TCAM-SSD: A Framework for Search-Based Computing in Solid-State Drives

Ryan Wong[†] Nikita Kim[‡] Kevin Higgs[†]
Engin Ipek[◇] Sapan Agarwal[▲] Saugata Ghose[†] Ben Feinberg[▲]

[†]Univ. of Illinois Urbana-Champaign [‡]Carnegie Mellon Univ. [◇]Samsung [▲]Sandia Nat'l. Laboratories

Over the past decade, the amount of data generated and consumed by modern applications has grown exponentially [6], which induces a large amount of data movement between processing elements, main memory, and storage. This data movement has become a major bottleneck in modern systems, as it consumes large amounts of energy and results in significant performance penalties [4]. *Processing-in-memory* (PIM) architectures provide a solution to alleviating data movement, by performing data processing closer to (i.e., *near*) or directly inside (i.e., *using*) memory arrays. While a large amount of research has explored processing-near-memory (PNM) and processing-using-memory (PUM) for main memory subsystems, they do not alleviate data movement to storage for applications with very large datasets.

NAND-flash-based *solid-state drives* (SSDs) provide an opportunity to perform PIM across potentially terabytes of data per drive. Unfortunately, existing PNM and PUM solutions for main memory cannot be directly applied in SSDs, because (1) the latency–bandwidth trade-off of SSDs is significantly more skewed towards bandwidth in SSDs, (2) key elements of the SSD architecture (e.g., firmware-based data management, available parallelism in NAND flash memory chips, flash channel controller) add significant complications compared to bank accesses in main memory, and (3) it is not easy to scale up the number of PIM logic elements to match the orders-of-magnitude increase in available data. Existing works propose either (1) PNM solutions that eliminate host–device communication but are still bottlenecked by the flash channel controllers (e.g., [1, 7]), or (2) PUM primitives that perform bitwise operations *inside* NAND flash chips but do not study integration with the SSD firmware and host (e.g., [2, 5, 9]).

In this work, we propose TCAM-SSD, a new framework for efficient in-SSD search-based computing. TCAM-SSD utilizes the in-memory search (IMS) primitive [9], which treats a NAND flash cell array as a bulk *ternary content-addressable memory* (TCAM). TCAM-SSD builds a full framework on top of IMS, providing (1) concurrent support for block reads and for TCAM search operations, (2) a mechanism to collect and retrieve search results from the TCAM operation, (3) the ability to coherently modify searchable data, and (4) a NVMe 2.0 compliant interface for user applications to invoke search operations. TCAM-SSD requires only lightweight changes to the peripheral circuitry of NAND flash arrays, and modest firmware changes to support content-based operations.

IMS Primitive. A single IMS primitive can perform a parallel search across all searchable data elements in a single NAND flash array. The IMS primitive requires data to be transposed compared to the conventional layout: instead of storing the bits that belong to the same data element on the same wordline (i.e., horizontally), IMS stores the bits of the data element on the same *bitline* (i.e., vertically). In this transposed layout, a single bit of data is represented by two adjacent NAND flash cells, which share the same bitline and sit in adjacent rows to one another. IMS uses these two cells to store a bit value 0 (Figure 1a), a bit value 1 (Figure 1b), or a bit value

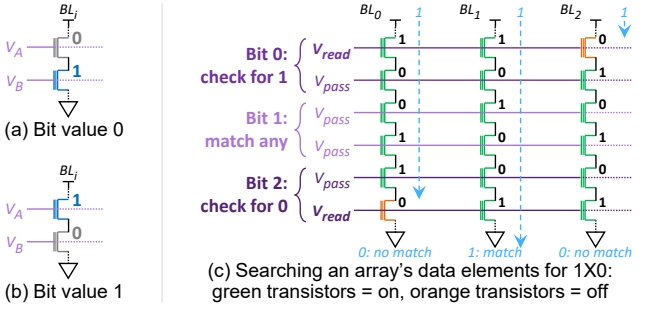

Figure 1: Associative search in a NAND flash array.

X (representing a bit that will match either a 0 or a 1; i.e., a bit that can be ignored). Each cell continues to use the same V_{th} states as conventional SLC cells.[1] When an IMS primitive is performed, voltages corresponding to the search data are applied to the NAND flash cell gates, as shown in Figure 1c. A bitline output of 1 means that the data element stored in the bitline is a match; otherwise, the bitline output is a 0.

TCAM-SSD. TCAM-SSD enables both baseline SSD functionality and IMS-based searches by splitting the SSD into two types of regions. *Data regions* perform read, program, and erase commands on horizontal data, identical to a conventional SSD. *Search regions* perform IMS-based commands on vertical data elements. For each data element in the search region, the data region contains a corresponding *data entry*, whose contents are application dependent. For example, if an application directly wants the value of the matching data element, its corresponding data entry contains a non-transposed replica of the value (e.g., to enable fast readout of matching data). If we want to implement a key–value store (KVS), each data element in the search region is a key, and its corresponding entry in the data region holds the value.

Figure 2 shows the front-end interface for TCAM-SSD. TCAM-SSD aims to use in-SSD search to eliminate two types of data movement required by conventional drive reads: (1) CPU–FE (front end), by enabling the SSD to return only matching pieces of data to the host CPU; and (2) FE–BE (back end), by performing IMS inside the NAND flash chip and reading out only matching pieces of data across flash chip channels. Applications interact with TCAM-SSD through drive-level commands that we introduce as extensions to the standard NVMe protocol. One of these commands allocates a new search region.[2] Our modified FTL performs block-level

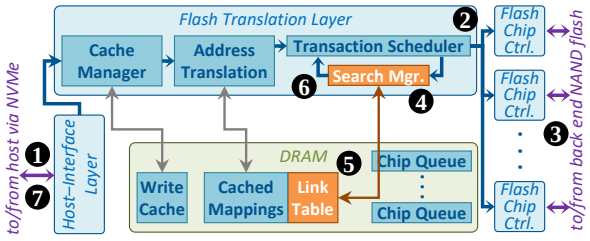

Figure 2: TCAM-SSD front end (new modules in orange).

Full Paper: https://arxiv.org/abs/2403.06938

SNL is managed and operated by NTESS under DOE NNSA contract DE-NA0003525.

[1] While TCAM-SSD can make use of multi-level cells, we use SLC, without loss of generality, to simplify our descriptions.

[2] Unlike IMS, which limits the data element length to the number of wordlines in an array, TCAM-SSD's firmware supports elements of arbitrary length.

allocation for the search region, and writes data elements vertically to the NAND flash chips. Additional commands update a data element, and append new data to a search region.

To perform a *ternary* search (i.e., a search where one or more bits may be set as don't care) over one or more search regions, the application issues a data search command over NVMe to the firmware (❶ in Figure 2). The transaction scheduler converts the data search command into chip-level *SRCH* commands (❷). When our new *SRCH* command is issued by the flash chip controller (❸), it uses per-wordline read reference voltages to represent each bit of the search key (or whether a bit is a don't care), and passes these voltages to the NAND flash blocks that contain the requested search regions requested. The NAND flash blocks, using modified peripheral circuitry to support per-wordline voltages, then perform IMS primitives to concurrently search through thousands of data elements in the block at once. The output of each bitline indicates whether the word stored along the bitline is a match.

Combined with block-level parallelism, a single *SRCH* command can search over tens of thousands of data elements simultaneously. The list of matches is returned to a search manager that we add to the firmware (❹), which uses a metadata table to decode where the specific data lies (❺). The firmware then schedules and issues read requests for only the matching data (❻), and the matching data is returned to the host (❼).

Evaluation. To evaluate TCAM-SSD, we develop a set of detailed analytical models to capture the low-level hardware and software modifications to the SSD, as well as all critical aspects of the system, including, NVMe initialization overheads, DRAM access times, and NAND flash access time. TCAM-SSD can optimize a wide range of application domains that employ search-based operations. To demonstrate the flexibility of our framework, we present three use cases: (1) using in-SSD record filtering to reduce data movement for online transaction processing (OLTP) in databases; (2) enabling high-throughput parallel search to accelerate database analytics; and (3) combining pointers and search operations to retrieve edge data for large graph analytics.

For OLTP, we run TPC-C on DBx1000 [11] with 1 M transactions and 3 M entries. TCAM-SSD is faster than CPU-only DBx1000 whenever a query needs to retrieve more than 3 pages from the SSD (Figure 3a). Furthermore, we show the CDF of total latency contributed by the queries as a function of disk pages fetched (Figure 3b). TCAM-SSD achieves a speedup of 60.9%, and reduces CPU–FE and FE–BE data movement by 92.3% and 77.0%, respectively.

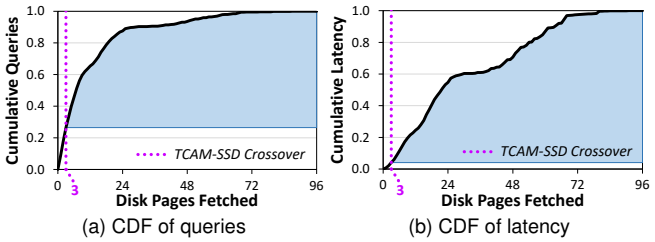

Figure 3: Cumulative distribution function (CDF) for TPC-C.

For database analytics, we evaluate modified versions of TPC-H queries [3, 10] that scan one 74 GB table from a 115 GB database populated using DBGEN [8]. We examine the effects of *selectivity* (the fraction of database records that match a query) and *locality* (how likely records are to share a NAND flash page). TCAM-SSD speeds up the scan operation for Query 1 by 18.3×, and for Query 2 by 17.1×, compared to a baseline database scan operation. This is because TCAM-SSD reduces both CPU–FE and FE–BE movement by 95%.

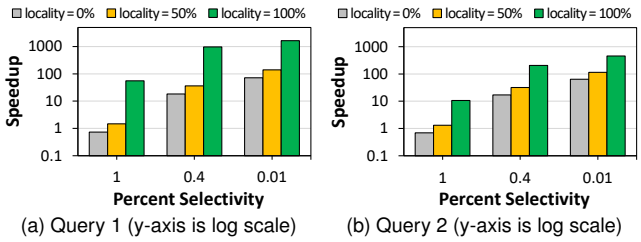

Figure 4: Speedup for analytical queries with TCAM-SSD, normalized to scan using a conventional SSD.

For graph analytics, we evaluate TCAM-SSD using the vertex access traversal trace for the SSSP algorithm on a collection of real-world and synthetic graphs ranging from 1.5 M edges (road-PA) to 1.3 B edges (mag240m). We compare TCAM to (1) in-memory index (IM), where the LBA of each vertex's edge list is stored in memory; and (2) out of memory (OOM), where both the graph index and edge list resides on disk. On average, TCAM-SSD reduces in-memory index metadata by 47.5%, compared to a baseline index with a 4 B pointer and 4 B of metadata (e.g., vertex weight) per entry (Figure 5). Overall, TCAM-SSD performs 12.8% better than OOM on average, as it avoids data movement costs between the CPU and the SSD.

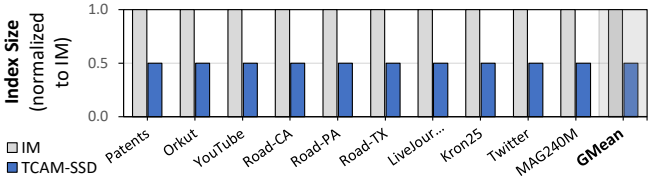

Figure 5: Graph index overhead, normalized to a conventional in-memory index.

Conclusion. We present TCAM-SSD, a framework for in-SSD search-based computing using NAND flash memory. With modest modifications to the NAND flash chips inside commodity solid-state drives (and with no modifications to the NAND flash array), we can enable highly-parallel ternary search operations. We show that for three use cases, TCAM-SSD can provide notable performance and data movement improvements for large dataset processing.

References

[1] Advanced Micro Devices, Inc., "Samsung SmartSSD," https://www.xilinx.com/applications/data-center/computational-storage/smartssd.html, 2021.

[2] C. Gao *et al.*, "ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs," in *MICRO*, 2021.

[3] B. Gu *et al.*, "Biscuit: A Framework for Near-Data Processing of Big Data Workloads," in *ISCA*, 2016.

[4] G. F. Oliveira *et al.*, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," *IEEE Access*, 2021.

[5] J. Park *et al.*, "Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory," in *MICRO*, 2022.

[6] D. Reinsel, J. Gantz, and J. Rydning, "Data Age 2025: The Digitization of the World From Edge to Core," IDC, white paper, 2018.

[7] Samsung Electronics Co., Ltd., "Samsung Electronics Develops Second-Generation SmartSSD Computational Storage Drive With Upgraded Processing Functionality," https://news.samsung.com/global/samsung-electronics-develops-second-generation-smartssd-computational-storage-drive-with-upgraded-processing-functionality, July 2022.

[8] Transaction Processing Council, "TPC-H DBGEN," https://github.com/electrum/tpch-dbgen.

[9] P.-H. Tseng *et al.*, "In-Memory-Searching Architecture Based on 3D-NAND Technology With Ultra-High Parallelism," in *IEDM*, 2020.

[10] L. Woods, Z. István, and G. Alonso, "Ibex—An Intelligent Storage Engine With Support for Advanced SQL Offloading," *VLDB*, 2014.

[11] X. Yu *et al.*, "Staring Into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores," *Proc. VLDB Endow.*, Nov. 2014.