DRAM-CAM: General-Purpose Bit-Serial Exact Pattern Matching

Lingxi Wu, *Member, IEEE,* Rasool Sharifi, *Member, IEEE,* Ashish Venkat, *Member, IEEE* and Kevin Skadron, *Fellow, IEEE*

Abstract—Exact pattern matching is a widely used kernel in many applications. A DRAM-based processing-in-memory (PIM) architecture, Sieve, was recently proposed to alleviate the bottleneck stage of sequence matching in genomics. This paper observes that other exact-pattern-matching-intensive workloads can benefit from a similar architecture. We extend Sieve with several cost-effective modifications, such as a population count logic, chip-level parallelism support, and a hardware data transposition unit, making a general-purpose DRAM-CAM and key-value store that outperforms both CPU and various PIM solutions.

1 Introduction

Exact pattern matching is a widely used computation kernel. A common software implementation is a lookup or hash table, but large data sets do not fit into the last-level cache (LLC) and exhibit poor locality. Furthermore, the computation per pattern lookup is also too small to mask the high memory-access latency, resulting in frequent processor stalls [1], making the task memory-bound. An alternative is a coarse-grained index that fits in the LLC, in which a key is mapped to a bucket of potential matches, with linear or binary search within a bucket. However, our prior results [1] show poor temporal locality in which buckets are accessed.

To address these limitations, data-centric architectures leveraging content addressable memory (CAM) have been proposed [2], [3]. This paper describes how to implement CAM functionalities inside *DRAM*, which offers several advantages over non-volatile memory (NVM) and SRAM alternatives. Even a highly compact 3T3R PCM NV-CAM cell is over 3X larger than a DRAM cell, and SRAM is much less dense and more power-hungry.

The proposed architecture, DRAM-CAM, is built on Sieve [1], a recently-proposed processing-in-memory (PIM) key-value accelerator designed originally for massively-parallel *k*-mer matching (searching for short DNA sequence patterns of size *k*), but more generally suitable for a variety of key-value applications. Sieve provides an average of 326X/32X speedup and 74X/48X energy savings over multi-core-CPU/GPU baselines for *k*-mer matching, using a column-wise data layout for patterns, allowing element-parallel, bit-serial matching (each bit position is checked across a large number of bitlines, i.e. data items). Sieve and SIMDRAM [4] showed that this offers better matching throughput than a traditional, row-wise data layout. This allows Sieve to integrate low-overhead bit-wise logic inside row buffers, coupled with subarray-level parallelism, to simultaneously compare thousands of patterns in each row cycle without incurring expensive data movement. Although a similar in-situ approach has been explored in prior proposals such as Ambit [5] and SIMDRAM, their multi-row activation-based approach, which relies on charge-sharing, is more energy-intensive and slower than the sequential single row activation and digital comparisons employed in Sieve [1], due to the overhead of row-copy operations involved to set up operand rows in the "Bitwise" group for pattern matching [4], [5]. Furthermore, column-wise data layout and single-row activation allow Sieve to exploit an Early Termination Mechanism (ETM) that prevents unnecessary DRAM row activation if all columns have encountered a mismatch. Therefore, even if the slow multi-row activation mechanism is replaced with rapid timing-constraint-violating DRAM commands that leave multiple rows open to perform fast row-wide logic operations, as described in ComputeDRAM [6], Sieve still performs better by a large margin due to the benefit of ETM, which is not possible in a row-wise data mapping. Furthermore, combining ComputeDRAM with a vertical data layout is unlikely to outperform Sieve, because of the much larger overhead of setting up queries for the target subarrays [1].

In this paper, we add several features that enable a wider range of pattern-matching applications, including population-count logic to count matches (in Sieve, a given *k*-mer will have at most one match), hardware support for faster transposition of data into the column-wise format, and optimizations for greater parallelism. The evaluation shows that DRAM-CAM provides up to three orders of magnitude of speedup and energy reduction over the CPU baselines, and on average outperforms the closest PIM competitor by 3.7X.

2 Background

Sieve Architecture. This work focuses on Sieve Type-3, shown in Fig. 1, which leverages subarray-level parallelism, with logic integrated into each local row buffer. Bit cells within each subarray are divided into three regions (Fig. 1 (c)). No physical modification is made to the bit cells. Region-1 stores the interleaved reference and query patterns (keys), transposed onto columns. Region-2 (optional) stores the offsets to the starting address of each reference pattern's payload. Region-3 stores the payloads (values). Data in Regions 2/3 are stored in row-major format.

Region-1 is further broken down into smaller pattern groups and a batch of query patterns are replicated in each pattern group in the middle (red in Fig. 1 (c)). Note

• *L. Wu, R. Sharifi, A. Venkat, and K. Skadron are with the Department of Computer Science, University of Virginia .*
E-mail: {lw2ef, as3mx, venkat, skadron}@virginia.edu

Manuscript received xxx, xx; revised xxx, xx.

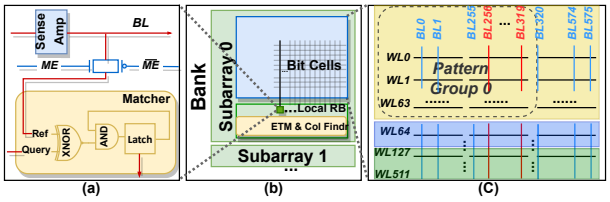

Fig. 1: Sieve illustration. (a) Matcher logic. (b) Matching logic in each row buffer. (c) Subarrays are partitioned into three regions for patterns, payload offsets, and payloads.

each batch contains different queries, and individual queries are replicated across pattern groups in a subarray. Pattern groups are needed because the transmission delay of long wires prevents us from broadcasting a query bit to all a row's matchers. The exact size of a pattern group is determined by the number of matchers that a query bit can reach in one DRAM row cycle, and the number of query patterns per batch is determined by the chip's prefetch size. After a batch of query patterns finishes matching in a subarray, they are replaced by a new batch via write commands. Batching also minimizes the overhead of PCIe transactions and amortizes the DRAM write commands for setting up queries to all pattern groups. To avoid broadcasting queries to every subarray, Sieve utilizes a coarse-grained index on the CPU that maps query patterns to a candidate subarray, so only one subarray is consulted for each individual query.

The matcher circuit, shown in Fig. 1 (a), is made of an XNOR gate, an AND gate, and a one-bit latch, and operates on the digital output of a sense amplifier. The XNOR gate checks if the reference bit and the query bit are equal. The bit latch stores the result of the XNOR operation, indicating if a reference and a query have matched exactly up to the current bit position. The value in each bit latch is initialized to 1 (default to match). The AND gate compares the previous matching result stored in the bit latch with the current result from the XNOR gate and updates the bit latch accordingly, with the running match progress. Further details are in [1]. The Early Termination Mechanism (ETM) interrupts further row activation by checking if the entire row of latches is storing zeros. Matching continues if at least one latch stores 1.

Exact Pattern Matching Workloads. We select the same applications from [2] (Table 1), minus *Vortex*, which is deprecated and not open source, and *ReverseIndex*, which does not map well to DRAM-CAM. *String Match* processes a key file of strings and a file of hashed (encrypted) strings to find which keys occur in the encrypted file. *Histogram* counts frequencies of pixel values in the RGB channels of a bitmap. *Word Count* generates the frequency for each word in a text file. *Apriori* performs associative rule mining, building a candidate itemset, and counts itemset frequencies in a transaction database. Our results (Sec. 4) suggest DRAM-CAM's bit-serial nature favors workloads with shorter patterns (several hundred bits or less). The Reverse Index is a "bad fit" because each pattern (URL links) is too long for DRAM-CAM to handle. DRAM-CAM also favors kernels that can issue large batches of pattern search requests to fully leverage parallelism in the DRAM hierarchy.

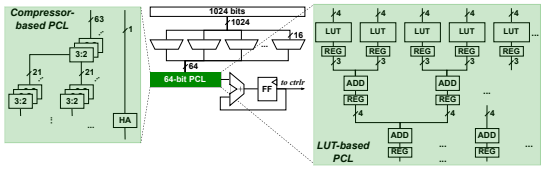

Fig. 2: Population count logic.

3 DRAM-CAM Architecture

DRAM-CAM retains the core architectural designs of Sieve and serves as a PCI-attached accelerator with an offload model. We introduce several hardware components and runtime optimizations pertaining to DRAM-CAM.

Population Count Logic (PCL). A population count logic (PCL) unit returns the total number of matches for each query. The PCL accumulates the number of ones from the row of latched bits at the subarray level, then aggregated at the controller level for the total number of hits. In many use cases, aggregating hits for each query accounts for nearly the entirety of the workload. Integrating PCL at the subarray level is difficult since it needs to process a large bit vector in a timely fashion with minimal hardware overhead.

Our PCL design, shown in Fig. 2, works on 1024 bits by processing chunks of 64 bits. To count the 1s in a group of 64 bits, we explore two options: lookup table (LUT) and Wallace-tree-architecture compressor tree circuit [7]. The first level of the LUT-based PCL requires 16 four-input LUTs that take four bits from the latches and output the number of ones in binary. The remaining levels of this PCL are like an adder tree, aggregating ones from all LUTs. One optimization is to insert registers between levels to form a pipelined PCL, which reduces latency but increases area and power overhead (see Table 2). The compressor-tree PCL is based on [7], which uses 57 3:2 compressors and 8 half-adders in ten cascading stages. The 3:2 compressor has the same truth table as a full adder. Each compressor processes 3 bits, outputting the number of ones in its sum and $carry$ bits as $sum + 2 \times carry$.

Data Transposition Unit (DTU). If the reference patterns are reused across different executions, transposing the data in software is a one-time cost amortized over a long period of use. However, some workloads require input data to be transposed on the fly and written to the DRAM-CAM prior to matching, which places the data transposition operation on the critical path. We integrate a simplified data transposition unit (DTU) from SIMDRAM [4] into DRAM-CAM. The DRAM-CAM DTU requires only one 4KB SRAM transposition buffer. DRAM-CAM's DTU works at a rate of transposing one cache line worth of data (512 bits) in one cycle. We estimate that such hardware DTU is 381.3X faster than a software one (estimated using a modified DRAMSim2; see Sec. 4), and adds an insignificant (<0.1%) amount of execution time. Once the CPU with the help of a dedicated runtime environment (future work) instructs the DRAM-CAM device to load the reference pattern sets (e.g., image data for Histogram) from disk using DMA, data first arrive at this SRAM buffer, then transposed row by row by simple custom logic and written into DRAM-CAM using DRAM commands.

Chip-level Parallelism (CLP). Sieve chips in a rank respond to queries in a lockstep manner due to the shared chip select signal (CS), a design carried over from a tradi-

TABLE 1: Mapping exact matching kernels onto DRAM-CAM

Benchmark	Index	ETM	PCL	DTU	CLP	Input	Payloads	DRAM-CAM patterns	DRAM-CAM computing
String Match	Yes	Yes	No	No	Yes	Key file	None	Encrypted file	Search keys in the encrypted file
Histogram	No	Yes	Yes	Yes	No	8-bit pixels	None	Image binary pixel values	Aggregate hits for each pixel pattern
Word Count	No	Yes	Yes	Yes	No	Unique words	None	Words from text file	Aggregate hits for each input word
Bitcount	Yes	No	No	No	Yes	32-bit binaries	Num of set bits	32-bit binaries	Retrieve number of set bits
Apriori	No	No	Yes	No	No	Itemsets bit vectors	None	1-hot encoded transactions	Check if transactions contain an itemset

tional DDR architecture. Chip-level parallelism (CLP) can be achieved to a certain degree by providing each chip with a dedicated chip select wire. Note this solution does not make each chip truly autonomous, because the data line (DL) still has to be shared inside a rank due to limited high-frequency data pin count, which is prohibitively expensive to scale. DRAM-CAM chips receive their input queries once the shared DL is available, thus only pattern matching is parallelizable, while query input is serialized. The downside of CLP is that the number of entries in the index table will be increased since chips need to be indexed. However, the granularity of the indexing scheme can be adjusted if needed to keep the index within L2 capacity.

Runtime Optimizations. To leverage the parallelism of DRAM, we want to leverage as many subarrays as possible, which reduces congestion and maximizes parallelism. DRAM-CAM starts offloading patterns by choosing a random subarray for pattern storage, and after it is filled with $subarray\_width$ patterns, randomly chooses the next subarray from a different channel/rank/bank for pattern placement. Further optimization is to replicate small reference pattern sets multiple times by storing them in unused subarrays, which allows applications to use them for greater parallelism. To support this optimization, the main changes occur in the index table, where one additional busy bit for each entry is needed to indicate if the subarray is currently being used or not. When a new query arrives, the index table chooses a subarray whose busy bit is 0 that stores the same reference patterns. If all candidate subarrays are busy, we choose a random one to wait upon. Pattern distribution (PD) offers 22% to 7.4X speedup while pattern replication (PR) offers 4X to 29.4X speedup over an unoptimized pattern storage scheme (Fig. 3). PR generally offers better performance than PD, because it allows DRAM-CAM to utilize subarray-level parallelism on top of bank-level parallelism.

Application Mapping. While some kernels map to DRAM-CAM naturally, such as *String Match (SM)* and *Bitcount (BC)*, others are not so straightforward and require algorithmic changes. *Histogram (HG)* and *Word Count (WC)* differ most from their CPU counterparts, where the input images or text files are transposed into DRAM-CAM prior to the matching process. Then a standardized input set such as all 8-bit pixel patterns or unique English words are passed as input to aggregate hits. For *Aprior (AP)*, the entire transaction database is transcribed using one-hot encoding, with each column representing a transaction and each row representing an item. To check if a candidate itemset is a subset of a transaction, the i^{th} row corresponding to the i^{th} 1 of the bit vector is opened. Table 1 shows more details of mapping each kernel onto DRAM-CAM. One interesting discovery is that the best way to utilize ETM in natural language (e.g., *Word Count*) is to match the patterns backward, due to the significant prefix overlapping.

TABLE 2: Population Count Logic Characteristics

	LUT no Pipeline	LUT Pipeline	Compressor Tree
Area (nm^2)	201	554	148
Delay (ns)	0.76	0.34	0.84
Power (uW)	0.03	0.06	0.02

TABLE 3: Workstation Configuration

CPU Model	Intel(R) Xeon(R) E5-2658 v4
L1 /L2 /L3 $	32 KB / 256 KB / 35 MB
Main Memory	DDR4-2400MHz (32 GB/2 Chan)

4 EVALUATION

The experimental setup and evaluation methodology are identical to those of [1]. The baseline DRAM energy is estimated by feeding memory traces to DRAMSim2, configured to match our workstation. The CPU energy is measured using the Intel PMC-power tool, then scaled down by 30% to exclude the interference from other system components, consistent with the methodology from DRISA [8] For application performance modeling, we use a trace-driven, in-house simulator that has a custom DRAMSim2 as the front end. We use the Micron DDR4 4Gb 8B x16 chip as the building block. We assume a pipelined implementation of DRAM-CAM, where the host (CPU) performs preprocessing and post-processing, while DRAM-CAM is responsible for pattern matching. We use Verilog to implement different versions of the population count circuit. Then, we estimate power/area/latency using Synopsys in 90nm. Finally, we use scaling factors from [9] to scale down results to 22nm. See the original Sieve paper [1] for more methodology details. Table 3 reports the CPU hardware setup. We measure the portion that can be offloaded to DRAM-CAM, which is 98.97% for *String Match*, 75.88% for *Histogram*, 92.99% for *Word Count*, 100% for *Bitcount*, and 63.00% for *Apriori*. Table 2 summarizes performance characteristics for PCL. The compressor-based PCL has lower area and power, while the pipelined LUT-based PCL is the fastest. We propose to fit PCL in the center strip of each DRAM chip, and each PCL is time-shared among subarrays of a bank. This setup increases the latency slightly. Decoupling CS signals to enable chip-level parallelism requires negligible hardware changes. For the data transposition unit, the primary component is a 4KB SRAM buffer. We estimate its area to be 0.015 mm^2, and it consumes 2.22 uW.

Performance improvement over CPU. Figure 3 reports the speedup and energy saving over a CPU baseline of various DRAM-CAM configurations, including the performance of our unoptimized (UNOPT) setup, which is closest to the original Sieve architecture while enabling these other applications, and the benefit of three optimizations: pattern distribution (PD), pattern replication (PR), and chip-level parallelism (CLP). For applications that need PCL, we model LUT with the pipeline. The optimizations are highly

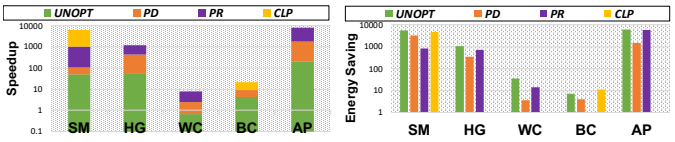

Fig. 3: Comparison with CPU baseline.

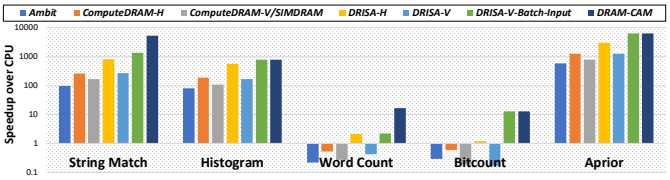

Fig. 4: Comparison to other in-DRAM accelerators.

effective when the reference pattern set is small, because it can be distributed and replicated many times to leverage the massive internal parallelism of DRAM. Additionally, chip-level parallelism offers approximately 2.9X speedup when applicable, but it does not help when a query needs to visit all subarrays to aggregate hits. *String Match (SM)* shares the most similarities with *k*-mer matching and benefits the most from such an accelerator. *Word Count (WC)* only experiences modest speedup. In fact, UNOPT is 1.5X slower than CPU. There are two reasons: (1) long string patterns and high match rates cause frequent and long sequences of DRAM row openings, and (2) a large input set (reference patterns) that limits optimization potential. This is in contrast to *Apriori (AP)*, which also stores large reference sets and long patterns. but only opens a few rows (<10). DRAM-CAM outperforms *Bitcount (BC)* on the CPU, because it stores a much larger lookup table (32-bit vs. 8-bit patterns).

The baseline DRAM-CAM (UNOPT) tends to show the best energy efficiency because the dynamic power consumption of DRAM-CAM depends on the number of banks that are used for pattern matching, and the UNOPT setup uses only a small percentage (0.7% $\sim$ 50%) of banks, resulting in up to 126.4X lower power than the CPU baseline. There is a tradeoff between greater parallelism and higher energy. PD shows worse energy saving than UNOPT, even though it offers better speedup, because UNOPT uses all subarrays of a smaller set of banks, but leverages subarray-level parallelism (SALP) to its full potential, thus making up the performance loss due to increased bank conflicts. On the other hand, PD usually utilizes fewer subarrays from a larger set of banks, resulting in sublinear speedup w.r.t. bank count. PR usually shows better energy saving than PD, except for the SM benchmark, by exploiting more SALP. SM has a small input set, and PD utilizes only two banks (low power). PR offers 16X speedup, but needs 128 banks, However, HG, WC, and AP have larger data sets, and PD requires the same amount of banks as PR, meaning they have similar dynamic power consumption. Since PR significantly reduces the execution time of those benchmarks, it offers better energy efficiency for those benchmarks. Finally, CLP increases power consumption minimally, but the performance improvement is significant, so the energy savings approach or surpass UNOPT.

Comparison to alternative PIM designs. We compare DRAM-CAM with several prior DRAM-based in-situ proposals. Fig. 4 reports the results, and the performance numbers are normalized to CPU baselines. We assume all indispensable architectural features such as population count logic are enabled for all architectures, even though they are missing from some prior works, and all appropriate hardware and software optimizations proposed in this work are equally applied to prior works. *Ambit* adopts the traditional horizontal data layout (row-major) and triple-row-activation (TRA) based logical operation (XNOR) for pattern matching. *ComputeDRAM-H* reduces TRA latency by half but retains the horizontal data layout. *ComputeDRAM-V/SIMDRAM* switches to vertical data layout (column-major) with TRA. In addition to charge-sharing based in-situ accelerators, we also simulate variations of DRISA, which combines analog bit-line functionality with digital logic in the row-buffer. *DRISA-H* uses horizontal data layout while *DRISA-V* uses vertical, and *DRISA-V-Batch-Input* is *DRISA-V* but utilizes Sieve-style batched queries.

TRA-based pattern matching is inherently slow, even with the modified version proposed in ComputeDRAM. Each row-wide comparison takes multiple DRAM cycles, whereas in-row-buffer logic takes only one. Moreover, exact matching needs to XNOR operand rows, which requires two TRA operations. Second, while for general-purpose computing, vertical data layout has shown better performance, for exact matching, horizontal data layout is better, because each query only needs to populate one row, whereas vertical data layout has to populate a two-dimensional block of bits ($subarray\_width \times query\_bit\_length$) for each query in order to support the bit-serial matching. Third, PIM generally favors short patterns over longer patterns, and this is especially true for column-major layouts. Fourth, DRAM-CAM outperforms DRISA because it has a more efficient way of setting up queries, plus early termination (ETM). Note also that GRIM-filter [10], an HMC PIM for short-sequence DNA alignment, may also support exact pattern matching, an interesting direction for future work.

5 Conclusions

This work develops general CAM functionality inside DRAM, making it capable of accelerating a wide range of exact pattern-matching workloads while achieving significant energy reduction over the CPU, with up to 6217X speedup and 5888X energy savings.

Acknowledgements

This work was supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA.

References

[1] L. Wu *et al.*, "Sieve: Scalable in-situ dram-based accelerator designs for massively parallel k-mer matching," in *ISCA*, 2021.
[2] Q. Guo *et al.*, "A resistive TCAM accelerator for data-intensive computing," in *Micro*, 2011.
[3] Q. Guo *et al.*, "AC-DIMM: Associative computing with STT-MRAM," in *ISCA*, 2013.
[4] N. Hajinazar *et al.*, "SIMDRAM: A framework for bit-serial simd processing using dram," in *ASPLOS*, 2021.
[5] V. Seshadri *et al.*, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[6] F. Gao *et al.*, "ComputeDRAM: In-memory compute using off-the-shelf DRAMs," in *MICRO*, 2019.
[7] R. Ramanarayanan *et al.*, "Combined set bit count and detector logic," U.S. Patent US8214414B2 Sep. 2008.
[8] S. Li *et al.*, "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.

[9] A. Stillmaker, Z. Xiao, and B. M. Baas, "Toward more accurate scaling estimates of cmos circuits from 180 nm to 22 nm," 2012.
[10] K. JS *et al.*, "Grim-filter: Fast seed location filtering in dna read mapping using processing-in-memory technologies,"