Search-in-Memory (SiM): Conducting Data-Bound Computations on Flash Chip for Enhanced Efficiency

Yun-Chih Chen
TU Dortmund, Germany
yunchih.chen@tu-dortmund.de

Yuan-Hao Chang
Academia Sinica, Taiwan
johnson@iis.sinica.edu.tw

Tei-Wei Kuo
National Taiwan University, Taiwan
ktw@csie.ntu.edu.tw

***Abstract*—Large-scale data systems utilize indexes like hash tables and trees for efficient data retrieval. These indexes are stored on disk and loaded into DRAM on demand, where they are post-processed and analyzed by the CPU. This method incurs substantial data I/O, especially when optimizations like prefetching is used. This issue is inherent in the von Neumann architecture, where storage systems are dedicated solely to data storage, while CPUs handle all computations. However, data indexing primarily involves filtering tasks, which require only simple equality tests and not the complex arithmetic capabilities of a CPU. This inefficiency in the von Neumann architecture has led to a growing interest in in-memory computing, initially centered on DRAM.**

Recently, NAND flash-based in-storage computing has gained attention due to its ability to compute over larger working sets without requiring initial memory loading. In response, we propose the Search-in-Memory (SiM) chip, which minimally modifies an existing flash memory chip to allow it to conduct equality tests internally and send only relevant search results, not the entire data page. Specifically, we implement data filtering by using the existing logic gates in a flash memory chip's peripheral circuits for *bit-serial* equality tests, which processes all bits on a page simultaneously. Additionally, we introduce a versatile SIMD interface with two primary commands: *search* and *gather*, making SiM adaptable to different application scenarios. We use "Optimistic Error Correction" to efficiently ensure data accuracy. Our evaluations show that this new architecture could significantly improve throughput over traditional CPU-centric architectures.

I. Limitations in Existing Solutions

1) The pursuit of higher I/O bandwidth: Data systems have long used aggressive prefetching to monetize the enormous I/O bandwidth of modern SSDs. Under the hood, SSDs achieve their I/O bandwidth by retrieving data from multiple NAND flash memory chips (each further contains independent units activated in parallel). However, as chip density increases, it becomes more difficult to deliver enough power to drive up such degree of parallelism. Heat dissipation is a fundamental issue even in data centers. Increasing I/O bus clock frequency is another way to meet the ever-increasing demand for I/O bandwidth, yet at a cost of significant power consumption. Yu et al. [1] found that at 1.6GHz clock frequency, data transfer can consume up to 50% of the system's total power. A more sensible approach is to reduce unnecessary data transfer, allowing us to operate the I/O bus at lower clock frequencies enough to meet application requirements.

2) Reducing Data Transfer in Data Indexing: Various near-storage processing approaches aim to decrease data transfer between SSDs and the host OS by implementing computing within the FPGA or CPU of the SSD controller [2]. To further minimize data transfer, there are in-storage processing solutions that shift computation closer to where the data is stored [3], [4]. These solutions utilize the existing logic gates in the peripheral circuits of flash memory chips for logic operations. However, their strict data placement requirements make them unsuitable for tasks like indexing. These limitations motivate us to propose the Search-in-Memory chip, a novel in-storage computing solution.

II. The Design of Search-in-Memory (SiM)

SiM treats an index page as an array of fixed-width data rather than viewing it as opaque data. SiM features two primary commands: *search* and *gather*. The `search` command compares an input key with the data array in the index page, generating a matching bitmap. Then, the `gather` command uses this bitmap to extract specific chunks within an index page. This targeted approach reduces the bandwidth waste of full page-sized I/O transfers.

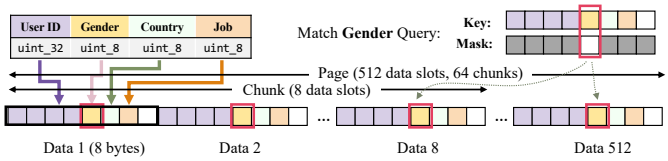

Fig. 1: Page format and data encoding

1) Command Format: As shown in Fig. 1, SiM recognizes a 4 KiB data page as an array of 8-byte data slots. When a `search` command is performed, the chip matches the 8-byte query key with these slots, returning a 512-bit bitmap as the match result. Every eight data slots is grouped into a *chunk* to serves as the minimal data transfer unit.

SiM's `search` command consists of the target page address and two 64-bit arguments: a query key and a mask that facilitates the comparison of specific bit ranges. In SiM-indexed relational database tables, where each row corresponds to an 8-byte key and data columns are encoded at specific bit ranges, the mask can isolate a specific column for matching. SiM's `gather` command resembles the `gather` SIMD instruction for the CPU: it uses a 64-bit index bitmap to indicate the

979-8-3503-4859-0/DATE24/© 2024 EDAA

desired chunks within an index page to read (a page contains 64 chunks).

2) Storage and Match Mode: SiM operates in two modes: *Match Mode* and *Storage Mode*. In *Storage Mode*, the flash memory chip is solely responsible for storing data. This mode inherits existing flash memory chip's functionality of multi-bit storage cell and high frequency I/O bus. On the other hand, *Match Mode* prioritizes efficient data retrieval. It stores only one bit per cell to ensure data reliability, and the I/O bus operates at a lower clock rate (thanks to reduced data transfer, page read latency is not degraded). SiM utilizes *Match Mode* to accelerate indexing operations while employing *Storage Mode* when writing new data and performing background maintenance.

3) Hardware Modifications: SiM utilizes the `XOR` gate in NAND flash memory's page sensing circuit for on-chip bit matching without transmitting the full page to the SSD controller. SiM repurposes the Failed Bit Counting (FBC) circuitry to determine if any bit in a 8-byte data slot mismatches the query key. An `OR` gate is added to each page sensor to toggle between *Storage Mode* and *Match Mode*.

4) On-chip Matching: Data stored in NAND flash memory can contain errors, which are typically corrected after the data is loaded to the SSD controller. In order to perform on-chip matching, we take the optimistic approach of sampling a few bytes at the start of the page for errors. A verification header is inserted when a page is written. At the start of the on-bit matching process, only the verification header is transmitted to the controller, leaving the rest of the data page in flash memory. If the verification is successful, the page is assumed to be error-free. Otherwise, the page is sent to the controller for further correction. Our optimistic error correction approach optimizes the common case of error absence in single-bit-per-cell pages while also providing a fallback solution for the rare cases. In addition to the verification header, we assign a 4-byte ECC parity to each chunk. This is checked in the controller during the execution of a `gather` command.

5) Host OS Integration: We implement a deadline-based command scheduler in the host OS to execute data matching requests at batch while ensuring bounded queuing delay. Each command is assigned a deadline upon submission. The scheduler holds the submitted commands in a queue until their respective deadlines expire. At that point, the scheduler searches for other commands in the queue that target the same page and submits them together as a batch.

III. Performance Evaluation and Results

We simulated SiM on the Amber emulator [5][1]. We evaluated the performance differences between SiM's integration in an external hash table and conventional CPU-centric architectures under read-intensive workloads with various query skewness. The CPU-centric architecture relies on host OS's page cache to cache frequently-accessed pages while the SiM-based hash table directly submit `search` commands to search for a target key in a page-sized hash bucket. The search commands are submitted to SiM SSD in batch through the deadline scheduler's arbitration.

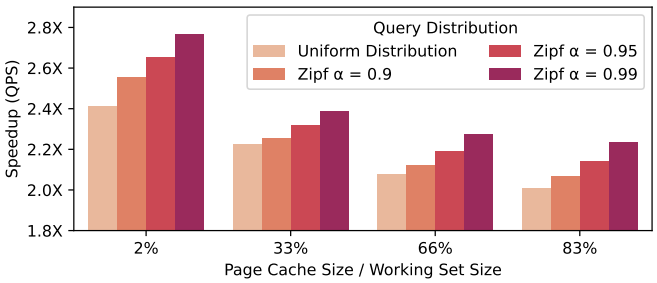

Fig. 2: Speedup comparison

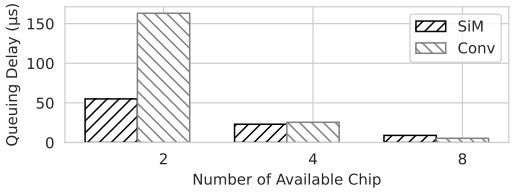

Fig. 3: Queuing Delay Comparison

The CPU-centric architecture requires sufficient amount of cache to achieve satisfatory performance. In contrast, the SiM-based hash table achieves superior throughput even without leverage host DRAM for caching. This can be observed in Fig. 2 under various query distribution and page cache coverage[2]. Even at low coverage with a concentrated query distribution, the SiM-based table still achieves up to 2.8X speed-up.

Fig. 3 contrasts the queuing delays of SiM and conventional architectures using a concentrated query distribution (Zipf with $\alpha = 0.99$) and a 16 μs request expiry time for SiM's deadline scheduler[3]. Even with limited chips, SiM achieves lower queuing delays due to its query merging mechanism that batches multiple requests over the same page, allowing more on-chip bit matching operations over a single page read, rather than multiple separate reads.

References

[1] J. S. P. Liang Yu and L. Pilolli, "Peak power management in a memory device," Patent, Feburary, 2020, US Patent 11520497B2.

[2] J. Im, J. Bae, C. Chung, Arvind, and S. Lee, "PinK: High-speed in-storage key-value store with bounded tails," in *ATC*, 2020, pp. 173–187.

[3] C. Gao, X. Xin, Y. Lu, Y. Zhang, J. Yang, and J. Shu, "ParaBit: Processing parallel bitwise operations in nand flash memory based ssds," in *MICRO*, 2021, p. 59–70.

[4] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-Cosmos: In-flash bulk bitwise operations using inherent computation capability of nand flash memory," in *MICRO*, 2022, pp. 937–955.

[5] D. Gouk, M. Kwon, J. Zhang, S. Koh, W. Choi, N. S. Kim, M. Kandemir, and M. Jung, "Amber*: Enabling precise full-system simulation with detailed modeling of all SSD resources," in *MICRO*, 2018, pp. 469–481.

[1]Due to space constraints the parameter details are omitted here, but they can be provided upon further inquiry.

[2]Page cache coverage is defined as the size of the page cache relative to the working set size, which in this case is the total size of the hash table.

[3]The X-axis denotes the degree of chip-level hardware parallelism.