PiF: In-Flash Acceleration for Data-Intensive Applications

Myungjun Chun*
Seoul National University

Jaeyong Lee*
Seoul National University

Sanggu Lee
Seoul National University

Myungsuk Kim
Kyungpook National University

Jihong Kim
Seoul National University

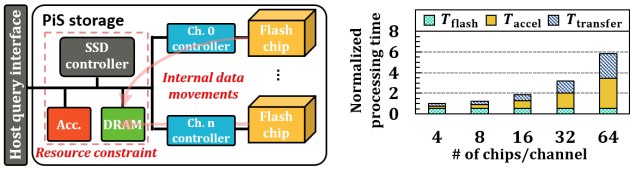
ABSTRACT

To minimize unnecessary data movements from storage to a host, processing-in-storage (PiS) techniques, which move a compute unit to storage, have been proposed. In this position paper, we propose an extreme version of PiS solutions, called a processing-in-flash (PiF) scheme, that moves computation inside flash chips where data are physically present. As a key building block of a PiF solution, we present a novel flash chip architecture, CoX. Using a prototype PiF SSD based on CoX chips, we demonstrate that PiF-based SSDs are promising in accelerating data-intensive applications.

1 INTRODUCTION

As the number of successful use cases based on big data analytics quickly increases, more complex data-intensive apps are widely developed. For such complex data-intensive apps, a very large amount of data is involved. Since real-world data-intensive apps (e.g., an intelligent query app [1, 2]) should handle a very large amount of data (e.g., a few terabytes [3]), a careful design of a memory/storage hierarchy is critical for these apps. For example, high-performance NVMe SSDs are commonly used for latency-sensitive apps as their storage devices because they provide the most effective solution to meet both the high-performance and large-capacity.

Using a high-performance storage system along with high-speed DRAMs, however, may not be an efficient solution for most performance-critical data-intensive apps. In many such apps, it is very likely that most data moved through a memory/storage hierarchy are not reused because of the little data locality in these apps. In order to overcome such limitations of conventional computer systems, processing-in-storage (PiS) techniques have been extensively investigated by many research groups [4–11]. In a PiS-based SSD, computation moves to a storage device so that the overhead of data movements to a host system can be minimized. Figure 1(a) illustrates an organizational overview of an example PiS SSD. Data stored in a flash chip of the PiS SSD first moves to an internal processing logic (e.g., F accelerator). Based on the processing results on the internal processing logic, qualified data for subsequent processing are transferred to the host, thus reducing a significant amount of data movements.



(a) An overall architecture of an example PiS SSD. **(b) Changes in processing time.**

Figure 1: A typical organization of a PiS SSD and its key limitations.

Although PiS techniques can achieve significant improvements over a conventional approach, their applicability is limited for data-intensive apps. First, a PiS-based SSD still needs a substantial amount of data movements because an accelerator in the PiS SSD is implemented at the SSD controller level. All data must be loaded from flash cells to SSD-internal memory before they are processed. In data-intensive apps, such internal data movements can incur a significant performance overhead with high power consumption.

Second, the acceleration capability of a PiS SSD is not scalable over the increasing storage capacity because an SSD controller is highly constrainted by its size, cost, and power consumption. Figure 1(b) illustrates the key limitations of the existing PiS approach for supporting data-intensive apps (See Section 3.3 for experimental settings.). The processing time of the data-intensive apps was broken into three components, T_{flash}, T_{transfer}, and T_{accel}, where T_{flash} is the total time spent reading data from flash chips, T_{transfer} is the total time spent for moving data to the SSD-internal memory

* The first two authors contributed equally to this research.

HotStorage'22, June 27-28, 2022, virtual conference

ACM ISBN 978-1-4503-9399-7/22/06…$15.00
https://doi.org/10.1145/3538643.3539742

for processing, and T_{accel} is the total time taken by an accelerator. As the storage capacity increases, the number of flash chips will increase, as shown in the x-axis of the graph. Thanks to increased chip-level parallelism, T_{flash} only slightly increases as the storage capacity increases. However, $T_{transfer}$ and T_{accel} may quickly increase as the number of chips increases. $T_{transfer}$ rapidly increases because the I/O bandwidth from flash chips to the SSD-internal DRAM is limited. T_{accel} can get longer when the acceleration capability cannot keep up with the increased data.

In order to overcome the limitations of a PiS technique, we propose a processing-in-flash (PiF) technique, an extreme version of the PiS approach, that offloads computation inside a flash chip (i.e., the lowest level of storage hierarchy where data are physically present). Furthermore, by offloading computation inside the flash chip, we can minimize $T_{transfer}$. By supporting an accelerator function at the chip level, the acceleration capability can be more scalable over the increasing number of flash chips. A PiF-based SSD, therefore, can be faster, more power-efficient, and more scalable over a PiS-based SSD.

Although the PiF approach seems to be promising, it is not straightforward to move computation into a flash chip. In this paper, we present the main technical hurdles in realizing a PiF-based SSD in practice with potential solutions. In order to demonstrate the effectiveness of the PiF approach, we present a prototype PiF-based SSD that employs an in-flash pattern matcher. Our evaluation results show that the PiF-based SSD is 5.7x faster and 6.6x more power-efficient on average over the PiS-based SSD.

2 COX DESIGN REQUIREMENTS

The key difference between a PiS SSD and a PiF SSD lies in how an acceleration function is placed within an SSD. Figure 2 shows an overall architecture of a PiF SSD. Unlike a PiS SSD as shown in Figure 1(a), the PiF SSD distributes its acceleration function among flash chips as well as a SSD controller. For example, the function F, which was accelerated in the PiS SSD, is divided into F_{front} and $F_{backend}$ in the PiF SSD and $F_{backend}$ is computed inside a flash chip.

2.1 Key Building Block: CoX Flash Chip

In order to build a PiF-based SSD, we propose a PiF-enabled flash chip that we call a Cell-over-X (CoX) flash chip. The

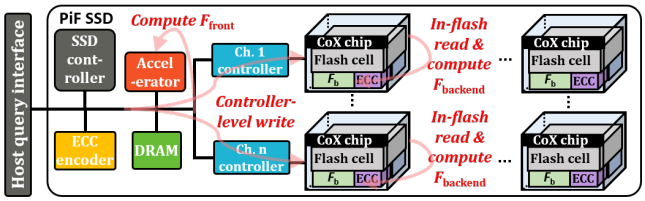
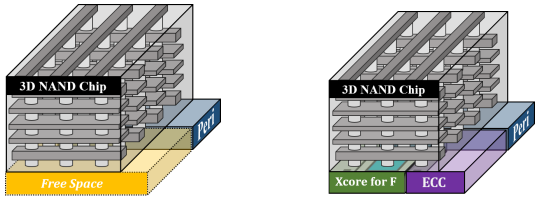


Figure 2: An overall architecture of an PiF SSD.



(a) A CoP flash chip. **(b) A CoX-F flash chip.**

Figure 3: An organizational comparison between an existing CoP flash chip and a CoX-F flash chip.

proposed CoX flash chip is based on the state-of-art Cell-over-Peri (CoP) technique that places peripheral circuits and core logic under a flash cell array [12]. Figure 3(a) and Figure 3(b) show an organizational overview of the existing CoP flash chip and a proposed CoX flash chip, respectively. The key motivation of designing the CoX flash chip is to place a chip-level accelerator module (called Xcore) and an ECC module on the bottom layer of the CoX chip by exploiting the unused free space of a CoP organization for in-flash processing. To emphasize an in-flash accelerator for a function F, we call such a CoX chip as CoX-F.

2.2 Area Constraint

One immediate concern in designing a CoX chip is whether sufficient free space exists on the bottom layer of the CoX chip so that Xcore and an ECC module, as well as peripheral circuit and core logic, can be implemented. In a CoX chip with the die area A, $(1 - P/A) \times 100\%$ is usable for Xcore and an ECC module where P is the area used for the peripheral circuit and core logic (e.g., row decoder/page buffer). In order to estimate the usable area for Xcore in flash chips, we investigated the per-module area ratio of modern flash chips. Figure 4 shows comparison results of six modern flash dies from two vendors. Flash cell arrays take about 82% of the die area, while the peripheral/page buffer/row decoder takes about 18% of the die area. Assuming that the size of flash cell arrays decides the die size of a CoX chip, about 78% of the CoX chip area can be used for Xcore and an ECC module. Since the average area of the six flash dies is 165 mm^2, the die size of a CoX chip is about 135 mm^2. Excluding 30 mm^2 for the peripheral/decoder/buffer, about 100 mm^2 can be used for Xcore and an ECC module. Unless the complexity of Xcore is very high, we find that there is sufficient free space for Xcore and an ECC module on the bottom layer of a CoX chip.

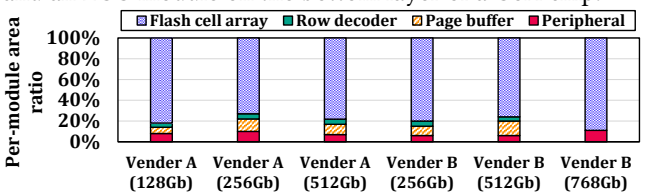


Figure 4: A breakdown of per-module area in various flash chips.

2.3 Power Constraint

High-performance SSDs are often limited by their maximum power budget. Since CoX chips may consume more power than conventional flash chips for in-flash acceleration, the acceleration capability of a PiF SSD may be limited by their power budget. Therefore, we require a simple but strong power constraint on a CoX chip, which can be summarized as $P_{\text{read}} + P_{\text{xcore}} + P_{\text{ecc}} \leq P_{\text{program}}$, where P_{read}, P_{program}, P_{ecc}, and P_{xcore} are the power consumption of a read, a program, an ECC module, and Xcore, respectively. When this constraint is satisfied, a PiF SSD can be interchangeably used where a normal SSD can be usable. In a typical 3D TLC NAND flash memory, about 20 mW can be allocated for operating Xcore and an in-flash ECC module [1].

2.4 Reliable In-Flash Read Requirement

In a PiF SSD, data stored in flash cells are read and processed by Xcore before they are restored to their error-free version by a controller-level ECC module. Therefore, one of the key requirements of a CoX chip is that in-flash reads should be reliable without depending on a controller-level ECC module. A straightforward solution may be to implement a controller-level ECC module inside a CoX chip. Even though it may be feasible to put a controller-level ECC module in a CoX chip, from the power consumption perspective, this is a bad design decision. As the number of CoX chips increases in a PiF SSD, the peak power consumption of the PiF SSD will increase, thus significantly limiting its acceleration capability.

A better solution is to employ two ECC modules, a weak ECC module ECC_w inside a CoX chip and a strong ECC module ECC_s in an SSD controller. For in-flash acceleration, ECC_w is used first. When ECC_w fails to correct bit errors of a flash page, ECC_s outside a CoX chip is used without an in-flash acceleration. Since it is not possible to encode data twice when it is written to flash cells, one by ECC_w and the other by ECC_s, both ECC_w and ECC_s should be able to decode the same encoded data. The efficiency of an in-flash acceleration largely depends on the error correction capability of ECC_w.

2.5 Xcore Offloading Requirements

As shown in Figure 2, a PiF SSD supports acceleration in two levels, F_{backend} by a CoX chip and F_{front} by a controller. To achieve high performance in a PiF SSD, therefore, it is important to intelligently decide F_{backend} so that the benefit of in-flash processing can be maximized. Ideal F_{backend} candidates can meet the following requirements. First, F_{backend} should significantly reduce the amount of data transfers from Xcore to a controller, thus avoiding a large amount of unnecessary data transfers. Second, F_{backend} should be suitable for data-parallel processing so that the parallelism of multiple CoX chips can be effectively exploited. When the dependency between CoX chips exists, a PiF SSD will be difficult to realize high performance because efficient communications between CoX chips are not easy. Third, the implementation of F_{backend} should be feasible under the power/area constraints of a CoX chip. For example, an in-flash ML model training function would be a potential candidate for Xcore. However, implementing such a function would be difficult because it requires a large SRAM buffer and lots of energy.

[1] For example, when P_{read} and P_{program} are 160 mW and 180 mW, respectively, 20 mW is the upper limit.

3 DESIGN OF PIF-PM

To demonstrate the feasibility of the PiF approach, we designed a prototype PiF SSD, called PiF-PM, that accelerates the performance of applications that require a data filtering process (e.g., log analysis [13, 14], SQL-based database [15, 16], web-scale data analysis [17], and graph mining [18, 19]). To build PiF-PM, we designed a CoX flash chip with a pattern matcher (PM), CoX-PM, that sends a flash page to an SSD controller only when the page contains a pattern that was set by a host. Pattern-matching-based applications satisfy most Xcore offloading requirements of Section 2.5, thus fitting very well with a PiF SSD. For example, when we support a low-level pattern-matching primitive with Xcore (i.e., F_{backend} = PM), a large amount of data transfers to a host can be reduced when matching pages are scarce. Since a pattern-matching operation can be processed independently on each CoX chip, high data-level parallelism among CoX chips can be maximally exploited. Furthermore, an implementation of a simple pattern matcher is less likely to incur a high power consumption or a large chip area in a CoX chip.

3.1 Overview of PiF-PM

PiF-PM follows a generic organization of a PiF SSD shown in Figure 2 with an in-CoX pattern matcher support. As shown in Figure 5 (a), a low-level flash controller directly interfaces with a CoX chip of PiF-PM with a few custom commands. Before CoX-PM gets started with its pattern-matching function, the flash controller initializes a hardware pattern matcher in CoX-PM by setting its target pattern with the *set_pattern* command (❶). In PiF-PM, a pattern is specified by a string of characters (e.g., "apple"). When a page contains at least one pattern string (e.g., "I would still plant my apple tree."), the page becomes a matched page.

To distinguish a conditional read with a matched pattern in CoX-PM from a normal read, we use the *read_when_matched* command. As shown in Figure 5 (b), when *read_when_matched* is issued with a physical page address to CoX-PM, a sensed page goes through an internal ECC decoder (❷). When the ECC decoder successfully corrects the flash page, it is sent to the pattern matcher (❸-T). When the page is matched, it is sent to the flash controller

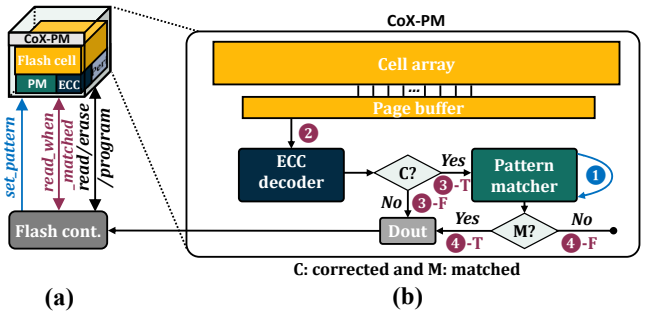


Figure 5: An operational overview of CoX-PM.

(❹-T). Since CoX-PM employs a two-level ECC scheme, the ECC decoder in CoX-PM can fail (❸-F). When the ECC decoder fails, the sensed page in the page buffer is sent to an SSD controller where a strong ECC module exists. The SSD controller would perform a pattern matching step as a PiS SSD would do.

3.2 Design of CoX-PM

On-chip ECC Decoder In order to meet the "encode-once but decode-twice" requirement of a two-level ECC scheme, we take advantage of soft and hard decoding algorithms of LDPC codes that are widely used for modern 3D NAND flash memory. As with most 3D flash-based SSDs, we use a soft decoding algorithm of LDPC codes as a strong ECC module (i.e., ECC_s). For a weak in-CoX ECC module (i.e., ECC_w), we use a hard decoding algorithm of LDPC codes. Specifically, our ECC_w is based on a bit flipping decoder [20] which has a lightweight hard decoding structure. On the other hand, as a controller-level ECC_s, we use the min-sum decoder [21] with a high error correction capability and complex soft decoding structure. Also, both decoders are constructed from a rate-0.89 (9216, 8192) QC-LDPC code so that the encoded data can be decoded by both ECC_s and ECC_w. Since the ECC_w module can operate with low power while the ECC_s module can provide the high error-correction capability, our hierarchical ECC structure efficiently supports in-flash read under the power constraint without compromising reliability.

Pattern Matcher Since many queries in data-intensive apps (e.g., Q2 in TPC-H [22]) commonly have multiple predicates, PM supports multiple patterns to be matched within a page. (In the current implementation, up to 8 patterns are supported. The maximum pattern size is 32 bytes.) We implemented PM by leveraging a concept of bit-split pattern matching automata [23] which supports multiple pattern matching at low cost. PM consists of four 1.2-KB SRAM tables to store the state of pattern-matching automata. PM fetches one-byte data per cycle from the ECC decoder for pattern matching. Since the SRAM tables of PM should be configured to support pattern-matching automata, the SRAM content is built in advance by an SSD controller[2].

Table 1: Overhead Analysis of CoX-PM.

Component	Area (mm^2)	Power (mW)
ECC Decoder	0.78	17.6
Pattern Matcher	0.07	1.06

Two custom commands for managing PM were implemented by using the vendor-specific command extension feature of the ONFI flash interface specification [24]. For the *read_when_matched* command, its three return values are specified in the reserved bits (the bits 3 and 4)[3] of the ONFI status register. Based on the return value of *read_when_matched*, an SSD controller decides if a pattern matching should be performed or not at the controller level.

Performance and Overhead To understand the performance and overhead of the proposed CoX-PM, we implemented ECC_w and PM using OpenRoad [25], an open-source ASIC design toolchain. To generate SRAM macros for PM, we used OpenRAM [26], an open memory compiler. In our current implementation, both the ECC decoding and PM take 40 μs on average for a 16-KB page. However, these extra latencies do not degrade the throughput of CoX-PM because both ECC_w and PM operate in a pipelined fashion[4].

We estimated the area/power consumption of the ECC decoder and PM using the analysis tool of OpenRoad. Table 1 summarizes the estimated results. The area estimate was 0.78 mm^2 and 0.07 mm^2 for the ECC decoder and PM, respectively. Since we estimate that about 100 mm^2 free space is available on the bottom layer of a CoX chip, the space overhead of CoX-PM seems to be negligible. For the peak power consumption, when CoX-PM is fully operating, CoX-PM consumes 18.66 mW more power over a normal flash chip. However, this additional power consumption does not violate the peak power limit of a normal flash chip. As explained in Section 2.3, we estimate that at least 20 mW can be safely allocated for CoX-PM.

3.3 Evaluations

In order to evaluate the effectiveness of `PiF-PM` over a PiS SSD, we built a prototype `PiF-PM` on an OpenSSD platform [28] where an SSD controller was implemented using Zynq-7000 SoC. For a fair comparison, a prototype PiS SSD, `PiS-PM`, was also built on the same OpenSSD platform.

[2]The content of SRAM tables, which depends on given target patterns, is generated as a bitstream by an FTL's special module. The bitstream is assigned to a unique ID, and the ID is used as an argument of *set_pattern*.

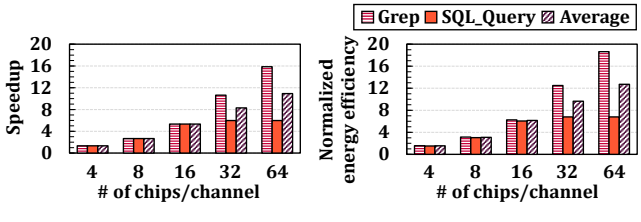
[3]When *read_when_matched* is executed, there are three cases, two normal executions (i.e., *matched_page_found* and *matched_page_not_found*) and one error execution (i.e., *decoding_failure*).

[4]There are two pipelines in CoX-PM. First, two pages are sensed and processed at the same time. While a page in a page buffer is being processed by ECC_w and PM, the following page is sensed to a cache buffer [27]. Second, ECC decoding and pattern matching are pipelined. After a 16-KB page is sensed to a page buffer (or a cache buffer), a 1-KB subpage is decoded by ECC_w while the previous 1-KB subpage is processed by PM. Since the page sensing latency is longer than 40 μs, both ECC decoding and pattern matching are fully overlapped within the latency of sensing the next page.

Since we cannot directly modify flash chips on the platform, we built an emulation environment for modeling flash chips using the onboard DRAM. Both CoX-PM chips and normal flash chips were emulated using this environment. Both SSDs have four channels. The number of chips per channel varies from 4 to 64. To fix the SSD capacity under all experiments, the number of flash block per plane were changed from 2048 to 128 as the number of chips increases from 4 to 64. The read latency, the program time, and the flash interface bandwidth per channel were set to 45 μs [29], 400 μs, and 1.2 Gb/s [29], respectively. We built an operation-centric SSD power model using a NAND system power calculator [30] and a Xilinx power estimator. The power/energy consumption of an SSD was estimated using the collected operation traces with the SSD power models. As data-intensive apps, we used two pattern-matching friendly apps, Grep and SQL_Query, from the previous study [5].

Figure 6 compares the performance and the energy efficiency of `PiF-PM` and `PiS-PM` while varying the number of chips per channel. All values in Figure 6 are normalized to `PiS-PM` for a given number of chips. To understand the maximum benefit of `PiF-PM` over `PiS-PM`, we assumed that all the flash cells are in the early stages of their lifetimes so that ECC_w can fully correct potential raw bit errors in flash cells. As shown in Figure 6(a) and 6(b), `PiF-PM` outperforms `PiS-PM` by up to 15.9 times and 18.6 times in the processing time and the performance per watt, respectively. `PiF-PM` achieves better efficiencies in Grep over SQL_Query because only 6.3% of pages were matched pages in Grep over 16.7% pages in SQL_Query. Figure 6 also shows that `PiF-PM` achieves higher gains over `PiS-PM` as the number of CoX chips increases. This is because the channel bandwidth is not scalable in `PiS-PM`, but the increased parallelism can be fully exploited in `PiF-PM`.

The comparison results shown in Figure 6 are based on the ideal flash reliability condition where all the raw error bits of flash cells can be corrected by the in-flash ECC_w. However, in real operating environments, this condition is not always satisfied. When ECC_w fails to correct some error bits, the performance of `PiF-PM` may get deteriorated. For example, Table 2 shows how the average performance of `PiF-PM` under 32 CoX chips per channel changes as the decoding



(a) Total processing time. (b) Performance per watt.

Figure 6: Comparisons of `PiS-PM` and `PiF-PM`.

Table 2: Impact of the ECC_w failure on performance.

Failure rate	0	0.05	0.1	0.15
Normalized performance	1	0.83	0.63	0.51

failure rate of ECC_w increases. For example, when only 85% of decoded pages are corrected by ECC_w, the performance of `PiF-PM` is reduced by 49.9%, which is still significantly higher than `PiS-PM`.

In order to better understand the performance implication of ECC_w on the performance of `PiF-PM`, we measured the number of bit errors in modern TLC flash memory under various conditions. Figure 7 compares the average number of raw bit errors per 1 KB for different combinations of a P/E cycle count and a retention time requirement. The current ECC_w, whose maximum correction capacity is 72 bits per 1 KB, successfully decodes raw flash pages except for the worst-case combination (i.e., 2K P/E cycles with the 12-month retention time requirement). Therefore, to maximize the efficiency of `PiF-PM`, it is important to manage data reliability so that the number of raw bit errors of a page does not exceed the maximum correction capacity of ECC_w.

4 RELATED WORK

Few studies exist on in-flash processing. The closest work to the PiF approach is ParaBit [31] that exploits in-flash bit-wise operators (that are commonly found in 2D/3D flash chips). However, there are significant differences between ParaBit and the PiF approach. ParaBit is limited to approximate computing where inaccurate results are acceptable because it does not guarantee error-free reads inside a flash chip. Furthermore, ParaBit does not take advantage of a modern CoP chip in supporting custom in-flash acceleration.

5 CONCLUSION

As an ultimate PiS solution, we have presented a PiF SSD that moves computation to inside flash chips so that data transfers can be minimized. As a key building block of a PiF SSD, we proposed a CoX flash chip that includes an accelerator and an ECC decoder. As a prototype PiF SSD, we designed `PiF-PM` where pattern matching operations are performed inside CoX chips. Our evaluation results demonstrate that a PiF SSD can be a promising solution for accelerating data-intensive applications in a power-efficient fashion.

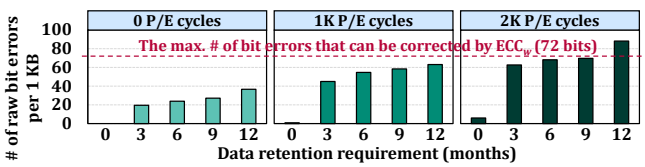


Figure 7: Changes in the number of raw bit errors.

ACKNOWLEDGMENTS

This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics, Republic of Korea, under Project Number SRFC-IT2002-06. The ICT at Seoul National University provided research facilities for this study. Myungsuk Kim was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2021R1G1A1094835). (Corresponding author: Jihong Kim.)

REFERENCES

[1] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia de Gonzalo, Youjie Li, et al. DeepStore: In-Storage Acceleration for Intelligent Queries. In *the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[2] Fedor Borisyuk, Albert Gordo, and Viswanath Sivakumar. Rosetta: Large scale system for text detection and recognition in images. In *the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2018.

[3] Dayong Wang, Charles Otto, and Anil K Jain. Face search at scale: 80 million gallery. *arXiv preprint arXiv:1507.07242*, 2015.

[4] Louis Woods, Zsolt István, and Gustavo Alonso. Ibex: An Intelligent Storage Engine with Support for Advanced SQL Offloading. *VLDB Endowment*, 7(11), 2014.

[5] Zhenyuan Ruan, Tong He, and Jason Cong. INSIDER: Designing in-Storage Computing System for Emerging High-Performance Drive. In *the USENIX Annual Technical Conference (ATC)*, 2019.

[6] Yanqin Jin, Hung-Wei Tseng, Yannis Papakonstantinou, and Steven Swanson. KAML: A Flexible, High-Performance Key-Value SSD. In *the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[7] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. Query Processing on Smart SSDs: Opportunities and Challenges. In *the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2013.

[8] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD. *IEEE Computer Architecture Letters*, 19(2), 2020.

[9] Gunjae Koo, Kiran Kumar Matam, Te I., H.V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. Summarizer: Trading Communication with Computing Near Storage. In *the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[10] Sang-Woo Jun, Andy Wright, Sizhuo Zhang, Shuotao Xu, and Arvind. GraFboost: Using Accelerated Flash Storage for External Graph Analytics. In *the International Symposium on Computer Architecture (ISCA)*, 2018.

[11] Mahdi Torabzadehkashi, Siavash Rezaei, Ali Heydarigorji, Hosein Bobarshad, Vladimir Alves, and Nader Bagherzadeh. GraFboost: Using Accelerated Flash Storage for External Graph Analytics. In *the Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2019.

[12] Jae-Woo Park, Doogon Kim, Sunghwa Ok, Jaebeom Park, Taeheui Kwon, Hyunsoo Lee, et al. 30.1 A 176-Stacked 512Gb 3b/Cell 3D-NAND Flash with 10.8Gb/mm2 Density with a Peripheral Circuit Under Cell Array Architecture. In *the IEEE International Solid-State Circuits Conference (ISSCC)*, 2021.

[13] Candace Suh-Lee, Ju-Yeon Jo, and Yoohwan Kim. Text mining for security threat detection discovering hidden information in unstructured log messages. In *the IEEE Conference on Communications and Network Security (CNS)*, 2016.

[14] Seongyoung Kang, Jiyoung An, Jinpyo Kim, and Sang-Woo Jun. MithriLog: Near-Storage Accelerator for High-Performance Log Analytics. In *the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[15] Boncheol Gu, Andre S Yoon, Duck-Ho Bae, Insoon Jo, et al. Biscuit: A framework for near-data processing of big data workloads. *ACM SIGARCH Computer Architecture News*, 44(3), 2016.

[16] Yangwook Kang, Yang-suk Kee, Ethan L Miller, and Chanik Park. Enabling cost-effective data processing with smart SSD. In *the IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2013.

[17] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, et al. SDF: Software-defined flash for web-scale internet storage systems. In *the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[18] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, et al. Arabesque: a system for distributed graph mining. In *the Symposium on Operating Systems Principles (SOSP)*, 2015.

[19] Anand Padmanabha Iyer, Zaoxing Liu, Xin Jin, Shivaram Venkataraman, et al. ASAP: Fast, approximate graph pattern mining at scale. In *the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[20] Khoa Le, Fakhreddine Ghaffari, David Declercq, and Bane Vasić. Efficient hardware implementation of probabilistic gradient descent bit-flipping. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(4):906–917, 2017.

[21] Thien T. Nguyen-Ly, Tushar Gupta, Manuel Pezzin, Valentin Savin, et al. Flexible, cost-efficient, high-throughput architecture for layered ldpc decoders with fully-parallel processing units. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 230–237, 2016.

[22] Peter Boncz, Thomas Neumann, and Orri Erling. TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark. In *the Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, 2013.

[23] Lin Tan and T. Sherwood. A high throughput string matching architecture for intrusion detection and prevention. In *the International Symposium on Computer Architecture (ISCA)*, 2005.

[24] ONFI. Open NAND Flash Interface Specification 4.1. http://www.onfi.org/, 2017.

[25] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, et al. INVITED: Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *the ACM/IEEE Design Automation Conference (DAC)*, 2019.

[26] Matthew R. Guthaus, James E. Stine, Samira Ataei, Brian Chen, et al. OpenRAM: An open-source memory compiler. In *the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[27] Ki Chang Chun, Hee Joung Park, Tae Seung Shin, and Sung Lae Oh. Memory device including page buffer and method of arranging page buffer having cache latches, 2018. US Patent 9,965,388.

[28] Jaewook Kwak, Sangjin Lee, Kibin Park, Jinwoo Jeong, and Yong Ho Song. Cosmos+ OpenSSD: Rapid Prototype for Flash Storage Systems. *ACM Transactions on Storage*, 16(3), 2020.

[29] Dongku Kang, Minsu Kim, Su Chang Jeon, Wontaeck Jung, et al. 13.4 A 512Gb 3-bit/Cell 3D 6th-Generation V-NAND Flash Memory with 82MB/s Write Throughput and 1.2Gb/s Interface. In *the IEEE International Solid-State Circuits Conference (ISSCC)*, 2019.

[30] Micron NAND System Power Calculator. https://www.micron.com/support/tools-and-utilities/nand-system-power-calculator.

[31] Congming Gao, Xin Xin, Youyou Lu, Youtao Zhang, et al. ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory based SSDs. In *the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.