TH-iSSD: Design and Implementation of a Generic and Reconfigurable Near-Data Processing Framework

JIWU SHU, KEDONG FANG, YOUMIN CHEN, and SHUO WANG, Tsinghua University

We present the design and implementation of TH-iSSD, a near-data processing framework to address the data movement problem. TH-iSSD does not pose any restriction to the hardware selection and is highly reconfigurable—its core components, such as the on-device compute unit (e.g., FPGA, embedded CPUs) and data collectors (e.g., camera, sensors), can be easily replaced to adapt to different use cases. TH-iSSD achieves this goal by incorporating highly flexible computation and data paths. In the data path, TH-iSSD adopts an efficient device-level data switch that exchanges data with both host CPUs and peripheral sensors; it also enables direct accesses between the sensing, computation, and storage hardware components, which completely eliminates the redundant data movement overhead, and thus delivers both high performance and energy efficiency. In the computation path, TH-iSSD provides an abstraction of *filestream* for developers, which abstracts a collection of data along with the related computation task as a file. Since existing applications are familiar with POSIX-like interfaces, they can be ported on top of our platform with minimal code modification. Moreover, TH-iSSD also introduces mechanisms including *pipelined near-data processing* and *priority-aware I/O scheduling* to make TH-iSSD perform more effectively. We deploy TH-iSSD to accelerate two types of applications: the content-based information retrieval system and the edge zero-streaming system. Our experimental results show that TH-iSSD achieves up to 1.6× higher throughput and 36% lower latency than *compute-centric* designs.

CCS Concepts: • **Information systems** → *Storage architectures;*

Additional Key Words and Phrases: Near data processing, information retrieval, deep learning, storage architecture

ACM Reference format:
Jiwu Shu, Kedong Fang, Youmin Chen, and Shuo Wang. 2023. TH-iSSD: Design and Implementation of a Generic and Reconfigurable Near-Data Processing Framework. *ACM Trans. Embedd. Comput. Syst.* 22, 6, Article 96 (November 2023), 23 pages.
https://doi.org/10.1145/3563456

1 INTRODUCTION

Recent years have witnessed a surge of deep learning applications, ranging from images/videos and text/NLP to billion-scale recommendation systems [66]. As the model and dataset sizes of deep

This work was supported by the National Natural Science Foundation of China (grant no. 61832011) and the Open Research Program of Zhejiang Lab (no. 2020KC0AB03).
Authors' address: J. Shu, K. Fang, Y. Chen, and S. Wang, Computer Science Department, Storage System Research Group, Tsinghua University, 30 Shuangqing Road, Haidian Qu, Beijing, Beijing, China, 100084; emails: shujw@tsinghua.edu.cn, {fkd19, chenym16}@mails.tsinghua.edu.cn, shvowang@tsinghua.edu.cn.

1539-9087/2023/11-ART96 $15.00
https://doi.org/10.1145/3563456

learning scale up, the memory requirement for training neural networks increases explosively. Researchers are struggling with the limited memory capacity to store the huge amounts of weights and activations for large networks with tens of billions of parameters. For example, with a large network such as ResNet-1001 [21], the local batch size for training cannot be larger than two ImageNet samples when we use the latest generation of the NVIDIA GPU (V100, 32 GiB of RAM); furthermore, the Megatron-LM model [57] has 8.3 billion parameters, which needs more than 250 GiB of DRAM to fit. To sustain the scaling the model and inference/training datasets, current DNN systems are incorporating more and more DRAMs into the system. This is quite costly with respect to both power and cost.

To run large models beyond the memory capacity limits, existing approaches either push part of the data back to fast external devices (e.g., **Solid-State Drives (SSDs)**) or embrace distributed solutions, which incur *expensive data movements*. In the former case, training data is stored at a low level (e.g., SSDs) and must be moved upward all the way to the volatile DRAM and CPU registers. As prior work [36] has pointed out, using external storage devices (e.g., high-end SSDs) is confronted with two critical sources of overhead. First, the current I/O software stack burdens the system significantly since the performance bottleneck has migrated from hardware to software when traditional **Hard Disk Drives (HDDs)** are replaced by high-speed SSDs. Second, massive data movement incurs high latency and wastes hardware bandwidth unnecessarily. In the latter case, distributed training/prediction can be used if multiple compute devices (e.g., GPUs) are available. On top of such a setting, the model and the training data are distributed across multiple compute nodes to fully exploit the model/data parallelism. However, this scheme does not reduce memory consumption anyway; instead, the training/prediction process requires frequent data exchange between different nodes. As a result, keeping data in DRAM yields a 10,000× reduction in latency, but distributing the memory through commodity network eliminates 1,000× of such benefit [47].

All preceding approaches adopt the *compute-centric* architecture, where data has to be moved from the drive or remote memory to the host, thereby affecting the overall computational efficiency and power consumption. A promising alternative to mitigate the preceding bottleneck is to move computation from the host to the drive, which is known as **Near-Data Processing (NDP)** [8, 9], or **In-Storage Processing (ISP)** when the storage media is non-volatile memory. ISP equips the storage with intelligence (i.e., processors) and lets it process the data stored therein firsthand, which exhibits low-power consumption, low latency, and high throughput. However, despite the potential of ISP, it has not been well deployed in commercial systems thus far due to two main reasons: one on the software side and the other on the hardware side. On the software side, existing systems typically offload only simple functions or operators, like scanning and sorting, to intelligent disks, leaving non-negligible processing overhead to host CPUs [4, 17, 25, 55, 70]; moreover, some of them adopt a highly customized solution with self-defined APIs and data layout, which are not compatible with existing operating systems [10] and applications, preventing them from being widely deployed in different types of applications without code modification [36]. On the hardware side, prior ISP designs are highly coupled with the hardware characteristics of intelligent drives, making them only suitable for deployment in a certain scenario. For example, FPGA-based SmartSSDs are hard to be integrated into an IoT device at an edge environment due to the space and power constraints.

Ideally, we want a generic framework for NDP that is able to (1) accelerate different types of deep learning applications with a unified abstraction and (2) support different hardware setups (e.g., FPGA, ARM processor, or SoCs) that can be deployed in a variety of scenarios. To achieve these goals, we present TH-iSSD, a full-stack redesigned ISP framework with the primary focus on ensuring a high level of programmability, efficiency, and flexibility. We make the following technical contributions:

- *Sensing-storage-computation integrated hardware architecture.* Existing solutions mainly focus on processing data efficiently by moving computation close to the data, where they assume that the data to be processed is already in the drive. However, in real-world deployment, the data should always be first collected from the data collectors (e.g., sensors, camera, applications) before they can be processed/analyzed, which already introduces one time of data transmission over the network. TH-iSSD takes a different approach by integrating the control logic of data collectors, storage drives, and computation accelerators in one hardware controller, which minimizes the cost of data movements. TH-iSSD is highly reconfigurable: the sensing element and accelerators of TH-iSSD can be easily replaced to fulfill the requirements of power supply and application logic given a certain deployment case (see Section 4 for details).
- *Storage-centric data management for ISP.* High-speed flash drives outperform traditional hard disks in terms of both bandwidth and latency, and offer high internal parallelism by adopting a multi-channel architecture. But unfortunately, flash exhibits several limitations, such as imbalanced read-write performance, limited write endurance, and erase-before-write requirements, which often leave the performance of flash drives underutilized. To address these issues, TH-iSSD introduces both placement-aware ISP and priority-aware I/O scheduling mechanisms to fully utilize the internal parallelism of the drive.
- *Flexible programming model via file system extension.* Existing ISP programs access data by either managing the drive as a raw block device without a file system [29, 36] or requiring careful coordination with the host [13], which requires considerable host code modification to leverage the ISP capability. TH-iSSD provides *filestream*, an easy-to-use and effective abstraction for users to program the computation logic without considering the data placement.

We deploy two representative I/O-intensive applications—that is, the **Content-Based Information Retrieval (CBIR)** system and the edge streaming system—on top of our proposed TH-iSSD framework. Experimental results show that TH-iSSD achieves a performance improvement of up to 1.6× and reduces latency by 36% compared to traditional *compute-centric* designs.

2 BACKGROUND AND MOTIVATION

2.1 Flash Memory Systems and Data Movement Overhead

In typical data processing tasks, CPUs need to interact with the storage system frequently to fetch data and write back the results. Due to the growing disparity of speed between CPU and storage outside the CPU chip, the storage system plays a crucial role in the performance of applications. Recently, flash memory based SSDs are considered as a major alternative over hard disks due to their better power efficiency and higher data accessing rate. However, even fast SSDs still incur expensive data movement overhead. In this section, we describe the flash memory architecture, analyze the data movement overhead, and illustrate the key reasons for such inefficiencies.

Flash Memory Architecture. As shown in Figure 1, modern high-speed SSDs are interconnected to hosts via **PCI Express (PCIe)** slots or M.2 slots through the NVMe protocol, and NAND flash memory chips are the primary non-volatile storage media. The host CPU reads or writes data from or to SSDs by issuing PCIe commands via memory-mapped I/O to the drive, then the drive fetches the I/O command and performs direct memory access operations to transfer the actual data. Current-generation NVMe SSDs allow the host CPU to issue dozens of requests in batch, which are queued within the SSD controller's command queues, then the controller may schedule these requests so as to maximize the internal parallelism. Each drive may contain dozens of flash chips, which are organized into multiple channels. A flash memory chip is a package that further

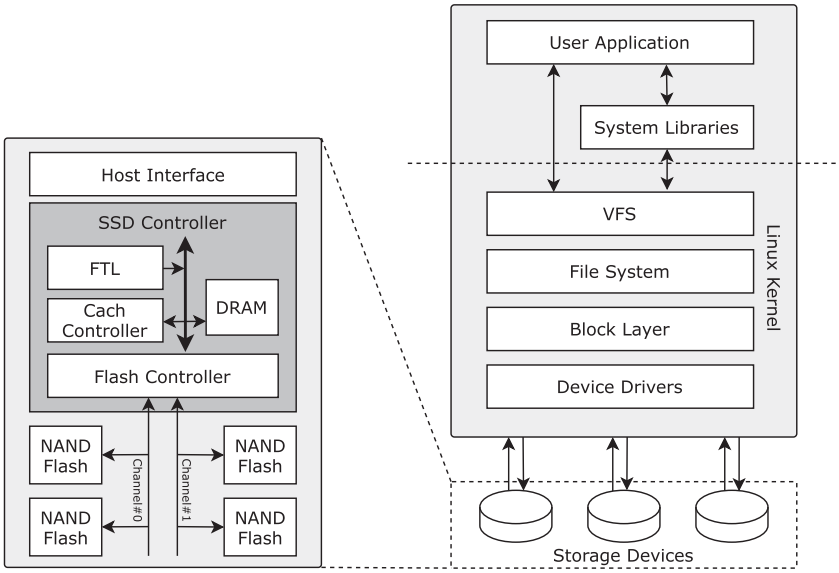


Fig. 1. The flash memory architecture.

contains multiple dies. A die is the smallest unit that can execute I/O requests independently. SSDs support both chip-level and channel-level parallelism, where different channels can be accessed in parallel and the chips that share a single channel can be multiplexed on that channel.

Flash memory does not allow in-place updates. Instead, every page becomes immutable once being written and must be erased before it can be written again. The drive erases pages at a block granularity, which contains between 64 and 512 pages. Each block can only be erased a limited number of times during its lifetime, and erase operations have much higher latency than that of page reads and writes. For these reasons, the SSD relies on the **Flash Translation Layer (FTL)** in the controller to map physical pages to a logical block address, where applications access the drive with the logical block address and the drive then applies a wear-leveling algorithm to spread updates to the whole physical space and garbage collects obsolete pages.

Data Movement Overhead. Typical high-end data center SSDs (16-channel, single-bank SSDs, which are very common now) can easily reach 6.4-GB/s bandwidth [27, 46]. Recent advances of 3D NAND flash technology achieve even higher bandwidth. However, this huge bandwidth decreases dramatically due to the complicated storage software stack and the bandwidth limitation of PCIe lanes. For example, PCIe Gen3 is adopted as the standard interconnection approach in the state-of-the-art platform, which is at 1-GB/s bidirectional transmission speed per lane. Four-lane link is most commonly used in commercial produces (e.g., drives from Intel [1] and Samsung [2]), implying the 4-GB/s external bandwidth, which can be easily surpassed by the internal bandwidth of the drive.

On the software side, the I/O stack is a long-lived component in the OS, which is designed for slow storage devices such as HDDs. Its complicated and layered design is shown to cause significant performance overhead for fast NVMe SSDs. First, the interrupt-based processing mode incurs extremely high latency (60.8 μs [58], which is even higher than the read/write latency of SSDs). Second, the relevant data must travel through the PCIe bus, multiple buffers in main memory, and multi-level CPU caches before they can be finally accessed by host CPUs, which causes redundant memory copies—this not only impacts latency but also wastes hardware bandwidth. Another

factor of the data movement overhead is the energy consumption. For scientific applications executed on high-performance servers, 28% to 40% of total energy cost is spent in moving data [26]. Furthermore, moving data off-chip to DRAM or external devices consumes orders of magnitude higher energy than that of on-chip data movement.

2.2 Near-Data Processing

To address the data movement overhead, recent studies have proposed the concept of NDP, which covers both **Processing in Memory (PIM)** and ISP. In this article, we focus primarily on ISP that can accommodate a large amount of data.

The design philosophy of "moving compute closer to data" is tenable for the following two reasons: first, as mentioned earlier, modern SSDs are usually manufactured with higher internal bandwidth than the external interconnection bandwidth; second, modern SSDs usually embed multi-core processors (e.g., ARM cores) as SSD controllers to manage the channel parallelism and internal bandwidth. These processors handle I/O request scheduling, data mapping through the FTL, wear leveling, and garbage collection. However, the computing capabilities in the drive are generally ignored, and the drive is treated as a storage-only device, leaving the device underutilized.

To fully exploit the SSD potential, ISP was recently proposed to fully utilize the internal parallelism and the processing capability of embedded processors [17, 55, 70]. The main idea is to treat the SSD as an intelligent device and execute application code directly inside the drive. In a data query system, for instance, the host can send the query to the drive directly instead of executing the query on the host side. Upon receiving a query, the embedded processors read relevant data from flash chips to the device buffer and execute the query. Then, only the results are returned back the host, which is expected to be much smaller than the raw data. As we can see, ISP significantly reduces the amount of data that should be exchanged between the host and drive, and thereby reduces latency and power consumption. Moreover, energy can be further reduced since ARM processors consume less power (3–4×) than host CPUs [67].

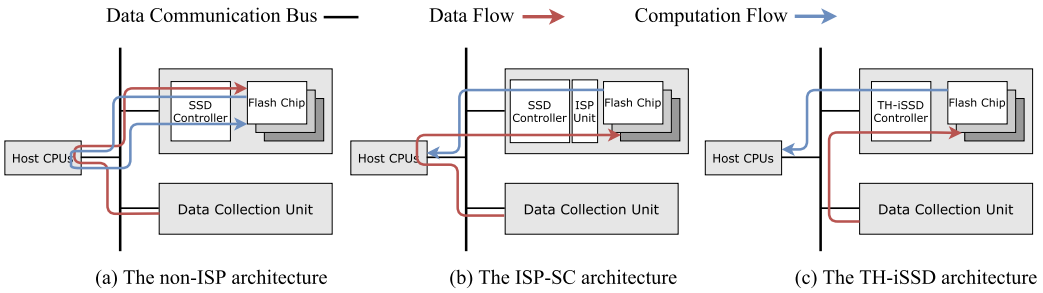
As the ARM processors within the drive are less powerful, many existing works can only offload I/O-intensive but computationally simple tasks to the drive [28]. However, many modern applications are both I/O and computation intensive (e.g., information retrieval, deep learning applications). To this end, recent work incorporates more powerful compute units in the drive directly. For example, the Samsung SmartSSD computational storage drive is equipped with an onboard FPGA, which enables offloading more complicated computation tasks within the drive [34].

2.3 Challenges

The idea of ISP has been extensively studied in the past 10 years; however, most of the previous ISP designs are still hard to deploy in real-world applications for the following reasons:

(1) *Performance*: Traditional HDDs or SSDs are assumed to serve only host I/O requests; therefore, the I/O software stack as well as the storage controller are highly optimized to better serve host I/Os. In real deployment scenarios of ISP, however, normal host I/Os typically *coexist* with in-storage computing I/Os. Considering the unique hardware features of flash memory (e.g., limited erase cycles, background GC tasks), we argue that ensuring the efficiency of ISP requires careful scheduling among parallel and heterogeneous I/Os to maximize the internal bandwidth of SSDs and improve user-facing service experiences (i.e., avoiding latency spikes).

(2) *Compatibility*: As an alternative hardware component, ISP-enabled SSDs are expected to be deployed in existing applications with minimal code modifications. This requires us to



(a) The non-ISP architecture (b) The ISP-SC architecture (c) The TH-iSSD architecture

Fig. 2. The difference between TH-iSSD and existing approaches.

provide a unified abstraction of the functionalities inside SSDs so that these functions can be easily invoked by applications as if they are still using a legacy storage device.

(3) *Flexibility*: Instead of providing dedicated ISP software design on a customized hardware platform, we want a sensing-storage-computation integrated ISP framework that can be highly adaptable to various application scenarios, where each individual component can be easily replaced. For example, in one of our weather applications, the devices are deployed at the edge without sustainable power supply, which requires that the device itself should be power efficient and the embedded processor runs a highly optimized weather prediction model.

3 DESIGN

This section gives an overview of TH-iSSD, its unique technical contribution compared to existing approaches, the storage management that maximize the internal parallelism of SSDs, and the interface that interacts with upper-layer applications.

3.1 TH-iSSD Overview

Figure 2 summarizes the comparison of our proposed TH-iSSD and existing approaches, including the traditional compute-centric architecture (denoted as *Non-ISP*), where the **Data Collection Unit (DCU)** (e.g., sensors, camera), the storage node, and the compute node are externally interconnected, and the recently proposed ISP architecture that only co-locates the storage and computation counterparts within the device (denoted as *ISP-SC*).

In *Non-ISP* (Figure 2(a)), the newly collected data has to be first stored to the storage device by host CPUs, then fetched to the main memory for processing before being stored to the device again. We can observe that the whole process involves frequent data movements and communication between the three components, which has great impact on the overall performance. *ISP-SC* shortens the data path by allowing the device to preprocess data directly. However, host CPUs still need to be involved when the DCU transmits data to the storage device (see the red line in Figure 2(b)).

Different from preceding approaches, TH-iSSD allows data to move between the DCU, computation unit, and the storage media directly within the device without an external interconnection (as shown in Figure 2(c)). The key enabler for TH-iSSD to achieve this is a carefully designed hardware architecture in the device, which interacts with host CPU, DCU, on-device processing unit, and the flash chips simultaneously, so all components can communicate directly with minimal overhead of data movement.

The TH-iSSD Architecture. Figure 3 depicts the overall architecture of our introduced TH-iSSD, which consists of three key components. The first component is a *unified interface* that interacts with both host CPUs and peripheral DCUs. This generic interface relies on the data switch to serialize input data from heterogeneous devices to form a device-friendly layout. The second component

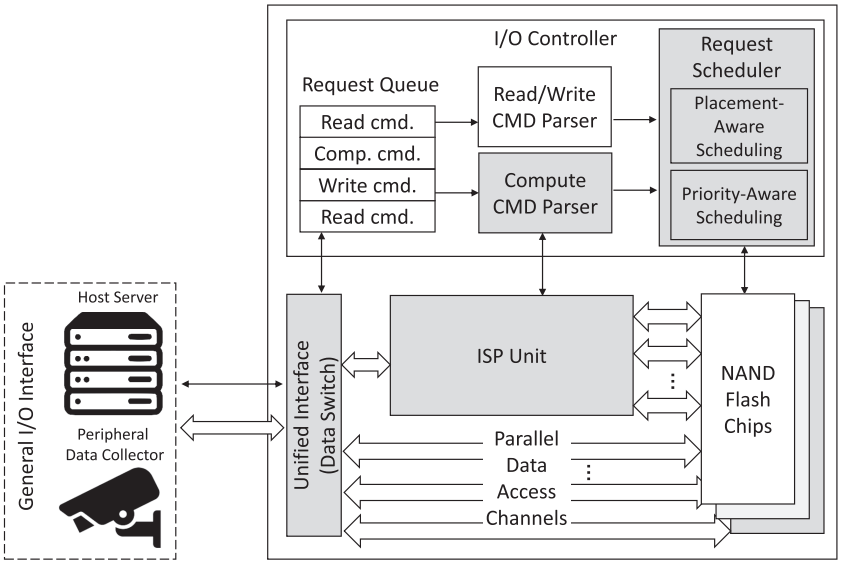


Fig. 3. The TH-iSSD architecture.

is an *ISP unit* that executes the offloaded operations within the device. It is worth mentioning that executing host-side code in the device is not as simple as we thought—host CPUs run the code by accessing data from main memory with byte-addressable load/store instructions, whereas an in-device ISP unit interacts with flash chips with the page granularity. To address this, TH-iSSD refurbishes the logic of host-side tasks or operations into their flash-aware counterparts, thus avoiding the flash storage being a possible bottleneck. The third component is an *I/O controller* that schedules host-side normal I/Os and ISP I/Os effectively by introducing priority-aware I/O scheduling policies.

The Data Consistency. Since the data is originally collected by the data collector and then transferred to the back-end SSD storage, the data would pass through data collector, host memory, and the SSD storage. It is critical to guarantee the data consistency among the three parties. To guarantee the consistency, TH-iSSD treats the host memory as a temporary buffer that would never be modified, and thus the data consistency could be ensured by only taking care of the consistency between the data collector and SSD storage. Since only the data stored in the SSD is non-volatile, TH-iSSD takes advantage of the log-structured file system to guarantee the data consistency by a data logging based mechanism.

3.2 ISP-Aware Data Path Management

TH-iSSD faces the following challenges when communicating with host-side applications and peripheral DCUs. First, TH-iSSD requires a mechanism to allow in-device accelerators or DCUs to access data in flash chips directly without the involvement of host CPUs; considering that the flash storage is managed by host CPUs via a file system, in-device *direct access* should never corrupt the data layout that the file system organizes. Second, TH-iSSD needs a general approach to serialize the received data from heterogeneous DCUs (which have varying interconnection protocols) before storing them to the device. Third, existing host-side applications are executed by accessing the main memory data; however, this is not the case in the device since ISP tasks need to interact with flash chips directly, which might become the performance bottleneck.

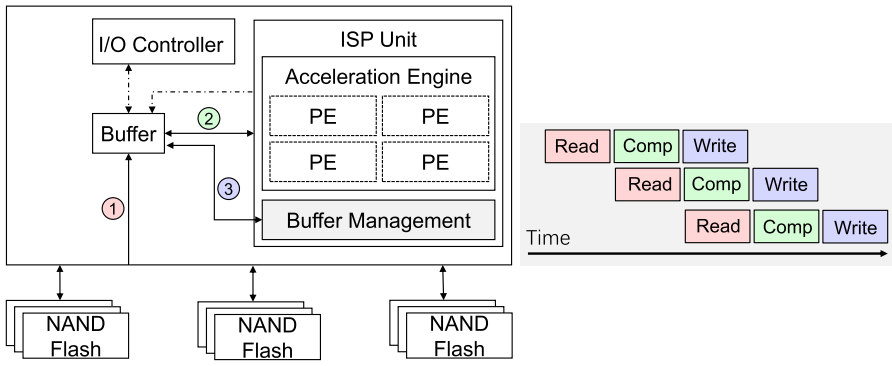


Fig. 4. Pipelined ISP.

In-Device Direct Access. To address the first challenge, TH-iSSD achieves in-device direct access leveraging the mapping information of the host-side file system. This is possible since existing POSIX file systems provide interfaces such as `fibmap` or `fiemap` and applications can use them to retrieve the locations of real file extents. For instance, when an application needs to offload a processing task to in-storage processors, then the application provides the device with a stream of physical addresses so that the embedded processes can read or write data directly without any software overhead. As mentioned earlier, offloaded tasks often involve accessing a large amount of data, so the overhead of issuing `fibmap/fiemap` calls is almost negligible. For DCUs that continue writing collected data, TH-iSSD requires the application to periodically preallocate enough flash space via the `fallocate` call. In this way, the newly collected data by DCUs can be stored in the reserved space directly without interacting with the host.

Data Switch. To achieve in-device direct access of DCUs, TH-iSSD further requires a data switch to serialize the inbound streaming data so that they can be stored into the flash chips or preprocessed by in-storage accelerators. In the former case, the data switch batches inbound data in the device buffer until they reach a flash page-aligned size, then these new pages are forwarded to the flash chip according to the mapping information provided by host applications. When the data needs to be preprocessed, the data switch forwards raw data directly to the ISP unit—we can observe that host CPUs are completely eliminated in this data path, which brings the following two obvious advantages. First, the newly collected data steam does not need to be serialized anymore since it is already in a format that the accelerator understands; otherwise, the data should be first organized into a file and stored in the file system, and is then deserialized into a memory format before being processed. Second, by eliminating the involvements of host CPUs, TH-iSSD avoids most redundant data movements and software overheads.

Pipelined ISP. To execute offloaded tasks in the device efficiently, we incorporate a pipelined approach to exchange data among flash chips, the in-device buffer, and the ISP unit. As shown in Figure 4, the computation task is split into small pieces, which are executed on multiple **Processing Elements (PEs)**. Each execution is further divided into multiple stages, namely the *read*, *compute*, and *write* stages. In the *read* stage (i.e., ①), the ISP unit sends a command to the I/O controller to load the relevant data from flash chips to the device buffer; in the *compute* stage (i.e., ②), the ISP unit exchanges data between the device buffer and its on-chip cache with a buffer management component, and executes the offloaded computation tasks; once the compute stage finishes, the ISP unit writes back the output data to the device buffer, which is finally written back to flash chips or sent to host CPUs (i.e., the *write* stage of ③). These stages are executed in a pipeline to maximize the potential parallelism of the device.

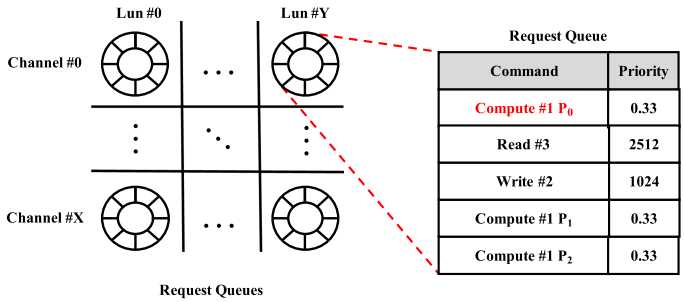


Fig. 5. The storage scheduling mechanism of TH-iSSD.

3.3 Priority-Aware I/O Scheduling

TH-iSSD receives ISP requests and normal host I/Os simultaneously, which requires a careful scheduling between these requests. As shown in Figure 5, the I/O controller of TH-iSSD manages a FIFO (first-in, first-out) request queue for each flash die and reorders inbound requests in a fine-grained approach. ISP I/O requests usually involve reading a large amount of data from the storage device, which may occupy the storage resources (i.e., hardware bandwidth) for a long time, causing serious latency spikes. As a result, traditional non-preemptible scheduling mechanisms are hard to balance well between the bandwidth and latency. To minimize the interference of ISP I/Os on normal host I/Os, TH-iSSD adopts a priority-based request scheduler to dynamically schedule the heterogeneous requests, which achieves the best of two worlds—the high bandwidth of ISP I/Os and low latency of normal host I/Os.

To help with understanding the scheduling policy, we introduce a comprehensive evaluation model to illustrate the relationship between application's priority and predicted execution time, which is defined in Equation (1). In the equation, A represents the priority predefined by the application for the request, which is 0 by default, and B is the I/O size in gigabytes. In the case the application does not define the request priority, normal host I/Os typically own a higher priority when they contain a smaller amount of data. If the application requires that the ISP task should be executed as soon as possible, then the related request can also be set to a high priority.

$$Priority\,(Req) = A + \frac{1}{B} \tag{1}$$

The request scheduling policy is optimized for reducing the latency of normal host I/Os, which is achieved via a preemption mechanism for high-priority requests to low-priority requests. Specifically, low-priority requests can be suspended and restored when they are preempted. The key reason for such a scheduling policy to work effectively is that ISP tasks are mostly I/O-intensive applications. For example, the amount of data that an ISP task needs to access for a single flash chip may reach 100 MB or even more. Considering the computational overhead, it usually takes a few or tens of seconds to execute an ISP task. To reduce the latency of normal host I/Os, TH-iSSD supports high-priority requests to be inserted into the request queue in a way that the requests in the queue are organized in a descending order of priority. Moreover, large ISP I/O requests are cut into multiple smaller pieces. Therefore, normal host I/Os are scheduled to be executed immediately after a small piece of ISP I/O has been processed, which avoids waiting for the ISP tasks to be completely executed. As is shown in Figure 5, all NAND flash dies are organized in X channels and Y luns. Each die has a unique request queue. Every request is divided into small pieces according to data address so that data of every piece is not distributed across dies. Specially, compute requests in a request queue are further divided into several smaller parts. There are three requests in flight,

Table 1. Description of TH-iSSD's Host-Side Programming Model

Name	Description
`char* fsopen(const char *path, int flags)`	Open a *filestream* and return its data layout.
`int fsclose(int fd)`	Close a *filestream*.
`int fsread(int fd, void *buf, size_t size)`	Read a *filestream* processed by the ISP unit.
`int fswrite(int fd, void *buf, size_t size)`	Write a *filestream* to the device.
`int reg_task(void *buf, size_t size)`	Register a computation task to the device.
`int send_params(void *buf, size_t size)`	Send necessary parameters to the device.

numbered #1, #2, and #3. Request #1 is a 3-GB compute request whose predefined priority is 0, so its priority is 0.33. Request #2 is a 1-MB write request with 0 as its predefined priority, so its priority is 1,024. Request #3 is a 2-MB read request that is assigned 2,000 as predefined priority by applications, so its priority is 2,512. The scheduling process is shown in the following three steps.

(1) When compute request #1 arrives at TH-iSSD, it is divided into different dies. As far as the request to the die of channel 0 and lun #Y is concerned, it is further divided into three parts, called P_0, P_1, and P_2. Since the request queue is empty at the beginning, P_0 is executed first and other two parts are waiting to be executed.
(2) Write request #2 arrives at TH-iSSD before P_0 of compute request #1 is finished. Because the priority of write request #2 is 1,024, which is higher than that of compute request #1, write request #2 is inserted before P_1 and P_2 of compute request #1.
(3) Read request #3 also arrives at TH-iSSD before P_0 of compute request #1 is finished. Because the priority of read request #3 is 2,512, higher than that of write request #2, read request #3 is inserted before write request #2 to keep requests in the queue in a descending order of priority. Once P_0 of compute request #1 is finished, read request #3 will be executed immediately.

As for the potential data conflicts among the three parties (data collector, host memory, and SSD storage), TH-iSSD resolves the problem in two steps. First, TH-iSSD only treats the host memory as the temporary buffer and no data would be modified by the data collector or host CPU. Therefore, TH-iSSD only needs to deal with the data conflicts within the ISP engine and the SSDs. Since TH-iSSD implements a customized full-function FTL for both I/O and computation requests, the data conflicts, such as read-after-write or write-after-write, are resolved within the FTL.

3.4 Host-Side Programming Model

TH-iSSD should expose a group of APIs to allow developers to easily make use of fast ISP without any efforts to write their own custom interfaces manually. We therefore introduce *filestream*, which is the host-side programming model of TH-iSSD. *Filestream* is an abstraction with a subset of the POSIX-like APIs (shown in Table 1) that allow host applications to issue both data path and computation path operations to the device. By accessing *filestream*, the underlying data movements or ISP tasks will be triggered transparently. Since the abstraction of *filestream* is familiar to existing applications (which typically use POSIX interfaces to access files), traditional application code can be easily transferred into the TH-iSSD host code. Similar to existing approaches [12, 19, 54], TH-iSSD neither implements the full set of POSIX-compatible operations nor provides crash consistency or other POSIX semantics. This is because the primary reason for introducing the file abstraction in our article is its familiar programming interfaces; being fully POSIX-compatible will introduce extra software overhead, which is unnecessary in most cases.

File Open/Close. Like a normal host-side file system does, the `fsopen` method creates a file descriptor for the opened file. The only difference is that the related file mapping information is returned back and kept in the TH-iSSD runtime. TH-iSSD will rely on the mapping information to help the device execute the offloaded ISP tasks. By invoking the `fsclose` call, except for destroying the file descriptor, the data layout of this file is removed as well.

File Read/Write. `fsread` and `fswrite` are used to enable the ISP functions inside the device. For example, by invoking an `fsread` method, the TH-iSSD runtime first obtains the mapping information based on the parameters provided by this call, and these mappings are then sent to the device using a special command (e.g., by using a reserved command code of the NVMe protocol); after receiving the command, the ISP unit reads the related data according to the mappings and preprocesses the data before sending it back to the host. Note that TH-iSSD still uses legacy POSIX interfaces to read or write a file when the data residing in it does not need to be preprocessed.

`reg_task` and `send_params` are used when the offloaded task in the device needs to be renewed. Like read or write calls, these two methods are also sent to the device using a special command code.

4 DEPLOYMENT AND EXPERIMENT

In this section, we describe how we apply TH-iSSD in two applications—that is, information retrieval systems and streaming systems—and illustrate how TH-iSSD contributes to improving their performance. At the end of each use case, we report some experimental results to reveal the benefits that TH-iSSD brings.

4.1 Information Retrieval System with TH-iSSD

CBIR systems are used to search for certain data items from a large-scale unstructured data warehouse. Typical examples include identifying the same person in an image collection, searching music songs with similar styles, and so on. CBIR typically consists of two major components: feature extraction and database indexing. Among them, the former extracts the feature vector for each data item, whereas the latter organizes feature vectors so that similar data items can be queried effectively. The popularity of high-definition digital cameras and ubiquitous IoT sensors has created explosive growth in the quantity of the dataset. As a result, how to search in such collections to find target data items with both high accuracy and cost efficiency becomes a major challenge.

Traditional solutions that find similar data items by using text-based search are shown to be neither accurate nor cost-efficient, since they require manually providing tags and the relevance between tags is hard to quantify. Fortunately, the rise of deep learning eases the problem of feature extraction by incorporating **Deep Convolutional Neural Network (DCNN)**-based features [37]. Recent studies have shown that the DCNN provides better representations and thus delivers high accuracies [38].

However, there is no such thing as a free lunch—the neural networks are highly non-linear, which cannot preserve geometric properties between feature vectors, preventing us from building an efficient index for these feature vectors. As a result, one has to scan the entire dataset to retrieve data items with high accuracy—this not only wastes computation resources but also incurs frequent data movement between host CPUs and external storage devices. Mailthody et al. [42] reveal that in DCNN-based CBIR systems, the storage I/O bandwidth is still the major bottleneck that occupies 56% to 90% of the total execution time.

To overcome the I/O bottleneck, we propose to push the feature extraction and indexing logic inside the drive with the TH-iSSD framework. In the following, we will first depict the hardware platform we choose to deploy TH-iSSD and how we implement the CBIR system on top of this platform along with several optimizations.

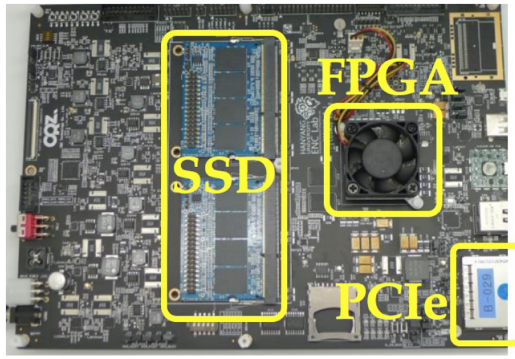


Fig. 6. Hardware platform of information retrieval system.

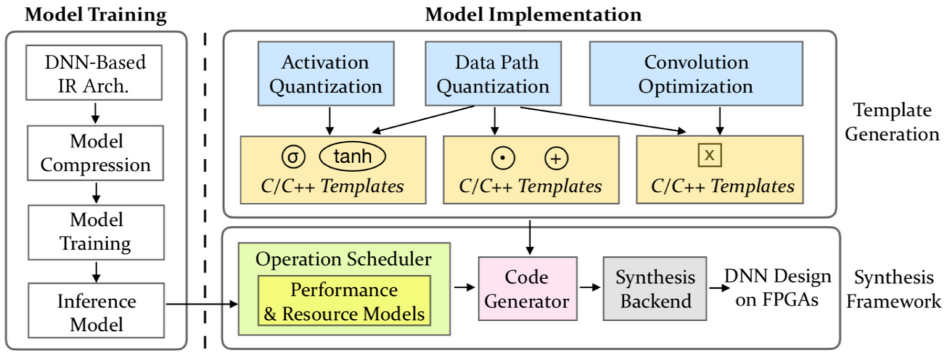


Fig. 7. Information retrieval system optimization framework overview.

4.1.1 Hardware Platform Description. The hardware platform used to validate the proposed TH-iSSD framework is composed of an XC7Z045 FPGA chip, 1 GB of DRAM, two NAND flash modules, an eight-way NAND interface, and a PCIe Gen2 eight-lane interface and other modules, as shown in Figure 6. As for the NAND flash chip controller, the interface is implemented on the XC7Z045 FPGA. The smart module is connected to the controller through four-way NAND interfaces, each carrying two chips, each which contains four dies, and the page size is 16 KB. The DRAM is used to cache data so as to speed up read and write operations.

The TH-iSSD system is deployed on the FPGA-based platform by integrating both a customized SSD FTL controller and the feature extraction/indexing engine. As shown in Figure 8, the customized SSD FTL is used to manage the data flow between the SSD and host CPU. When a CBIR request is received by the host, it is sent to the CBIR engine. The engine first fetches the target images from the SSD to the feature extraction accelerator implemented in the FPGA and then searches the matched results given by the indexing accelerator. Compared to the traditional architecture, the data transfers only occur between the FPGA and the SSD and thus saves the data communication time and efforts from and to the host CPU.

To facilitate the deployment of the proposed TH-iSSD system to various FPGA platforms, we propose an automatic hardware flow to accelerate the information retrieval system that is shown in Figure 7. It consists of two parts: the operator templates generation (upper part) and automatic synthesis framework (lower part). Since the number of primitive operators of **Deep Neural Network (DNN)**-based information retrieval system is limited, we propose to manually write the template for each primitive operator. As for the DNN retrieval algorithms studied in this work, we define

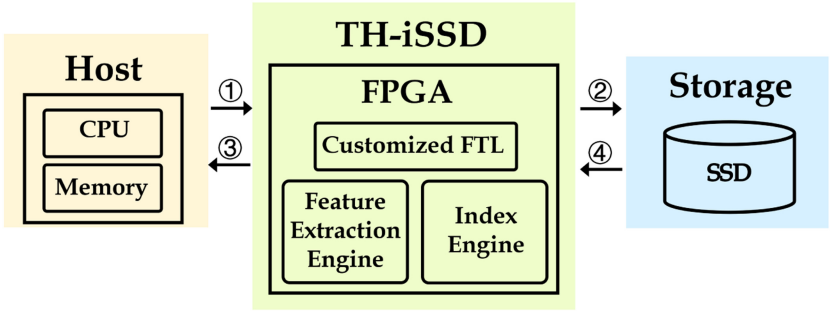


Fig. 8. Hardware architecture of CBIR deployment on TH-iSSD.

hyperbolic tangent *tanh*, sigmoid σ, point-wise vector addition, point-wise vector multiplication, and convolution as primitive operators. The optimization techniques including loop unrolling and pipelining are all applied to these operators. It is necessary to note that the proposed primitive operator templates are general enough to implement almost any kind of DNN variant, to the best of our knowledge.

The automatic synthesis framework is fed with the well-trained inference model provided by the software flow. Then, a directed acyclic data dependency graph is generated to represent the computation flow of the DNN. The operators in the graph are scheduled to compose a multi-stage coarse-grained pipeline so as to maximize the performance under certain resource constraints with the help of analytical performance and resource models. The scheduling result is then given to the code generator. The code generator takes the operator scheduling result as input and generates the final C/C++-based code automatically by integrating the associated primitive operator templates together. Since the interface of each template is well defined and the tunable parameters are expressed using C/C++ macros, the code generation is quite efficient. The synthesis backend, which is an off-the-shelf commercial HLS tool, accepts the C/C++ code as input and outputs the optimized accelerator implementation on FPGAs.

4.1.2 Experimental Results. Next, we conduct experiments to analyze the performance of TH-iSSD by comparing against its baseline system.

Experimental Setup. To validate the performance advantage brought by TH-iSSD, we implement a baseline system that adopts a *compute-centric* architecture where the host CPU dispatches workloads to the FPGA-based accelerator and forwards data between the FPGA and storage devices. We employ three benchmarks from *Natural Language Processing* [61], *Medical Imaging* [50], and *Satellite Remote Sensing* scenarios to analyze the proposed system with respect to end-to-end performance (in terms of both throughput and latency) and power consumption in the following. To show the superiority of the proposed TH-iSSD design, we set the CBIR inference dataset to 350 GB, which is way larger than the host memory capacity.

Throughput. Figure 9 shows the throughput of TH-iSSD under different types of workloads by varying the batch size. Overall, TH-iSSD delivers good performance and outperforms the baseline significantly. Specifically, with a batch size of 1, TH-iSSD's throughput is 1.32×, 1.33×, and 1.66× higher than that of the baseline for *Remote Sensing*, *Object Detection*, and *Natural Language Processing*, respectively. As the batch size increase further, the performance gap between the two systems increases as well, where TH-iSSD achieves a speedup of 1.40× on average.

Discussion. The end-to-end performance of the proposed TH-iSSD design is constrained by two factors: computation ability and the data transfer rate. The FPGA-based hardware acceleration engine employed in TH-iSSD is quite efficient, and the bottleneck is the SSD access bandwidth. As

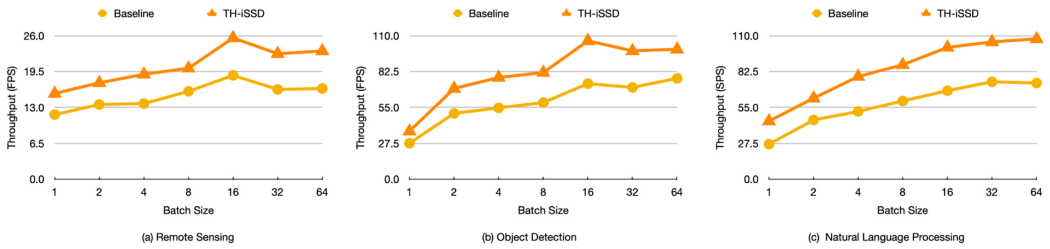


Fig. 9. Throughput of the information retrieval system.

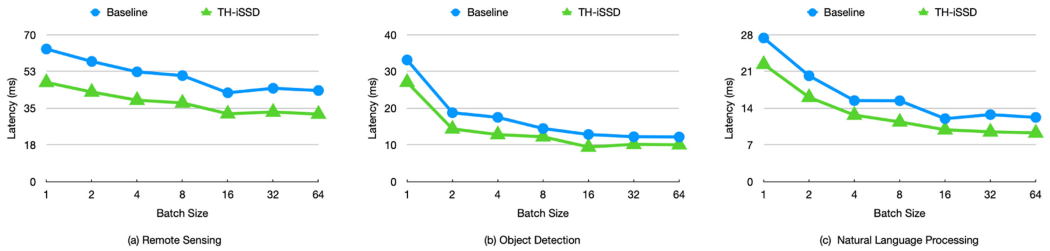


Fig. 10. Latency of the information retrieval system.

for the remote sensing benchmark, when the batch size reaches 8, the throughput and latency no longer improve due to the saturated data transfer channels between the FPGA and the SSD. The same performance scaling trend also appears in the rest of the two benchmarks.

Latency. Figure 10 depicts the latency of the compared systems by running the same workloads as those used in the throughput evaluation discussion. Specifically, latency is measured as the time for the CBIR system to process a query, including generating the feature vector of the query's data, and fetching related data items in the indexing database. We can observe from the figure that TH-iSSD makes much faster progress in query execution as the batch size increases. On average, TH-iSSD's delay is 25.3%, 20.8%, and 21.5% shorter than that of the baseline under different types of workloads.

4.2 Edge Zero-Streaming System with TH-iSSD

Pervasive cameras and sensors at the edge continuously generate new data, which needs to be analyzed in time to capture the events in which we are interested. Streaming processing is different from CBIR systems in the following aspects: CBIR systems build an indexing database offline and react to a user's queries if necessary, which exhibit read-dominated access patterns and are designed to deliver high throughput and low latency for individual queries. Instead, steaming processing systems should analyze the constantly produced new data from data collectors in line rate (both read- and write-intensive), which necessitate real-time processing capabilities.

Edge streaming systems can be roughly divided into two categories: (1) streaming data on a shared network back to the data center for analysis (i.e., full streaming) and (2) analyzing data locally in the device and communicating with the cloud only when queries are requested (i.e., zero streaming). In *full streaming*, transmitting data in real time to the cloud often incurs high latency and wastes wireless bandwidth unnecessarily. This is because the collected data is often cold regarding that interesting events are often unforeseeable and rare. In *zero streaming*, edge devices eliminate the need to stream data to the cloud; however, the device itself only contains very limited hardware resources (DRAM capacity, computation power, etc.), which are hard to process data at capture time.

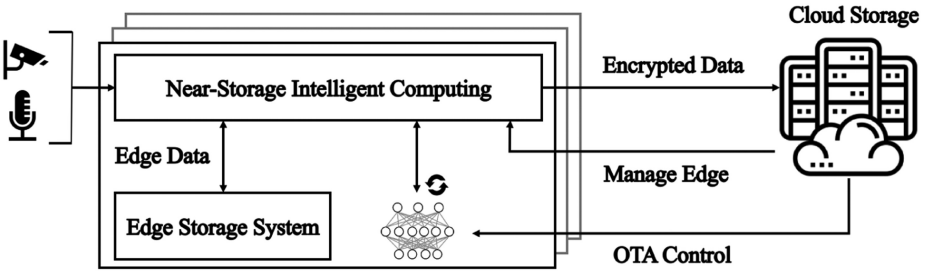


Fig. 11. EZS overview.

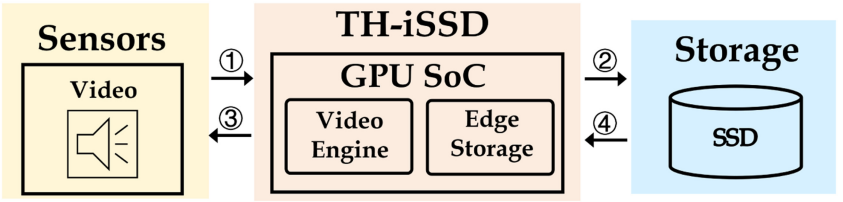


Fig. 12. Hardware architecture of EZS deployment on TH-iSSD.

As we can see, our proposed TH-iSSD is a good fit for such edge devices, providing an efficient computation and data path for real-time stream processing. Next, we describe the hardware platform configuration that is suitable to be deployed at the edge and how TH-iSSD can be deployed in an edge streaming system. As for the **Edge Zero-Streaming System (EZS)** implementation (Figure 11), the TH-iSSD system is deployed on the GPU SoC-based platform by integrating both customized edge storage and the video processing engine. As shown in Figure 12, the customized edge storage engine is used to manage the data flow between the SSD and the GPU SoC. When the EZS request is constantly received by the SoC, it is immediately sent to the EZS engine. The engine first processes the videos through the GPU video processing engine and then stores the results in the local edge storage system. Compared to the traditional architecture, the data transfers only occur between the GPU SoC and the SSD and thus save huge amounts of the data communication time and effort.

4.2.1 Hardware Platform Description. NVIDIA Xavier is the latest SoC in the NVIDIA AGX Xavier and Drive AGX Pegasus platforms, as shown in Figure 13. Xavier has an eight-core "Caramel" CPU (based on ARM V8) and an eight-core Volta GPU with 512 CUDA cores, and 16 GB of shared main memory. We define *host* as the designated term given to the CPU and the supporting architecture that enables memory and instructions to be sent to GPU. Kernels are set up by the developer and consist of threads, thread-blocks, and grids. Memory is a crucial part of kernel execution. Prior to execution, data has to be introduced to the GPU in some way. After the kernel finishes the execution, the data needs to be sent back, or the host needs to be informed depending on the memory management technique.

The proposed NDP framework for the EZS is shown in Figure 11. The workflow of the EZS is done in two steps. During the video recording period, the EZS will do its best to build the target label to acquire long-term video knowledge. The target label is very likely, such as one for every 30 frames per second; recording a large number of object labels, they provide reliable knowledge-spatial distribution of various objects in the long-term video. During query execution, the EZS runs a small DNN, such as YOLO, to upload frames locally and continuously upgrade the cloud

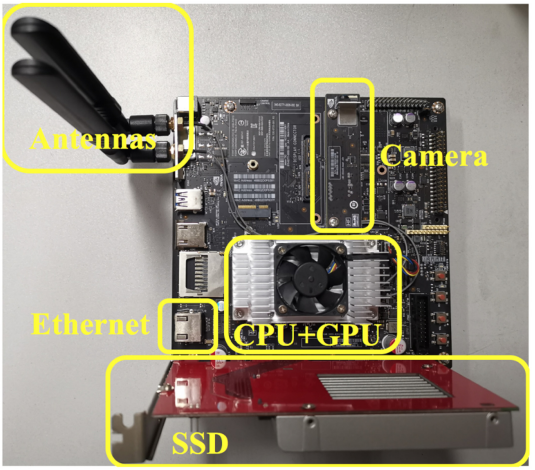


Fig. 13. Hardware platform of the EZS.

intelligent index. As shown in Figure 11, after the cloud receives the query, it first retrieves all target labels in the query video range from the smart index. The camera runs a lightweight DNN to determine the priority of the query frame to be uploaded. The operator scores the frames; a higher score indicates that the frame is more likely to contain any objects of interest. The preceding steps are repeated until the query is aborted or completed. Throughout the entire query process, the cloud continuously optimizes the results presented to users.

4.2.2 Experimental Results. We also evaluate the performance of TH-iSSD in a zero-streaming application with similar workloads. Our experiments are conducted differently from those in the CBIR system in the following aspects: first, the baseline system still requires the host CPU for coordination but replaces its accelerator with an embedded GPU (NVDIA Xavier); second, the streaming application processes workloads to extract indexing information in real time instead of reacting to the user's queries.

Throughput. Figure 14 presents a throughput comparison that runs three types of workloads with varying batch sizes. We can observe that our ISP solution performs better than the baseline system consistently as we change the workloads and batch sizes, which achieves 1.45× better throughput over the baseline on average. The experimental results also show a trend similar to that in Section 4.1, where the performance gap between the two systems increases as the batch size becomes bigger.

Latency. Figure 15 shows the latency of the compared systems on various workloads. Specifically, TH-iSSD reduces the end-to-end latency by up to 36.8% compared with the baseline system. Our solution achieves low latency by allowing data to be flowed between different hardware components directly, which eliminates the communication delays incurred by external connections. Moreover, TH-iSSD does not reply on the host CPU for data forwarding, which further shortens the data path.

5 RELATED WORKS

5.1 In-Storage Processing

Active Disks. The idea of NDP was proposed as early as the past century. A line of research has investigated the idea of pushing computation to HDDs. Among them, some works [4, 25, 52] offer details on the use of active disks for the data center. IDISKS [25] presents an "intelligent" disk to

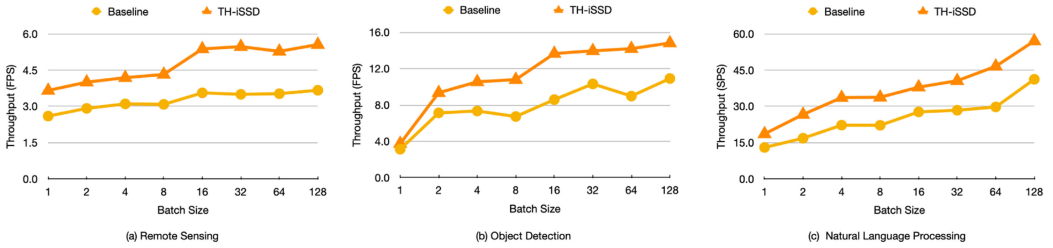


Fig. 14. Throughput of the EZS.

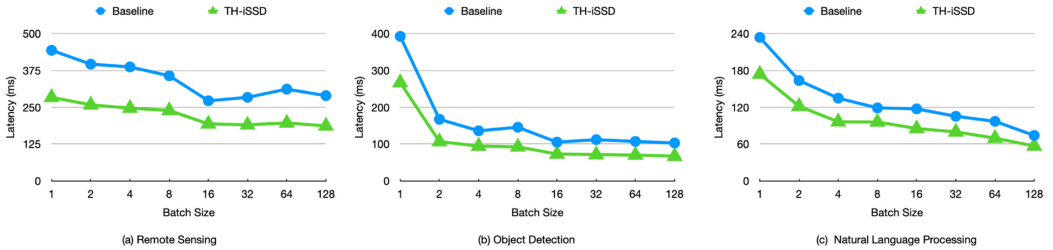


Fig. 15. Latency of the EZS.

overcome the I/O bottleneck of a conventional storage system by integrating embedded processors within the drive with high-speed serial links. ActiveDisk [53] presents an analytical model and explores the possibility of the speedups in an active storage system for a specific application, including data mining and multimedia. Riedel [51] predicts the supporting factors for the success of active disks and the characteristics of applications that are suitable for active disks. The idea of adopting the concept of NDP in magnetic storage devices, however, has limited cost-effectiveness due to the magnetic disk latency and relatively small input/output size.

Offloading Individual Operators. ISP has been extensively studied in the past decades, but most works focus on offloading popular but inherently simple operations, such as sort, join, and query, into the device [4, 17, 25, 35, 52, 53, 55, 62, 70]. For example, Kim et al. [27] proved through simulations that the use of a dedicated SoC chip can greatly accelerate the *scan* operation and reduce energy consumption. Later, they further used a dedicated SoC chip to accelerate the join operation [28], which improves performance by 5×. Woods et al. [69] designed a hardware accelerator based on the FPGA to accelerate Group-By and Filter operations. There is also a line of research that proposes general ISP models that help embedded accelerators execute the offloaded tasks more efficiently. For instance, Active Flash [62] presents an analytic model to systematically evaluate the potential for in-SSD computation; Summarizer [29] introduces a set of programming interfaces to help applications offload tasks to the device, which shares a similar design goal as ours. Biscuit [20] introduces a flow-based programming model, where offloaded tasks are processed in a way similar to task graphs with data pipes. Tseng et al. [63] propose a new model—Morpheus—and an SSD prototype based on Morpheus, which allows applications to move deserialization operations from the host to the storage device, saving CPU and memory for computationally intensive workloads, reducing power consumption, and saving I/O bandwidth. Although the preceding approaches achieve impressive performance, host-side CPUs still need to be frequently involved in either moving data or executing the application logic.

Graph Computing. Graph processing is a sort of computation task that generates small-sized and random I/Os, which is considered to be I/O intensive and requires frequent accesses to the graph

in storage when processing large-scale graphs. Consequently, graph processing fits well to the ISP paradigm. Many prior studies have intensively explored the usage of ISC in graph processing, which achieved competitive performance and energy efficiency [23, 31–33, 41, 45, 60, 72]. For example, GraphSSD [45] is a graph semantic aware SSD framework that provides a full system solution for storing, accessing, and performing graph analytics on SSDs. GraphOne [32] proposes a new real-time graph analytics framework that takes account of the storage engine and makes the data logging and archiving quite efficient. The data management mechanism is data centric and reduces the data movements between host CPUs and the storage. FlashGraph [72] manages the frequently accessed graph data in DRAM, which avoids the possible I/O bottleneck in graph processing. MOSAIC [41] assigns concentrated, memory-intensive operations (i.e., vertex-centric operations on a global graph) to fast host processors (scale-up) and offloads the compute- and I/O-intensive components (i.e., edge-centric operations on local graphs) to co-processors (scale-out), and thus achieves both high performance and cost efficiency. To overcome the limited computing capability problem of existing drives that are typically equipped with low-end processors, many powerful hardware accelerators (e.g., FPGAs or specialized accelerators) have been built in the context of ISP in recent years. For example, GraFBoost [23] introduces the sort-reduce accelerator, which logs random update requests and then uses hardware-accelerated external sorting with interleaved reduction functions to sequentialize fine-grained random accesses to flash storage.

Intelligent Drives. The idea of embedding machine learning models within the drive (and the device therefore becomes an intelligent drive) has been proposed in recent years, which is devoted to reducing the amount of data that should be exchanged between host CPUs and devices. Kang et al. [24] build an intelligent SSD prototype that supports the Hadoop MapReduce framework based on a SATA SSD. The intelligent SSD takes advantage of the computing power of an in-device processor to accelerate data processing, achieving low power consumption, high parallelism, and low memory consumption. Choe et al. [16] propose to accelerate the stochastic gradient descent algorithm with a multi-core processor inside the SSD, and parallelize the SGD process utilizing the multi-channel feature of flash memory. Cognitive SSD [36] is an unstructured data retrieval engine based on deep learning. It has a flash-accessing accelerator for near-data deep learning, which is implemented with an FPGA inside the SSD, supporting to perform inference inside storage devices. iSSD [15] integrates stream processors into the flash memory controllers to accelerate linear regression, *k*-means, decision tree classification, and naive Bayesian workloads, which is significantly better than traditional SSDs in terms of both energy efficiency and performance. However, all preceding approaches, including graph processing, are restricted to the context of a single scenario, whereas TH-iSSD is designed to be deployed to accelerate different applications.

5.2 Hardware Accelerators

PIM is an emerging hardware architecture that directly uses the physical characteristics of the memory cell to perform calculations. The basic hardware units of PIM include ReRAM [6, 39, 40, 68] and PCM [5, 11, 64]. PIM supports vector-matrix multiplication operations, bitwise logic operations, and search operations, which are commonly used in AI or graph computation tasks. ISAAC [56] uses a ReRAM array as the calculation unit for vector-matrix multiplication and eDRAM as the storage unit, and pipelines inference calculation to improve the efficiency of neural network inference. Song et al. [59] copy multiple computing units to reduce bubbles in the pipeline, thereby improving the efficiency of PIM for neural network training tasks. LerGAN [44] is a ReRAM-based PIM that eliminates redundant calculations and storage overhead of the GAN through software and hardware co-design. There has also been a line of research designing dedicated accelerators for kNNs [43, 71], CNNs, and DNNs [14]. For instance, DianNao [14] exploits the locality properties of large layers and customized storage structures to realize both good

performance and broad application scope in accelerators for machine learning. Later on, this research group further introduced a series of hardware accelerators designed for neural networks (i.e., DianNao family), with a special emphasis on the impact of memory on accelerator design, performance, and energy. Google developed TPU [3], a custom ASIC for neural network machine learning. PIM and dedicated accelerators achieve impressive performance for neural network training and inference; however, they require a fundamental redesign of both software and hardware, which is unlikely to be widely deployed in the near future.

Felix et al. [18] utilize ISP in storage servers deploying parallel file systems, which reduces data movement overhead and greatly improves overall system performance. Vincon et al. [65] propose a key-value storage system based on ISP, which can directly control the data layout and calculation execution in the storage hardware, and synchronize the data layout and data format with software and hardware parsers, so that ISP operations can be executed efficiently without modification on the host software stack. Park et al. [49] integrate the Hadoop MapReduce framework with SSD devices with ISP capabilities, and propose an ISP-friendly Hadoop system by implementing Hadoop Mapper in SSD firmware and offloading Map tasks from the host to the SSD. Kufeldt et al. [30] propose an NDP-friendly storage device interface and use this interface to offload tasks such as data recovery and migration from the host processor to the processor in the device for execution, significantly improving performance.

Andersen et al. [7] propose a low-power cluster architecture—FAWN—for data-intensive workloads and build a consistent, replicable, and high-availability key-value storage system FAWN-KV based on FAWN. FAWN integrates low-power processors with flash storage devices to balance computing and I/O on the host side, and achieves large-scale and efficient parallel access to data. Ouyang et al. [48] analyze the unique characteristics of computing tasks in network data analysis, eliminate the interference between normal data processing and SSD internal computing tasks, design an active SSD, and implement a system prototype. Based on flash memory storage devices, Jun et al. [22] propose a new system architecture—BlueDBM—for near-data computing, which can achieve a better performance price ratio in big data analysis scenarios.

6 CONCLUSION

This article presented TH-iSSD, a full-stack redesigned ISP framework to accelerate I/O-intensive applications. We would like to point out that not all techniques used in TH-iSSD are new. Instead, our key technical contribution lies in the following two aspects. First, the sensing, computation, and storage components are highly integrated in TH-iSSD, which incurs minimal data movement overhead. Second, instead of purely considering how to compute faster in the device like recent studies, we primarily focus on how to manage data in the device in an ISP-aware manner. We applied TH-iSSD in two representative applications (i.e., CBIR and streaming systems), and experimental results showed that TH-iSSD achieves significant higher throughput and lower latency than *compute-centric* designs.

REFERENCES

[1] Intel. (n.d.). Intel Optane SSD DC P4800X. Retrieved February 27, 2023 from https://www.intel.com/content/www/us/en/solid-state-drives/optane-ssd-dc-p4800x-brief.html.

[2] Samsung. (n.d.) Samsung NVMe SSD 960 Pro. Retrieved February 27, 2023 from https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ssd-960-pro-m-2-512gb-mz-v6p512bw/.

[3] Google Cloud. (n.d.) Cloud TPU. Retrieved February 27, 2023 from https://cloud.google.com/tpu.

[4] Anurag Acharya, Mustafa Uysal, and Joel Saltz. 1998. Active disks: Programming model, algorithms and evaluation. *ACM SIGOPS Operating Systems Review* 32, 5 (1998), 81–91.

[5] Sapan Agarwal, Robin B. Jacobs Gedrim, Alexander H. Hsia, David R. Hughart, Elliot J. Fuller, A. Alec Talin, Conrad D. James, Steven J. Plimpton, and Matthew J. Marinella. 2017. Achieving ideal accuracies in analog neuromorphic

computing using periodic carry. In *Proceedings of the 2017 Symposium on VLSI Technology*. IEEE, Los Alamitos, CA, T174–T175.

[6] Hiroyuki Akinaga and Hisashi Shima. 2010. Resistive random access memory (ReRAM) based on metal Oxides. *Proceedings of the IEEE* 98, 12 (2010), 2237–2251. https://doi.org/10.1109/JPROC.2010.2070830

[7] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. 2009. FAWN: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. 1–14.

[8] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H. Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. 2014. Near-data processing: Insights from a MICRO-46 workshop. *IEEE Micro* 34, 4 (2014), 36–42. https://doi.org/10.1109/MM.2014.55

[9] Rajeev Balasubramonian and Boris Grot. 2016. Near-data processing [Guest editors' introduction]. *IEEE Micro* 36, 01 (2016), 4–5.

[10] Antonio Barbalace, Anthony Iliopoulos, Holm Rauchfuss, and Goetz Brasche. 2017. It's time to think about an operating system for near data processing architectures. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. 56–61.

[11] Geoffrey W. Burr, Robert M. Shelby, Severin Sidler, Carmelo Di Nolfo, Junwoo Jang, Irem Boybat, Rohit S. Shenoy, et al. 2015. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Transactions on Electron Devices* 62, 11 (2015), 3498–3507.

[12] Mike Burrows. 2006. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*. 335–350.

[13] Adrian M. Caulfield, Todor I. Mollov, Louis Alex Eisner, Arup De, Joel Coburn, and Steven Swanson. 2012. Providing safe, user space access to fast, solid state disks. *ACM SIGARCH Computer Architecture News* 40, 1 (March 2012)(2012), 387–400. https://doi.org/10.1145/2189750.2151017

[14] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News* 42, 1 (2014), 269–284.

[15] Sangyeun Cho, Chanik Park, Hyunok Oh, Sungchan Kim, Youngmin Yi, and Gregory R. Ganger. 2013. Active disk meets flash: A case for intelligent SSDs. In *Proceedings of the 27th International ACM Conference on Supercomputing*. 91–102.

[16] Hyeokjun Choe, Seil Lee, Hyunha Nam, Seongsik Park, Seijoon Kim, Eui-Young Chung, and Sungroh Yoon. 2016. Near-data processing for differentiable machine learning models. *arXiv preprint arXiv:1610.02273* (2016).

[17] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. 2013. Query processing on smart SSDs: Opportunities and challenges. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. ACM, New York, NY, 1221–1230. https://doi.org/10.1145/2463676.2465295

[18] Evan J. Felix, Kevin Fox, Kevin Regimbal, and Jarek Nieplocha. 2006. Active storage processing in a parallel file system. In *Proceedings of the 6th LCI International Conference on Linux Clusters: The HPC Revolution*. 85.

[19] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, New York, NY, 29–43. https://doi.org/10.1145/945445.945450

[20] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, et al. 2016. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA'16)*. IEEE, Los Alamitos, CA, 153–165. https://doi.org/10.1109/ISCA.2016.23

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[22] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, et al. 2015. BlueDBM: An appliance for big data analytics. In *Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. IEEE, Los Alamitos, CA, 1–13.

[23] Sang Woo Jun, Andy Wright, Sizhuo Zhang, Shuotao Xu, and Arvind. 2018. GraFBoost: Using accelerated flash storage for external graph analytics. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA'18)*. IEEE, Los Alamitos, CA, 411–424. https://doi.org/10.1109/ISCA.2018.00042

[24] Yangwook Kang, Yang-Suk Kee, Ethan L. Miller, and Chanik Park. 2013. Enabling cost-effective data processing with smart SSD. In *Proceedings of the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST'13)*. IEEE, Los Alamitos, CA, 1–12.

[25] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. 1998. A case for intelligent disks (IDISKs). *ACM SIGMOD Record* 27, 3 (1998), 42–52.

[26] Gokcen Kestor, Roberto Gioiosa, Darren J. Kerbyson, and Adolfy Hoisie. 2013. Quantifying the energy cost of data movement in scientific applications. In *Proceedings of the 2013 IEEE International Symposium on Workload Characterization (IISWC'13)*. IEEE, Los Alamitos, CA, 56–65.

[27] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, and Sang-Won Lee. 2011. Fast, energy efficient scan inside flash memory SSDs. In *Proceedings of the International Workshop on Accelerating Data Management Systems (ADMS'11)*.

[28] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, Sang-Won Lee, and Bongki Moon. 2016. In-storage processing of database scans and joins. *Information Sciences* 327 (2016), 183–200.

[29] Gunjae Koo, Kiran Kumar Matam, Te I, H. V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: Trading communication with computing near storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50'17)*. ACM, New York, NY, 219–231. https://doi.org/10.1145/3123939.3124553

[30] Philip Kufeldt, Carlos Maltzahn, Tim Feldman, Christine Green, Grant Mackey, and Shingo Tanaka. 2018. Eusocial storage devices: Offloading data management to storage devices that can act collectively. *;login: Usenix Magazine* 43, 2 (2018), 16–22.

[31] Pradeep Kumar and H. Howie Huang. 2016. G-store: High-performance graph store for trillion-edge processing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'16)*. IEEE, Los Alamitos, CA, 830–841.

[32] Pradeep Kumar and H. Howie Huang. 2020. GraphOne: A data store for real-time analytics on evolving graphs. *ACM Transactions on Storage* 15, 4 (2020), 1–40.

[33] Jinho Lee, Heesu Kim, Sungjoo Yoo, Kiyoung Choi, H. Peter Hofstee, Gi-Joon Nam, Mark R. Nutter, and Damir Jamsek. 2017. ExtraV: boosting graph processing near storage with a coherent accelerator. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1706–1717.

[34] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. SmartSSD: FPGA accelerated near-storage data analytics on SSD. *IEEE Computer Architecture Letters* 19, 2 (2020), 110–113.

[35] Chao Li, Yang Hu, Longjun Liu, Juncheng Gu, Mingcong Song, Xiaoyao Liang, Jingling Yuan, and Tao Li. 2015. Towards sustainable in-situ server systems in the big data era. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. ACM, New York, NY, 14–26. https://doi.org/10.1145/2749469.2750381

[36] Shengwen Liang, Ying Wang, Youyou Lu, Zhe Yang, Huawei Li, and Xiaowei Li. 2019. Cognitive SSD: A deep learning engine for in-storage data retrieval. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC'19)*. 395–410. https://www.usenix.org/conference/atc19/presentation/liang.

[37] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. 2015. Deep learning of binary hash codes for fast image retrieval. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'15)*. 27–35. https://doi.org/10.1109/CVPRW.2015.7301269

[38] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. 2016. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2064–2072.

[39] Qi Liu, Jun Sun, Hangbing Lv, Shibing Long, Kuibo Yin, Neng Wan, Yingtao Li, Litao Sun, and Ming Liu. 2012. Real-time observation on dynamic growth/dissolution of conductive filaments in oxide-electrolyte-based ReRAM. *Advanced Materials* 24, 14 (2012), 1844–1849.

[40] Tz-Yi Liu, Tian Hong Yan, Roy Scheuerlein, Yingchang Chen, Jeffrey KoonYee Lee, Gopinath Balakrishnan, Gordon Yee, et al. 2014. A 130.7-mm^2 2-layer 32-Gb ReRAM memory device in 24-nm technology. *IEEE Journal on Solid State Circuits* 49, 1 (2014), 140–153. https://doi.org/10.1109/JSSC.2013.2280296

[41] Steffen Maass, Changwoo Min, Sanidhya Kashyap, Woonhak Kang, Mohan Kumar, and Taesoo Kim. 2017. Mosaic: Processing a trillion-edge graph on a single machine. In *Proceedings of the 12th European Conference on Computer Systems*. 527–543.

[42] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia De Gonzalo, Youjie Li, Hubertus Franke, Jinjun Xiong, Jian Huang, and Wen-Mei Hwu. 2019. DeepStore: In-storage acceleration for intelligent queries. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 224–238.

[43] Elias S. Manolakos and Ioannis Stamoulias. 2010. IP-cores design for the kNN classifier. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, Los Alamitos, CA, 4133–4136.

[44] Haiyu Mao, Mingcong Song, Tao Li, Yuting Dai, and Jiwu Shu. 2018. LerGAN: A zero-free, low data movement and PIM-based GAN architecture. In *Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. IEEE, Los Alamitos, CA, 669–681.

[45] Kiran Kumar Matam, Gunjae Koo, Haipeng Zha, Hung-Wei Tseng, and Murali Annavaram. 2019. GraphSSD: Graph semantics aware SSD. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*. ACM, New York, NY, 116–128. https://doi.org/10.1145/3307650.3322275

[46] Micron. 2017. Micron NAND Flash by Technology. Retrieved February 27, 2023 from https://www.micron.com/products/nand-flash.

[47] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. 2014. Scale-out NUMA. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. ACM, New York, NY, 3–18. https://doi.org/10.1145/2541940.2541965

[48] Jian Ouyang, Shiding Lin, Zhenyu Hou, Peng Wang, Yong Wang, and Guangyu Sun. 2013. Active SSD design for energy-efficiency improvement of web-scale data analysis. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'13)*. IEEE, Los Alamitos, CA, 286–291.

[49] Dongchul Park, Jianguo Wang, and Yang-Suk Kee. 2016. In-storage computing for Hadoop MapReduce framework: Challenges and possibilities. *IEEE Transactions on Computers*. Early access, July 28, 2016.

[50] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, et al. 2020. MLPerf inference benchmark. In *Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. IEEE, Los Alamitos, CA, 446–459.

[51] Erik Riedel. 1999. *Active Disks: Remote Execution for Network-Attached Storage*. Carnegie Mellon University.

[52] Erik Riedel, Christos Faloutsos, Garth A. Gibson, and David Nagle. 2001. Active disks for large-scale data processing. *Computer* 34, 6 (2001), 68–74. https://doi.org/10.1109/2.928624

[53] Erik Riedel, Garth Gibson, and Christos Faloutsos. 1998. Active storage for large-scale data mining and multimedia applications. In *Proceedings of the 24th Conference on Very Large Databases*. 62–73.

[54] Zhenyuan Ruan, Tong He, and Jason Cong. 2019. INSIDER: Designing in-storage computing system for emerging high-performance drive. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC'19)*. 379–394.

[55] Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, Yanqin Jin, Yang Liu, and Steven Swanson. 2014. Willow: A user-programmable SSD. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. 67–80.

[56] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA'16)*. IEEE, Los Alamitos, CA, 14–26. https://doi.org/10.1109/ISCA.2016.12

[57] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv:1909.08053* [cs.CL] (2020).

[58] Yongseok Son, Nae Young Song, Hyuck Han, Hyeonsang Eom, and Heon Young Yeom. 2014. A user-level file system for fast storage devices. In *Proceedings of the 2014 International Conference on Cloud and Autonomic Computing*. IEEE, Los Alamitos, CA, 258–264.

[59] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*. IEEE, Los Alamitos, CA, 541–552. https://doi.org/10.1109/HPCA.2017.55

[60] Linghao Song, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. 2018. GraphR: Accelerating graph processing using ReRAM. In *Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*. IEEE, Los Alamitos, CA, 531–543.

[61] Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. *arXiv preprint arXiv:1905.05950* (2019).

[62] Devesh Tiwari, Simona Boboila, Sudharshan S. Vazhkudai, Youngjae Kim, Xiaosong Ma, Peter Desnoyers, and Yan Solihin. 2013. Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*. 119–132. https://www.usenix.org/conference/fast13/technical-sessions/presentation/tiwari.

[63] Hung-Wei Tseng, Qianchen Zhao, Yuxiao Zhou, Mark Gahagan, and Steven Swanson. 2016. Morpheus: Creating application objects efficiently for heterogeneous computing. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 53–65.

[64] Yoeri van de Burgt, Ewout Lubberman, Elliot J. Fuller, Scott T. Keene, Grégorio C. Faria, Sapan Agarwal, Matthew J. Marinella, A. Alec Talin, and Alberto Salleo. 2017. A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nature Materials* 16, 4 (2017), 414–418.

[65] Tobias Vincon, Arthur Bernhardt, Ilia Petrov, Lukas Weber, and Andreas Koch. 2020. nKV: Near-data processing with KV-stores on native computational storage. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*. 1–11.

[66] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in Alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18)*. ACM, New York, NY, 839–848. https://doi.org/10.1145/3219819.3219869

[67] Jianguo Wang, Dongchul Park, Yang-Suk Kee, Yannis Papakonstantinou, and Steven Swanson. 2016. SSD in-storage computing for list intersection. In *Proceedings of the 12th International Workshop on Data Management on New Hardware (DaMoN'16)*. ACM, New York, NY, Article 4, 7 pages. https://doi.org/10.1145/2933349.2933353

[68] Zhiqiang Wei, Y. Kanzawa, K. Arita, Y. Katoh, K. Kawai, S. Muraoka, S. Mitani, et al. 2008. Highly reliable TaOx ReRAM and direct evidence of redox reaction mechanism. In *Proceedings of the 2008 IEEE International Electron Devices Meeting*. IEEE, Los Alamitos, CA, 1–4.

[69] Louis Woods, Zsolt István, and Gustavo Alonso. 2014. Ibex—An intelligent storage engine with support for advanced SQL off-loading. *Proceedings of the VLDB Endowment* 7, 11 (2014), 963–974. https://doi.org/10.14778/2732967.2732972

[70] Louis Woods, Jens Teubner, and Gustavo Alonso. 2013. Less watts, more performance: An intelligent storage engine for data appliances. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. ACM, New York, NY, 1073–1076. https://doi.org/10.1145/2463676.2463685

[71] Yao-Jung Yeh, Hui-Ya Li, Wen-Jyi Hwang, and Chiung-Yao Fang. 2007. FPGA implementation of kNN classifier based on wavelet transform and partial distance search. In *Proceedings of the Scandinavian Conference on Image Analysis*. 512–521.

[72] Da Zheng, Disa Mhembere, Randal C. Burns, Joshua T. Vogelstein, Carey E. Priebe, and Alexander S. Szalay. 2015. FlashGraph: Processing billion-node graphs on an array of commodity SSDs. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*. 45–58. https://www.usenix.org/conference/fast15/technical-sessions/presentation/zheng.

Received 31 December 2021; revised 21 April 2022; accepted 28 May 2022