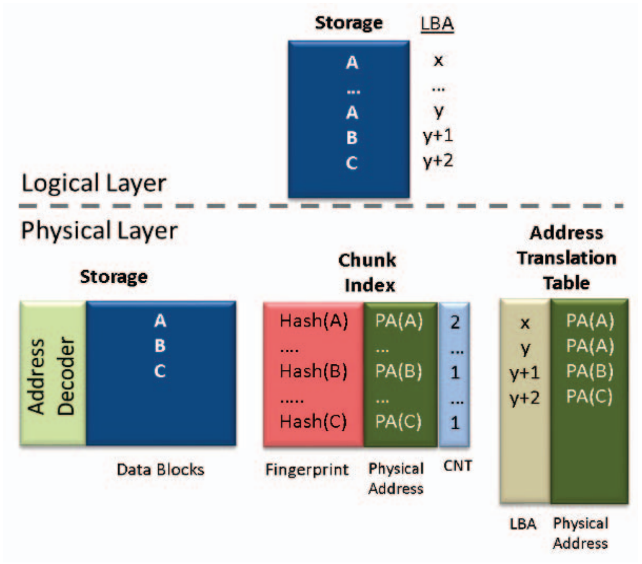

Fig 7. Conventional deduplication scheme after writing the following sequence of (data block, LBA): (A, x), (A, y), (B, y+1), (C, y+2). The storage, chunk index and address translation table reside in the physical layer.

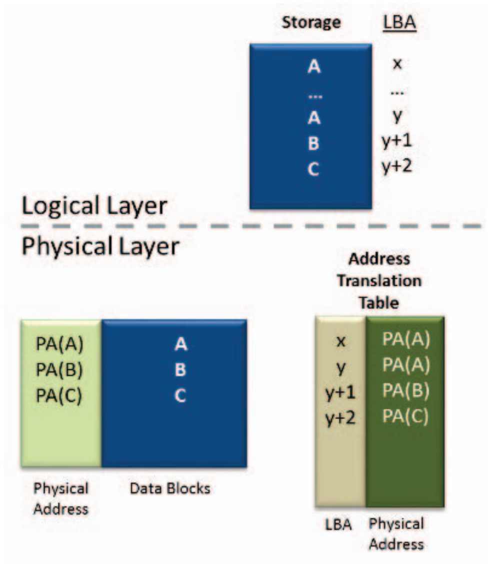

Fig 8. ReCAM based deduplication scheme, following the same sequence of writes as in Fig 7.

The values of block_start in all other ReCAM rows of the data block are zero.

During write, a new data block is compared (in parallel) against all data blocks stored in the ReCAM to search for a stored identical copy. This is achieved by a sequence of one single compare followed by S-1 continuous compare operations. During the single compare, $COMP_S$ is set (Fig 5a) and start_block bit column is masked-on, to enable comparison of only the first segment of each data block in the storage. During the following S-1 continuous compares, the result (TAG) of every consecutive compare is ANDed with the result of the previous compare. Thus, in each compare, only the rows matched in the previous compare are active, and the number of active rows drops progressively, significantly reducing the compare energy. The outcome of a series of S compare operations is signaled by MATCH (Fig 5b). MATCH='1' means that the new data block is a duplicate and should not be stored. One cycle is required for each single compare, resulting in S cycles for discovering the block is a duplicate.

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. Each segment is written in one cycle to storage.

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). Writing the LBA and PA can take 1-2 cycles each, depending on $ReCAM\_width$. 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 (done in 1-2 cycles). 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 one cycle 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 Fig 6). This step takes a total of 3 cycles. In the second step, the PA (retrieved at the first step) is searched again in the address translation table, which takes 1-2 cycles; if MATCH returns '0', it means that the deleted block has no duplicates. In this case, it is deleted from the ReCAM storage in S cycles. Overall, delete operation also takes $O(S)$ cycles.

IV. Simulation

We simulate the ReCAM based deduplication using the cycle-accurate CAM simulator introduced in [19], 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).

We compare our ReCAM deduplication implementation with opendedup [14], 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, we use blocks of 1KB, 2KB, 4KB and 8KB. We run 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, we use the file system benchmark IOzone [11]. 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 [7] was used for measurements.

As demonstrated by [20], 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 Fig 9. The measured throughput of opendedup is also presented in Fig 9 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 [17], similar to the simulated performance of opendedup.

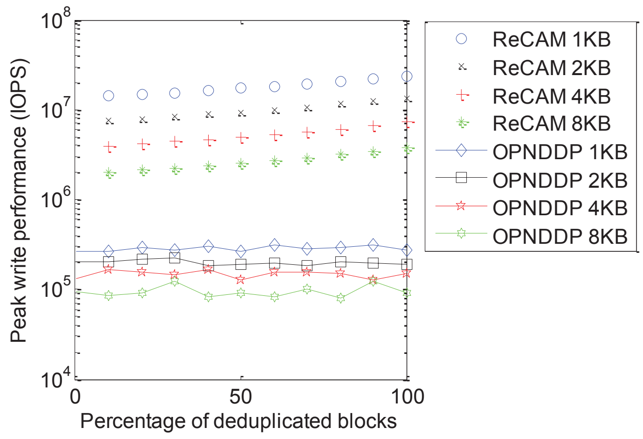

Fig 9. Write performance for different block sizes *vs*. percentage of deduplicated blocks, for data blocks of 1KB, 2KB, 4KB and 8KB (OPNDDP = Opendedup)

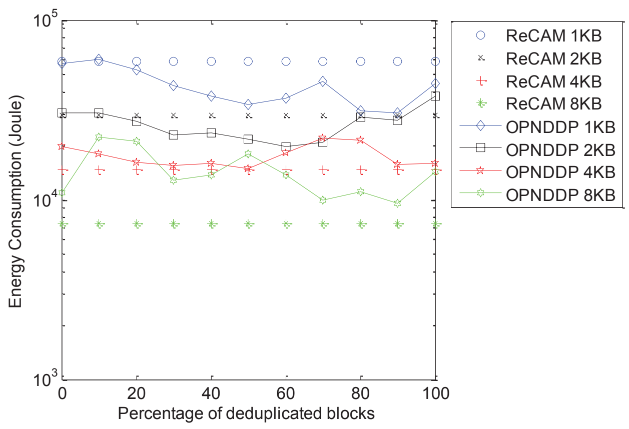

Fig 10. Deduplication energy for different block sizes vs. percentage of deduplicated blocks, for data blocks of 1KB, 2KB, 4KB and 8KB while writing 50GByte of data

The simulated energy consumption of ReCAM based deduplication as a function of percentage of deduplicated blocks is presented in Fig 10.

To understand the energy benefits of continuous compare, we simulate the energy consumption without deactivating the ReCAM rows that have mismatched in previous compares (using single rather than continuous compare, with $COMP_S$ input in Fig 5(a) set to '1'). This results in much higher simulated energy consumption.

The measured energy consumption of opendedup (including the SSD energy consumption) is also presented in Fig 10 for comparison. The energy consumption of ReCAM based deduplication is in the same range (slightly higher for smaller blocks, lower for larger blocks).

V. Conclusions

This paper explores deduplication in a novel solid state storage based on Resistive Content Addressable Memory (ReCAM). ReCAM enables storage with *in-situ* processing capabilities. We show that ReCAM-based in-storage deduplication implementation can provide up to 100x higher throughput than typical CPU and DRAM based deduplication schemes, while consuming similar or lower energy.

More generally, ReCAM may enable additional types of applications that combine storage with processing, such as in smart SSD [2] and cache in large-scale hybrid storage systems [4][8].

Acknowledgment

(appropriate acknowledgements will be added to the final paper.)

References

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

[2] Chen, F., Luo, T., & Zhang, X. "CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives." *FAST*. Vol. 11. 2011.

[3] 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.

[4] Debnath, Biplob K., Sudipta Sengupta, and Jin Li. "ChunkStash: Speeding up inline storage deduplication using flash memory", USENIX Annual Technical Conference, p. 16, 2010.

[5] Do, J., et al. "Query processing on smart SSDs: Opportunities and challenges," in Proceedings of SIGMOD'13, ACM, pp. 1221–1230

[6] Fujii H., et al. "x11 performance increase, x6. 9 endurance enhancement, 93% energy reduction of 3D TSV-integrated hybrid ReRAM/MLC NAND SSDs by data fragmentation suppression", IEEE VLSIC 2012.

[7] Intel Performance Counter Monitor. Intel Corporation. URL: www.intel.com/software/pcm

[8] Kim, Y., Gupta, A., Urgaonkar, B., Berman, P., & Sivasubramaniam, A., "HybridStore: a cost-efficient, high-performance storage system combining SSDs and HDDs." IEEE 19th International Symposium on MASCOTS, 2011.

[9] Kvatinsky, S., Friedman, E. G., Kolodny, A., & Weiser, U. C. "TEAM: threshold adaptive memristor model", IEEE Transactions on Circuits and Systems I, 2013.

[10] Myoung-Jae L., et al. "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5− x/TaO2− x bilayer structures." Nature materials 10.8 (2011): 625-630.

[11] Norcott, W. D., & Capps, D., "Iozone filesystem benchmark." URL: www.iozone.org 55 (2003).

[12] Patel, R., Kvatinsky, S., Friedman, E. G., & Kolodny, A., "Multistate Register Based on Resistive RAM", IEEE Transactions on VLSI, 23(9), pp. 1750-1759, 2015.

[13] Sang-Woo J., et al. "BlueDBM: an appliance for big data analytics." ISCA-42, 2015.

[14] Silverberg S., "Opendedup SDFS." (2010).

[15] Stevenson, J. P., Firoozshahian, A., Solomatnikov, A., Horowitz, M., & Cheriton, D., "Sparse matrix-vector multiply on the HICAMP architecture." Proceedings of the 26th ACM international conference on Supercomputing, 2012.

[16] Ventra, M. D., Pershin, Y. V., & Chua, L. O. "Circuit elements with memory: memristors, memcapacitors, and meminductors," Proceedings of the IEEE 97(10), pp. 1717-1724, 2009.

[17] X-Brick tech spec. URL: https://www.emc.com/collateral/data-sheet/h12451-xtremio-4-system-specifications-ss.pdf

[18] Yavits, L., Kvatinsky, S., Morad, A., & Ginosar, R., "Resistive Associative Processor", IEEE Computer Architecture Letters, 14(2), pp. 148 – 151, 2015.

[19] Yavits, L., Morad, A., & Ginosar, R., "Computer Architecture with Associative Processor Replacing Last Level Cache and SIMD Accelerator", IEEE Transactions on Computers, 2013

[20] Zhu, B., Li, K., & Patterson, R. H., "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," in 6th USENIX Conf. File and Storage Technologies (FAST), 2008