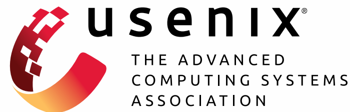

Cognitive SSD: A Deep Learning Engine for In-Storage Data Retrieval

Shengwen Liang and Ying Wang, *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing; University of Chinese Academy of Sciences;* **Youyou Lu and Zhe Yang,** *Tsinghua University;* **Huawei Li and Xiaowei Li,** *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing; University of Chinese Academy of Sciences*

https://www.usenix.org/conference/atc19/presentation/liang

This paper is included in the Proceedings of the 2019 USENIX Annual Technical Conference.

July 10–12, 2019 • Renton, WA, USA

ISBN 978-1-939133-03-8

Open access to the Proceedings of the 2019 USENIX Annual Technical Conference is sponsored by USENIX.

Cognitive SSD: A Deep Learning Engine for In-Storage Data Retrieval

Shengwen Liang[†,*], Ying Wang[†,*,1], Youyou Lu[‡], Zhe Yang[‡], Huawei Li[†,*,1], Xiaowei Li[†,*]
State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing[†],
University of Chinese Academy of Sciences[*], *Tsinghua University*[‡]

Abstract

Data analysis and retrieval is a widely-used component in existing artificial intelligence systems. However, each request has to go through each layer across the I/O stack, which moves tremendous irrelevant data between secondary storage, DRAM, and the on-chip cache. This leads to high response latency and rising energy consumption. To address this issue, we propose Cognitive SSD, an energy-efficient engine for deep learning based unstructured data retrieval. In Cognitive SSD, a flash-accessing accelerator named DLG-x is placed by the side of flash memory to achieve near-data deep learning and graph search. Such functions of in-SSD deep learning and graph search are exposed to the users as library APIs via NVMe command extension. Experimental results on the FPGA-based prototype reveal that the proposed Cognitive SSD reduces latency by 69.9% on average in comparison with CPU based solutions on conventional SSDs, and it reduces the overall system power consumption by up to 34.4% and 63.0% respectively when compared to CPU and GPU based solutions that deliver comparable performance.

1 Introduction

Unstructured data, especially unlabeled videos and images, etc., have grown explosively in recent years. It is reported that the unstructured data occupies up to 80% of storage capacity in commercial datacenters [10]. Once being stored and managed in the cloud machines, the massive amount of unstructured data leads to intensive retrieval requests issued by users, which pose significant challenge to the processing throughput and power consumption of a datacenter [19]. Consequently, it is critical to support fast and energy-efficient data retrieval in the cloud service infrastructure to reduce the total cost of ownership (TCO) of datacenters.

Unfortunately, conventional content-based multimedia data retrieval systems suffer from the issues of inaccuracy, power inefficiency, and high cost especially for large-scale unstructured data. Fig. 1(a) briefly depicts a typical content-based data retrieval system composed of CPU/GPU and conventional storage devices based on a compute-centric architecture [14]. When a data retrieval request arrives from the internet or the central server, the CPU has to reload massive potential data from disk into the temporary DRAM [14] and match the features of the query with those of the loaded unstructured data to find the relevant targets. This compute-centric architecture is confronted with several critical sources of overhead and inefficiency. (1) The current I/O software stack significantly burdens the data retrieval system when it simply fetches data from the storage devices on retrieval requests [60], as shown in Fig. 1(a). The situation is even worse since it is reported the performance bottleneck has migrated from hardware (75∼50us [11]) to software (60.8us [48]) as traditional HDDs are replaced by non-volatile memory [41,48]. (2) Massive data movement incurs energy and latency overhead in the conventional memory hierarchy. This issue becomes severe as the scale of data under query increases because the relevant data at the low-level storage must travel across a slow I/O interface (e.g., SATA), main memory and multi-level caches before reaching the compute units of CPU or GPUs [24], which is depicted in Fig. 1(a).

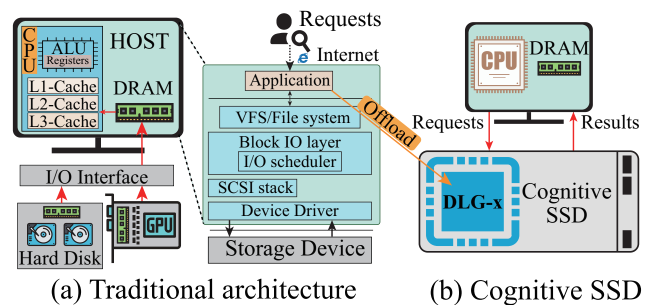

Fig. 1. Traditional architecture (a) vs. Cognitive SSD (b).

To address these issues, as shown in Fig. 1(b), this work aims to tailor a unified data storing and retrieval system within the compact storage device, and eliminate the major IO and data moving bottleneck. In this system, retrieval requests are directly sent to the storage devices, and the target data analysis and indexing are completely performed where the unstruc-

[1]Corresponding authors are Ying Wang and Huawei Li.

tured data resides. Building such a data retrieval system based on the proposed Cognitive SSD bears the following design goals: (1) providing a high accuracy, low latency, and energy efficient query mechanism affordable in a compact SSD, (2) exploiting the internal bandwidth of flash devices in an SSD for energy-efficient deep learning based data processing, and (3) enabling developers to customize the data retrieval system for different dataset. These points are stated in detail below.

First, instead of relying on the general-purpose CPU or GPU devices in Fig. 1(a), we must have a highly computation-efficient yet accurate data retrieval architecture in consideration of the SSD form factor and cost. A conventional data retrieval framework is inaccurate or too computationally expensive to be implemented within a resource-constrained SSD. In this work, we are the first to propose a holistic data retrieval mechanism by combining the deep learning and graph search algorithm (DLG), where the former could extract the semantic features of unstructured data and the latter could improve database search efficiency. The DLG solution achieves much higher data retrieval accuracy and enables user-definable computation complexity through deep learning model customization, making it possible to implement a flexible and efficient end-to-end unstructured data retrieval system in the SSD.

Second, although DLG is a simple and flexible end-to-end data retrieval solution, embedding it into SSDs still takes considerable effort. We designed a specific hardware accelerator that supports deep hashing and graph search simultaneously, DLG-x, to construct the target Cognitive SSD without using power-unsustainable CPU or GPU solutions. However, the limited DRAM inside an SSD is mostly used to cache the metadata for flash management, leaving no free space for the deep learning applications. Fortunately, we have proved that the bandwidth of internal flash interface surpasses that of external IO interface in a typical SSD, which matches the bandwidth demand of the DLG-x with proper data layout mapping. By rebuilding the data path in the SSD and deliberately optimizing the data-layout related to deep learning models and graphs on NAND flash, the DLG-x could fully exploit internal parallelism and directly access data from NAND flash bypassing the on-board DRAM.

Finally, as we introduce deep learning technology into the SSD, we must expose the software abstraction of Cognitive SSD to users and developers to process different data structures with different deep learning models. Thus, we abstract the underlying deep learning mechanism, feature analysis, and data structure indexing mechanism as user-visible calls by utilizing the NVMe protocol [6] for command extension. Not only can users' requests trigger the DLG-x accelerator to search the target dataset for query-relevant structures, but also system developers can freely configure the deep hashing architecture with different representation power and overhead for different dataset and performance requirement. In contrast to conventional ad-hoc solutions, Cognitive SSD allows system developers to adjust the retrieval accuracy as well as the real-time performance of the data retrieval service through provided APIs. Meanwhile, Cognitive SSD also supports the flexible combination of special commands to achieve different data retrieval related tasks, like in-storage data categorization and hashing-only functions. In summary, we make the following novel contributions:

1 We propose Cognitive SSD, to enable within-SSD deep learning and graph search by integrating a specialized deep learning and graph search accelerator (DLG-x). The DLG-x directly accesses data from NAND flash without crossing multiple memory hierarchies to decrease data movement path and power consumption. To the best of our knowledge, this work is the first to combine the deep learning and graph search methods for fast and accurate data retrieval in SSD.

2 We employ Cognitive SSD to build a serverless data retrieval system, which completely abandons the conventional data query mechanism in orthodox compute-centric systems. It can independently respond to data retrieval requests at real-time speed and low energy cost. It can also scale to a multi-SSD system and significantly reduces the hardware and power overhead of large-scale storage nodes in data centers.

3 We build a prototype of Cognitive SSD on the Cosmos plus OpenSSD platform [7] and use it to implement a data retrieval system. Our evaluation results demonstrate that Cognitive SSD is more energy-efficient than a multimedia retrieval system implemented on CPU and GPU, and reduces latency by 69.9% on average compared to the implementation with CPU. We also show that it outperforms conventional computing and storage node used in the data center when Cognitive SSD scales out to form smart lightweight storage nodes that include connected Cognitive SSD array.

2 Background and Preliminaries

2.1 Unstructured Data Retrieval System

Content-based unstructured data retrieval systems aim to search for certain data entries from the large-scale dataset by analyzing their visual or audio content. Fig. 2 depicts a typical content-based retrieval procedure consists of two main stages: feature extraction, and database indexing. Feature extraction generates the feature vector for the query data, and database indexing searches for similar data structures in storage with that feature vector encoded in a semantic space.

Feature Extraction and Deep Learning. The rise of deep learning transfers the focus of researches to deep convolution neural network (DCNN) [38] based features [61], as it provides better mid-level representations [37,40]. Fig. 2 depicts a typical DCNN that contains four key types of network layers: (1) convolution layer, which extracts visual feature from input by moving and convolving multidimensional filters across the input data organized into 3D tensors, (2) activation, the

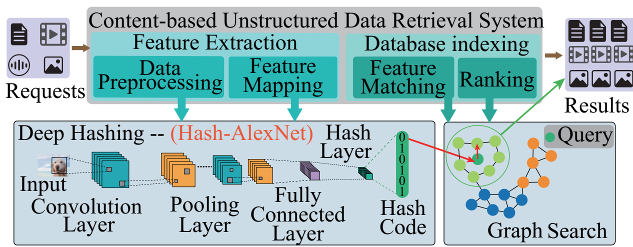

Fig. 2. Content-based multimedia data retrieval system.

nonlinear transformation that we do over the input signal, (3) pooling layers, which down-sample the input channels for scale and other types of invariance, and (4) fully connected (FC) layer, which performs linear operations between the features and the learned weights to predict the categorization or other high level characteristics of input data. Such a neural network is flexible and can be designed to have different hyper-parameters, like the number of the convolution and pooling layers stacked together and the dimension and number of convolution filters. Changing these parameters will impact the generalization ability and also computational overhead of neural networks, which are usually customizable for different dataset or application scenarios [47]. Some prior work directly employs the high-dimension output vector of the FC layer for data retrieval and is thought too expensive in terms of memory footprint and computation complexity [39]. Thus, we adopt deep hashing [38] to achieve effective yet condensed data representation. Fig. 2 exemplifies a deep hashing architecture, Hash-AlexNet, where a hash layer follows the last layer of AlexNet [35] to project the data feature learned from AlexNet into the hash space, and the generated hash code can be directly used to index the relevant data structures and get rid of the complex data preprocessing stage.

Database indexing: Graph-based approximate nearest neighbor search(ANNS) methods named NSG [27], a complement to deep hashing, achieves both accurate and fast data retrieval results, as was proved in previous work [17, 26]. The main idea of NSG is mapping the query hash code into a graph. The vertex of the graph represents an instance, and the edge stands for the similarities between entities, where the value of the edge represents the strength of similarity. On top of that, NSG can iteratively check neighbors' neighbors in the graph to find the true neighbors of the query based on the neighbor of a neighbor is also likely to be a neighbor concept. In this manner, the NSG could avoid unnecessary data checking to reduce retrieval latency.

In summary, deep hashing followed by graph search can perform low-latency and high-precision retrieval performance compared to traditional solutions using brute-force search or hash algorithms. Meanwhile, it also makes the retrieval framework more compact and efficient because of the similar compute patterns and data stream, so that they can fit into compact and power-limited SSDs.

2.2 Near data processing & deep learning accelerator

For hardware-software co-design, there are two directions in SSD research: open-channel SSD and near-data processing (NDP). While open-channel SSD enables direct flash memory access via system software [15, 42, 44, 59], near-data processing (NDP) moves computation from the system's main processors into memory or storage devices [16, 18, 25, 46, 50, 51, 56].

In NDP, Morpheus [52] provides a framework for moving computation to the general-purpose embedded processors on NVMe SSD. FAWN [13] uses low-power processors and flash to handle data processing and focuses on a key-value storage system. SmartSSD [34] introduces the Smart SSD model, which pairs in-device processing with a powerful host system capable of handling data-oriented tasks without modifying the operating system. [53] supports fundamental database operations including sort, scan, and list intersection by utilizing Samsung SmartSSD. [23] investigates by simulation the possibility of employing the embedded ARM processor in SSDs to run SGDs, which is a key components of neural network training. However, none of them can handle deep learning processing due to the performance limit of the embedded processor. Thereby, [22] presents intelligent solid-state drives (iSSDs) that embed stream processors into the flash memory controllers to handle linear regression and k-means workloads. [43] integrates programmable logics into SSDs to achieve energy-efficiency computation for web-scale data analysis. Meanwhile, [32] also uses FPGAs to construct BlueDBM that uses flash storage and in-store processing for cost-effective analytics of large datasets, such as graph traversal and string search. GraFBoost [33] focuses on the acceleration of graph algorithms on an in-flash computing platform instead of deep learning algorithms as this work.

Cognitive SSD Prior active disks are integrating either general purpose processors incapable of handling high-throughput data or specialized accelerators with only the support of simple functions like scanning and sorting. These in-disk computation engines are unable to fulfill the requirement of high-throughput deep neural network (DNN) inference because computation-intensive DNNs generally rely on power-consuming CPU or GPUs in the case of data analysis and query tasks. To enable energy-efficient DNN, prior work proposes a variety of energy-efficient deep learning accelerators. For example, Diannao and C-Brain map large DNNs onto a vectorized processing array and employ a data tiling policy to exploit locality in neural parameters [20, 49]. Eyeriss applies the classic systolic array architecture to the inference of CNN, and outperforms CPU and GPU in energy efficiency dramatically [21]. However, these researches focus on optimizing the internal structure of accelerator and relied on large-capacity SRAM or DRAM instead of external non-volatile memory. In contrast to these works and prior active SSD designs, we propose **Cognitive SSD**, the first work that enables the storage device to employ deep learning to conduct

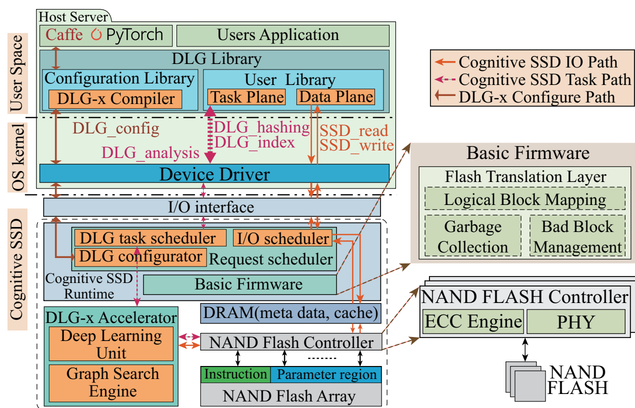

Fig. 3. Overview of the Cognitive SSD system.

in-storage data query and analysis. It is designed to replace the conventional data retrieval system and contains a flash-accessing accelerator (DLG-x) for deep learning and graph search. The DLG-x is deliberately reshaped to take advantage of the large flash capacity and high internal bandwidth, and it is also re-architected to enable graph search to target indexing.

3 Cognitive SSD System

Target Workload. As shown in Fig. 2, this work combines the strengths of deep hashing and graph search technique (DLG) to reduce the complexity of retrieval systems on the premise of high accuracy, which makes it possible to offload retrieval task from CPU/GPU into the resource-constraint Cognitive SSD. Based on that, we build an end-to-end data retrieval system that supports multimedia data retrieval such as audio, video and text. For example, audio can be processed by recurrent or convolutional neural network models on Cognitive SSD to generate hash codes, which act as an index for retrieving relevant audio data inside the SSD. In this paper, image retrieval is used as a showcase. As shown in Fig. 3, Cognitive SSD is designed to support the major components in the framework of DLG, allowing developers to customize and implement data retrieval solutions. Such a near-data retrieval system consists of two major components: the DLG library running in lightweight server that manages user requests, and the Cognitive SSD is plugged into the host server via the PCIe interface. As shown in Table 1, as the interface of Cognitive SSD system, the DLG library is established by leveraging the *Vendor Specific Commands* in the I/O command set of the *NVMe* protocol. It contains a *configuration library* and a *user library*. The configuration library enables the administrator to choose and deploy different deep learning models on the Cognitive SSD quickly according to application demand. After the feature-extracting deep hashing model has been deployed on the Cognitive SSD, a data processing request arriving at the host server could send and establish a query session to it by invoking the APIs provided by the user library. Then, the runtime system on the embedded processor of the Cognitive SSD receives and parses the request to activate the corresponding DLG-x module, which is associated with the user-created session. Next, we elaborate on the software and hardware design details of Cognitive SSD.

3.1 The Cognitive SSD Software: DLG Library

3.1.1 Configuration Library

Update Deep Learning Models: Because the choice of deep learning models significantly impacts the data retrieval system performance, the system administrator must be able to customize a specific deep hashing model according to the complexity and volume of database, and the quality of service measured by response latency or request throughput. Thereby, the configuration library provides a DLG-x compiler compatible with popular deep learning frameworks (i.e., Caffe) to allow the administrator to train the new deep learning model and generate corresponding DLG-x instructions offline. Then, the administrator can update the learning model running on the Cognitive SSD by updating the DLG-x instructions. The updated instructions are sent to the instruction area allocated in the NAND flash and stay there until a model change command (*DLG_config* in Fig. 3) is issued. Meanwhile, the DLG-x compiler also reorganizes the data layout of the DLG algorithm to fully utilize the internal flash bandwidth according to the structures of neural network model and graph, before the parameters of deep learning model and graphs are written to the NAND flash. The physical address of weight and graph structure information is recorded in the DLG-x instruction. In this manner, the DLG-x obtains the physical address of required data directly at runtime, instead of adopting the address translation or look-up operations that incur additional overhead. More details about the data reorganization scheme are introduced in § 4.

3.1.2 User Library

Data Plane: The data plane provides *SSD_read* and *SSD_write* APIs for users to control data transmission between the host server and the Cognitive SSD. These two commands operate directly on the physical address bypassing the flash translation layer. Users can invoke these APIs to inject data sent from users to the data cache region or the NAND flash on the Cognitive SSD based on the parameter of data address and data size. Afterwards, users can use those addresses to direct the operands in other APIs.

Task Plane: To improve the scalability of the DLG-x accelerator that supports deep hashing neural networks and graph search algorithms, we abstract the function of the DLG-x into three APIs in the task plane of user library: *DLG_hashing*, *DLG_index*, and *DLG_analysis*. These APIs are established using the ***C0h***, ***C1h***, and ***C2h*** commands of NVMe I/O protocol, respectively. All of them possess two basic parameters carried by NVMe protocol ***DWord***s: the *data address* indicating the data location in Cognitive SSD, and the *data size* in bytes.

First, the *DLG_hashing* API is designed to extract the condensed feature of input data and map it into the hash or seman-

Table 1: DLG Library APIs for Cognitive SSD

	-	API	NVMe command	DWord10	DWord11	DWord12	Description
Configuration Library	-	*DLG_config*	0xC3	address	size	Instruction/model	Update instruction and model on DLG-x
User Library	Task Plane	*DLG_hashing*	0xC0	data address	data size	Hashcode length	Extract the hashing feature of input data
		DLG_index	0xC1	data address	data size	T	Fast database indexing
		DLG_analysis	0xC2	data address	data size	User-defined	Analysis of input data
	Data Plane	*SSD_read*	0xC4	data address	data size	–	Physical Address Read
		SSD_write	0xC5	data address	data size	–	Physical Address Write

tic space, which is fundamental in a data retrieval system and useful for other analysis functions like image classification or categorization. This command contains an extended parameter: *hashcode length*, which determines the capacity of the carried information. For example, compared to the database with 500 objects types, the database with 1000 objects needs a longer hash code to avoid information loss. Second, the *DLG_index* API is abstracted from the graph search function of the DLG-x. It also includes an extended parameters: *T*, represents the number of search results configured by users based on the applications scenarios. Finally, the *DLG_analysis* API allows users to analyze the input data using the data analysis and processing ability of deep neural networks and it also possesses a reserved field for user-defined functions. These task APIs are the abstraction of the key near-data processing kernels provided by Cognitive SSD, and they can be invoked independently or combinedly to develop different in-SSD data processing functions. For instance, users could combine the *DLG_hashing* and *DLG_index* APIs to accomplish data retrieval on a large-scale database, where *DLG_hashing* maps the features of query data to a hash code and *DLG_index* uses it to search for the top-*T* similar instances.

3.1.3 Cognitive SSD Runtime The Cognitive SSD runtime deployed on the embedded processor inside the Cognitive SSD is responsible for managing the incoming extended I/O command via PCIe interface. It also converts the API-related commands into machine instructions for the DLG-x accelerator, as well as handles basic operations for NAND flash. It includes a request scheduler and the basic firmware. The request scheduler contains three modules: the ***DLG task scheduler***, the ***I/O scheduler***, and the ***DLG configurator***. The *DLG configurator* receives *DLG_config* commands from the host and updates the instructions generated by the compiler and parameters of the specified deep learning model for Cognitive SSD. The *DLG task scheduler* responds to users requests as supported in the task plane and initiates the corresponding task session in Cognitive SSD. The I/O scheduler dispatches I/O requests to the basic firmware or the DLG-x. The basic firmware includes the flash translation layer, for logical block mapping, garbage collection, bad block management functions, and communicates with the NAND flash controller for general I/O requests.

Note that the DLG-x accelerator occupies a noticeable portion of the flash bandwidth once activated, which perhaps degrades the performance of normal I/O requests. To alleviate this problem, instead of letting the task or I/O scheduler wait until the request is completed (denoted as Method A), the *DLG task scheduler* receives the NVMe command sent from host with doorbell mechanism and actively polls the completion status of the DLG-x periodically (denoted as Method B) to decide if the next request is dispatchable. We tested the normal read/write bandwidth of Cognitive SSD prototype described in § 5.1 with the Flexible IO Tester (fio) benchmark [4], under the worst-case influence where the DLG-x accelerator operations occupied all the Cognitive SSD channels. Experiments (Table 2) demonstrate that adopting Method B only causes a drop of 27%-44% in the normal I/O bandwidth whilst using Method A decreases almost 91% of the read/write bandwidth averagely when the DLG-x accelerator is busy dealing with the over-committed retrieval tasks.

Table 2: The I/O Bandwidth of Cognitive SSD.

-	I/O Bandwidth (MB/s) (I/O size = 128KB)			
-	Write	Random Write	Read	Random Read
Method-A	79.79	76.19	72.30	81.13
Method-B	524.86	421.23	654.31	698.98
Peak-Bandwidth	886.58	761.90	903.79	901.41

3.2 Hardware Architecture: Cognitive SSD

Fig. 3 depicts the hardware architecture of Cognitive SSD. It is composed of an embedded processor running the Cognitive SSD runtime, a DLG-x accelerator and NAND flash controllers connected to flash chips. Each NAND flash controller connects one channel of NAND flash module and uses an ECC engine for error correction. When the devices in each channel operate in lock-step and are accessed in parallel, the internal bandwidth surpasses the I/O interface. More importantly, though SSDs often have compact DRAM to cache data or metadata, the internal DRAM capacity can hardly satisfy the demand of the deep learning, which is notorious for its numerous neural network parameters. Worse still, the basic firmware like FTL and other components also occupy major memory resources. Therefore, the NAND flash controller is exposed to the DLG-x accelerator, which enables the DLG-x to read and write the related working data directly from NAND flash, bypassing the internal DRAM.

3.3 The Procedure of data retrieval in Cognitive SSD

Fig. 3 also depicts the overall process of Cognitive SSD when users perform unstructured data retrieval task. First, assume that the hardware instruction and parameters of Hash-AlexNet

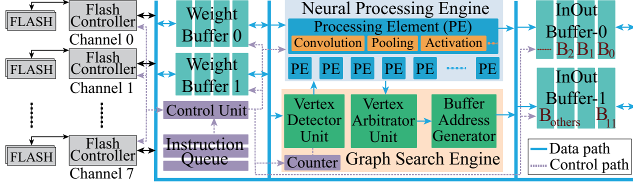

Fig. 4. The Architecture of DLG-x accelerator.

model have been generated and written to the corresponding region by leveraging the DLG-x compiler and the *DLG_config* command shown in Fig. 3. The Hash-AlexNet is the developer designated neural network for feature extraction of input data. Then, when the host DLG library captures a retrieval request, it packages and writes the user input data from the designated host memory space to Cognitive SSD through the *SSD_write* API. Meanwhile, the *DLG_hashing* command carrying the address of input data is sent to Cognitive SSD for hash code generation. Receiving the command, the request scheduler of the cognitive runtime parses it and notifies the DLG-x accelerator to start a hashing feature extraction session. Then, the DLG-x automatically fetches input query data from the command-specified data address and then loads deep learning parameters from NAND flash. Meanwhile, the other command, *DLG_index*, is sent and queued by the task scheduler. After the hash code is produced, the *DLG_index* is dispatched to invoke the graph search function in the DLG-x and uses the hash result to search the data graphs for relevant data entries. In this case, the DLG-x keeps fetching graph data from the NAND flash and sends the final retrieval results to the host memory once the task is finished.

4 DLG-x Accelerator

4.1 Architecture: Direct Flash Accessing

In contrast to a traditional hardware accelerator [20], the DLG-x accelerator is designed to directly obtain the majority of the working-set data from NAND flash. Fig. 4 illustrates the high-level diagram. The DLG-x accelerator has two activation buffers (InOut Buffer) and double-banked weight buffers. The intermediate results of each neural network layers are temporarily stored in the activation buffers, while the weight buffers act as a bridge buffer between the Neural Processing Engine (NPE) and the flash, which stream out the large quantity of neural parameters to the NPE. The NPE comprises a set of processing engines (PEs), which can perform fixed-point convolutions, pooling, and activation function operations. The Graph Search Engine (GSE) cooperates with the NPE and is responsible for graph search with the hash code generated by NPE. Both the NPE and GSE are managed by the control unit that fetches instructions from memory. Considering the I/O operation granularity of NAND flash, we reorganize the data layout of neural networks including both the static parameters and the intermediate feature data, to exploit the high internal flash bandwidth.

4.2 I/O Path in Cognitive SSD

Bandwidth Analysis: At first, we analyze and prove that the internal bandwidth of NAND flash can satisfy the demand of deep neural network running on the DLG-x accelerator. Assuming that the DLG-x and flash controller runs on the same frequency and the NPE unit of the DLG-x comprises N_{PE} PEs. The single channel bandwidth of NAND flash is BW_{flash}. Thereby, the bandwidth of M channels equals to:

$$BW_{flash}^{m} = M \times BW_{flash} \tag{1}$$

Suppose that a convolution layer convolves a $I_c \times I_h \times I_w$ input feature map (IF) with a $K_c \times K_h \times K_w$ convolution kernel to produce a $O_c \times O_h \times O_w$ output feature map (OF). The subscript c, h, and w correspond to the channel, height, and width respectively. The input/weight data uses an 8-bit fixed-point representation. It is easy to derive that the computation latency $L_{compute}$ and the data access latency L_{data} from NAND flash to produce one channel of feature map are:

$$L_{compute} = \frac{OP_{compute}}{OP_{PE}} = \frac{2 \times K_c \times K_h \times K_w \times O_h \times O_w}{2 \times N_{PE}} \tag{2}$$

$$L_{data} = \frac{S_{param}}{BW_{flash}^{m}} = \frac{K_c \times K_h \times K_w}{BW_{flash}^{m}} \tag{3}$$

Where the $OP_{compute}$ and S_{param} is the operation number and the parameters volume of a convolutional layer. OP_{PE} gauges the performance of the DLG-x measured in operations/cycle. To avoid NPE stalls,we must have $L_{compute} >= L_{data}$, and O_w is usually equal to O_h, so we have

$$O_w >= \sqrt{N_{PE}/BW_{flash}^{m}} \tag{4}$$

The above equation indicates that if only the width and height of the output feature map is larger than or equal to the right side of formula 4, which is four in our prototype with $N_{PE} = 256$ and $BW_{flash}^{m} = 16 bytes/cycle$, the NPE will not stall. For example, in the Hash-AlexNet mentioned in § 2.1, the minimum width of the output feature map in convolution layers is 7, which already satisfies in inequality 4 design. However, in the FC layers, $L_{compute}$ is smaller than L_{data}, so the data transfer time becomes the bottleneck. Thereby, the DLG-x accelerator only uses a column of PEs to deal with a FC layer because our prototype hardware design only supports eight channels, which does not meet inequality 4 with $M = 128$ and consequently causes PE underutilization. Besides, the parameter-induced flash reads will be minimized if the size of the weight buffer meets the condition: $S_{buffer} >= Max(K_c \times K_h \times K_w)$. The parameters exceeding the size of weight buffer will be repetitively fetched from the flash. To further improve the performance, we utilize ping-pong weight buffers to overlap the data loading latency with computation.

Data Layout in flash devices: Owing to the bandwidth analysis on the base of multi-channels data transmission, we propose flash-aware data layout to fully exploit flash bandwidth with the advanced NAND flash command-*read page cache*

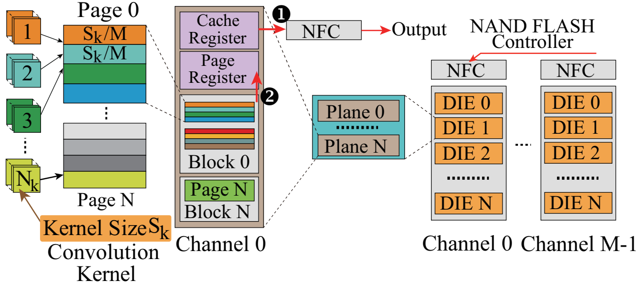

Fig. 5. The Data Layout in NAND flash.

command [11]. The read page cache sequential command provided by NAND flash manufacturer can continuously load the next page within a block into the data register (❷) while the previous page is being read to the buffer of the DLG-x or the cache region of the SSD from the cache register(❶). Thus, based on the NAND flash architecture with the provided page cache command, we choose to split the convolution kernels and store them into flash devices for parallel fetch. As shown in Fig. 5, assuming there are N_k convolution kernels with S_k kernel size, and M NAND flash channels are used by the DLG-x accelerator, each convolution kernel is divided into S_K/M sub-blocks and all such sub-blocks are interleaved to the flash channels. The convolution kernels exceeding the size of a page are placed into continuous address space in the NAND flash because the cache command reads out the next page automatically without any extra address or operation.

Data Flow: Taking the Hash-AlexNet as an example, when a request arrives at the DLG-x accelerator, the input data and the first kernel of the first convolution layer is transferred in parallel to the InOut buffer-0 and the weight buffer-0. After that, the DLG-x accelerator begins to compute the output feature map and stores them into the InOut buffer-1. When the first kernel is processed, the second kernel is being transferred from the NAND flash to weight buffer-1, then followed by the third and fourth kernel in sequence. Once the hash code is generated, it is sent to the graph search registers of NPE to locate the data structures similar to the query data if the *DLG task scheduler* decodes and dispatches a following *DLG_index* command.

4.3 Fusing Deep Learning and Graph Search

For fast and accurate database indexing, the DLG-x accelerator fuses the deep learning and graph search mechanism into unified hardware, and reuses the computation and memory resources for higher efficiency. Once the hash code of the query data has been generated, the DLG-x uses it to initially index the corresponding data graphs and searches for the closest data entries from graphs.

The graph search method originates from Navigating Spreading-out Graph [27] (NSG), which well fits the limited memory space of the Cognitive SSD for the large-scale multimedia data retrieval. The NSG algorithm includes an offline stage and an online stage. In the offline phase, the NSG method constructs a directed $K_{nbors}-NN$ graph for the storage data structures to be retrieved. In a graph, a vertex represents a data entry by keeping its ID and hash code. The unique ID represents a file and the hash code is the feature vector of this file, which could be obtained by invoking the *DLG_hashing* API in advance. The bit-width of ID (W_{id}) and hash code (W_{hash_code}) are user-configurable parameters in the API. In a graph, a vertex may be connected to many vertices, which have different distances from each other. However, only the top-K_{nbors} closest vertices of a vertex could be defined as its "neighbors", where K_{nbors} is also a reconfigurable parameter and enables users to pursue the trade-off between accuracy and retrieval speed. The DLG-x accelerator only accelerates the online retrieval stage and the database update occurs offline because the latter task is infrequent. The database update consists of hash code extraction stage and $K_{nbors}-NN$ graph construction stage, where the former is accelerated by the DLG-x accelerator and the latter is completed with the DLG library on CPUs. At offline graph construction, it takes about 10∼100 seconds to update the $K_{nbors}-NN$ graph on million-scale data on a server CPU. The hardware architecture for online graph search is presented in Fig. 4.

The graph search function of the DLG-x starts from evaluating the distance of random initial vertices in the graph and walks the whole graph from vertex to vertex in the neighborhood to find the closest results. As shown in Fig. 4, to maximize the utilization of on-chip memory, the weight buffer and InOut buffer are reused to store the neighbors of vertices and the search results of the graph search engine respectively. Since the Hamming distance (H-distance) is an integer value, the InOut buffer is divided into blocks according to the range of H-distance. For instance, the first block of the InOut buffer B_0 only stores the vertices with zero Hamming distance away from the query vertex, and the second block B_1 corresponds to the distance of one hop. The last area B_{others} stores the vertices with Hamming distance larger than the final value V_{final}, where the V_{final} is a re-definable parameter and calculated with formula 5.

$$V_{final} = \left\lfloor \frac{S_{buffer}}{D_{block} \times W_{id}} - 1 \right\rfloor \quad (5)$$

In the above equation, S_{buffer} is the on-chip buffer size of the DLG-x accelerator and D_{block} represents the number of vertex IDs that can be stored in each block. W_{id} is usually equal to 32bits. For instance, in our design, with $S_{buffer} = 256KB$ and $D_{block} = 5000$, it is easy to have $V_{final} = 12$. Note that the limited size of the region B_{others} cannot hold all the distant data vertices generated at runtime, and thus B_{others} is configured to a ring buffer to accommodate the incoming vertices cyclically.

A Vertex Detector Unit (VDU) is inserted to check whether the selected vertex has been evaluated. In VDU, the vertex will be discarded once found to have been walked before, otherwise it will be sent to the NPE unit to compute the Hamming distance from the query vertex. With the distance

provided by the NPE, the Buffer Address Generator (BAG) module allocates memory space in the InOut buffer for the vertex and then puts the vertex into the assigned areas of the InOut buffer. The unevaluated vertices will be fetched from the InOut buffer and the neighbors of these vertices are loaded from the weight buffer by the control unit. Meanwhile, the control unit will finally return the top-T closest vertices when the number of vertex in the InOut buffer reaches the threshold configured by users, where T is also configured by users via the *DLG_index* API.

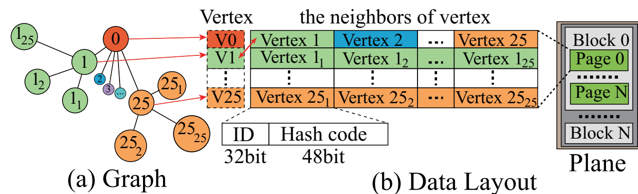

Fig. 6. The data organization on one page.

Data Layout for fast In-SSD NSG search: NAND flash read operations are performed at a page granularity (16KB), so that every time the DLG-x accesses the neighborhoods of one vertex (250bytes), it must read one whole page from flash, which perhaps causes low bandwidth utilization if locality is not well preserved. In our design that $K_{nbors} = 25$, $W_{id} = 32bits$ and $W_{hash_code} = 48bits$. Inspired by the intuition that the neighbor of a neighbor is also likely to be a neighbor of the query data in the graph, we can infer that the neighbors of the accessed vertex will be used soon due to the spatial locality. Therefore, as shown in Fig. 6, V_0 and all its neighbors $(V_1, V_2, ..., V_{25})$, are continuously aligned and stored from the beginning of a page. As a result, such a layout with redundancy is able to reduce flash access by 37x compared to a non-optimized graph layout. However, such a layout cause duplicates of vertices in storage and sacrifice additional storage space for better data access performance, which is worthwhile regarding the large capacity of SSDs.

Besides data layout transformation, the bit-width of the hash code is also worth elaborating. Due to the limited page size S_{page}, W_{hash_code} and K_{nbors} must conform to the resource constraint given by:

$$K_{nbors}^2 \times (W_{hash_code} + W_{id}) < S_{page} \qquad (6)$$

Generally $S_{page} = 16KB$, and W_{id} is 32-bit wide and can represent 2^{32} files. Because the parameters W_{hash_code} and K_{nbors} directly impact the deep hashing performance by influencing the indexing accuracy and also the memory bandwidth consumption during graph search, once S_{page} is determined, W_{hash_code} and K_{nbors} must be adjusted to reach a perfect balance between accuracy and retrieval time at the offline stage. Thus, the DLG-x must support different parameter formats in order to achieve best-effort computing efficiency for databases of different volume and complexity.

Note that our graph layout and the according searching strategy are adapted to the underlying hardware for higher energy efficiency, and they will lead to a marginal amount of query accuracy losses compared to the original algorithm. Table 3 indicates the accuracy loss compared with the original lossless DLG algorithm (denoted as Original) on the CIFAR-10 [9] and ImageNet [45] datasets. The result shows that when *T=1000*, the accuracy drops by 2.26% and 4.36%, as a side-effect of the ~ 37x performance boost. Fortunately, the DLG library APIs are flexible enough to allow the developers to trade-off between accuracy and performance by manipulating the API arguments.

Table 3: The accuracy loss.

Dataset		Accuracy(%) at T samples				
		200	400	600	800	1000
CIFAR-10	Original	86.65	86.58	86.56	86.51	86.54
	Our	85.49	85.02	84.75	84.47	84.28
	Loss	1.16	1.56	1.81	2.04	2.26
ImageNet	Original	33.78	33.77	33.48	32.89	31.83
	Our	30.66	29.79	29.13	28.43	27.47
	Loss	3.12	3.98	4.35	4.45	4.36

Data Flow: we show an example to brief the overall flow of the DLG-x based data retrieval. Firstly, when a query comes, the DLG-x fetches the input data and parameters of the deep learning model from the NAND flash into the InOut buffer and the weight buffer of the DLG-x respectively. Then, the NPE unit generates the hash code for the input data and writes it to the graph search registers of the NPE unit. After that, the DLG-x transfers the $K_{nbors} - NN$ graph from the NAND flash array to the weight buffer. At the first stage, the initial vertices are calculated and sent into the corresponding areas of the InOut buffer. At the second stage, the DLG-x control unit reads the first unevaluated vertex from the InOut buffer in ascending order of Hamming distance. Then, the graph search engine obtains the neighbors of the unevaluated vertex from the NAND flash and transfers the neighbors to NPE to generate their Hamming distances from the query vertex as well. Next, the Buffer Address Generator unit generates the write addresses for these neighbor vertices in the InOut buffers according to the calculated Hamming distance and writes these vertices to the InOut buffers. Meanwhile, the counter in the graph search engine determines whether the termination signal should be issued by monitoring the total number of vertices stored in the InOut buffer. Once the termination signal is generated, the Cognitive SSD runtime reads out the vertices from the InOut buffer and then transfers the ID-directed results stored in NAND flash to the host server via the PCIe interface.

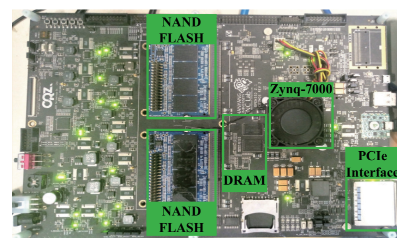

Fig. 7. Cognitive SSD prototype.

5 Evaluation

5.1 Hardware Implementation

To explore the advantages of the Cognitive SSD system, we implemented it on the Cosmos plus OpenSSD platform [7]. The Cosmos plus OpenSSD platform consists of an XC7Z045 FPGA chip, 1GB DRAM, an 8-way NAND flash interface, an Ethernet interface, and a PCIe Gen2 8-lane interface. A DLG-x accelerator is designed with DeepBurning [55] and integrated to the modified NAND flash controllers, and they are all implemented on the programmable logic of XC7Z045. The Cognitive SSD runs its firmware on a Dual 1GHz ARM Cortex-A9 core of XC7Z045. The Cognitive SSD is plugged into the host server via a PCIe link. The host server manages the high-level requests and maintains the DLG library for API calls. Fig. 7 shows the Cognitive SSD prototype constructed for this work.

5.2 Experimental Setup

We first selected the content-based image retrieval system (CBIR) based on deep hashing and graph search (DLG) algorithm as workload and evaluated the performance of DLG solution compared to other conventional solution (§5.3). we evaluated the DLG-x of the Cognitive SSD prototype in §5.4, and deployed the Cognitive SSD prototype to a single node and multi-node system, and evaluated them in §5.5, and §5.6, respectively. Except for the Cognitive SSD prototype, our experimental setup also consists of a baseline server running Ubuntu 14.04 with two Intel Xeon E5-2630 CPU@2.20GHz, 32GB DRAM memory, four 1TB PCIe SSDs and an NVIDIA GTX 1080Ti. Meanwhile, we implemented the CBIR system in C++ on the baseline server, where the deep hashing is built on top of Caffe [31]. Based on this platform, we constructed four solutions baselines: B-CPU, B-GPU, B-FPGA, and B-DLG-x. For B-CPU, the DLG algorithm runs on the CPU. For B-GPU, the deep hashing runs on the GPU and graph search runs on the CPU. For B-FPGA, we use ZC706 FPGA board [12] to replace Cognitive SSD, and the deep hashing runs on ZC706 FPGA board and graph search runs on the CPU. B-DLG-x implements the DLG algorithm on ZC706 FPGA board without any near-data processing technique compared to Cognitive SSD.

5.3 Evaluation of DLG algorithm

Experimental Setup. We used the precision at top T returned samples ($Precision@T$), measuring the proportionality of corrected retrieved data entries, to verify the performance of our deep hashing method on different models and datasets [36]. The performance is contrasted with traditional hash methods with 512-dimensional GIST feature, including Locality-Sensitive Hashing (LSH) [54] and Iterative Quantization (ITQ) [28]. The used datasets are listed in Table 4.

Evaluation. Fig. 8(a)-(d) shows the $Precision@T$ on different datasets with different deep hashing models. Due to the poor performance of LSH and ITQ [28] methods on ImageNet, we added the AlexNet-ITQ (A+ITQ) and AlexNet-CCA-ITQ (AC+ITQ) methods [58] that uses the output of the FC layer on Alexnet as the feature vector for search. Our DLG solution performs better than the other approaches on different datasets with different scales regardless of the choice of T value, especially compared to the conventional hash method. It also shows the robustness of the DLG solution when deploying on a real-world system. Meanwhile, Fig. 8(d) shows the performance of our approach is significantly improved when the code length increases to 64 bits. Thereby, the DLG-x accelerator is configured to support different hash code length to achieve the trade-off between retrieval accuracy and latency.

Table 4: Datasets used in our experiments

Dataset	Total	Train/Validate	Labels
CIFAR-10 [9]	60000	50000/10000	10
Caltech256 [29]	29780	26790/2990	256
SUN397 [57]	108754	98049/10705	397
ImageNet [45]	1331167	1281167/50000	1000

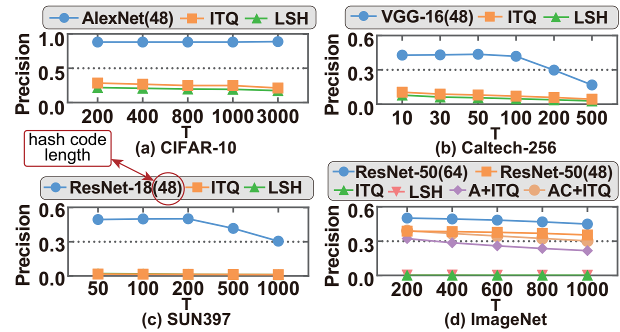

Fig. 8. Precision curves w.r.t top-T.

5.4 Evaluation of DLG-x

Experimental Setup. We implemented the deep hashing and graph search algorithm (DLG) on the DLG-x accelerator of Cognitive SSD and compared the latency and power of the DLG-x to the solutions based on CPU and GPU, where we ignore the FPGA baseline because its computational units are the same as the DLG-x. Firstly, we only compared the latency and power of the deep hashing unit on the DLG-x running various deep hashing models to CPU and GPU, where GPU only reports the total power consumed by NVIDIA GTX 1080Ti without the power of the CPU. Secondly, we only evaluated the latency of graph search function on the DLG-x with respect to different number (T) of top retrieved data entries on the CIFAR-10 and ImageNet dataset.

Performance. Firstly, Table 5 shows the latency of the DLG-x on various deep hashing schemes outperforms the solution based on CPU. While the latency of the DLG-x is higher than GPU because of the hardware resource and frequency limitation, it consumes less power compared to GPU. More importantly, the latency of the CPU and GPU on Table 5 only contains the computation delay of neural network without the delay of parameters transmission between storage and

Table 5: Deep hashing performance on different platforms.

Model	-	Latency (ms)	Power (Watt)
Hash-AlexNet	DLG-x	**38**	**9.1**
	CPU	114	186
	GPU	1.83	164
Hash-ResNet-18	DLG-x	**94**	**9.4**
	CPU	121	185
	GPU	7.13	112

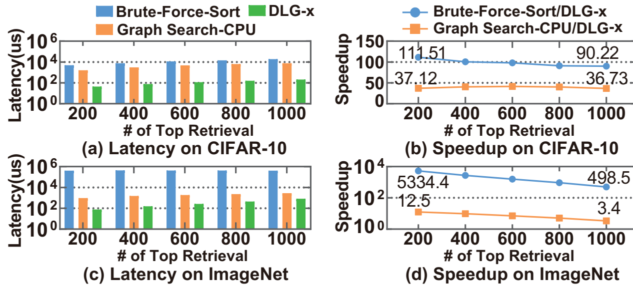

Fig. 9. Graph search performance of the DLG-x.

memory. When considering the delay of parameters transmission between storage and memory, the deep hashing occupies about 87.7% and 73.9% of the total processing time on the DLG-x accelerator and CPU baseline, on average, respectively. And the GPU only accounts for 3.5% of the total runtime on average because the high-speed data processing capability of GPU makes the data transmission becomes the bottleneck of system. Besides, the latency of deep hashing on the GPU occupies 54.17∼26.06% without considering the delay of data movement because the latency of graph search increases with the increase of T value.

Secondly, in this experiment, we utilized the Hash-AlexNet model to generate the hash code database for the construction of a $K_{nbors}-NN$ graph on the CIFAR-10 and ImageNet dataset. We compared the retrieval speed of the DLG-x accelerator with two counterparts: the brute-force search method that evaluates all the hash codes stored in the database, and the CPU executed graph search algorithm. The result is depicted in Fig. 9. For the CIFAR-10 dataset, the DLG-x accelerator is 111.51-90.22x and 37.12-36.73x faster than the brute-force method and the CPU-run graph search algorithms respectively, while for the ImageNet dataset it achieves a 5334.4-498.5x and 12.5-3.4x speed up over the latter two baselines. As introduced in § 4.3, the retrieval accuracy is only degraded by 2.26% and 4.36% when $T = 1000$ on the CIFAR-10 and ImageNet datasets, respectively.

Power Consumption. We measured and compared the power consumption of Cognitive SSD system with four baselines: B-CPU, B-GPU, B-FPGA, and B-DLG-x by using a power meter under two different situations: (1) IDLE: No retrieval requests need to respond, and (2) ACTIVE: A user continuously accesses the Cognitive SSD system. The result is illustrated in Table 6. When the Cognitive SSD+CPU system is IDLE, its power consumption is slightly higher than B-CPU and B-GPU because the Cognitive SSD prototype board IDLE power is higher than that of the PCIe SSD and GPU. For active power, when delivering comparable data retrieval performance, the Cognitive SSD system reduces the total power consumption by up to 34.4% and 63.0% compared with B-CPU and B-GPU. Simply replacing the GPU with the FPGA board reduces power consumption by 40.7%. Furthermore, putting the DLG-x on an identical FPGA board without the NDP decrease power consumption by 43.72%, which is attributed to the efficiency of hardware specialization. Placing the DLG-x into the Cognitive SSD system further eliminates power consumption by another 19.3%, which is the benefit of near-data processing. In the case of the Cognitive SSD+CPU solution, the power of CPU is low because it is only responsible for instruction dispatch without any data transfer between storage and CPU. In other cases, the CPU is not only in charge of data transfer management but also for instruction dispatch or executing the graph search algorithm.

Table 6: Power consumption.

Power (Watt)	Cognitive SSD	Cognitive SSD+CPU	B-DLG-x	B-FPGA	B-CPU	B-GPU
IDLE	17	98.5	89	89	80	90
ACTIVE	20	122	185.7	195.6	186	330

Table 7: The hardware utilization of Cognitive SSD.

Module	#	LUT	FF	BRAM	DSP
Flash Controller	8	11031	7539	21	0
NVMe Interface	1	8586	11455	28	0
DLG-x Accelerator	1	67774	18144	137	197
`In Total`	1	203099	145078	354	197
Percent(%)	-	92.91	33.18	64.95	21.8

FPGA Resource Utilization. The placement and routing were completed with Vivado 2016.2 [8]. Table 7 shows the hardware utilization of Cognitive SSD. It only reports the resources overhead of the flash controller, NVMe controller, and the DLG-x accelerator module. The item of `In Total` counts in all FPGA resources spent by the Cognitive SSD.

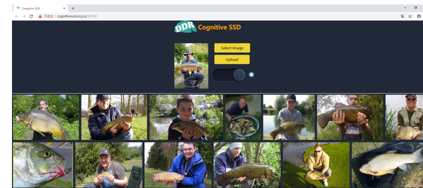
Fig. 10. A CBIR system based on Cognitive SSD.

5.5 The Single-node System Based on Cognitive SSD

Experimental Setup. We implemented the CBIR system by using the ImageNet dataset on the Cognitive SSD with a baseline server, where the baseline server is only responsible for receiving and sending retrieval requests to Cognitive SSD. The deep hashing architecture is a Hash-AlexNet network and the hash code length is 48 bits. As shown in Fig. 10, we built a web-accessible CBIR system based on web framework CROW [30] to evaluate the latency and query per second (QPS) of the system by simulating the user requests sent to

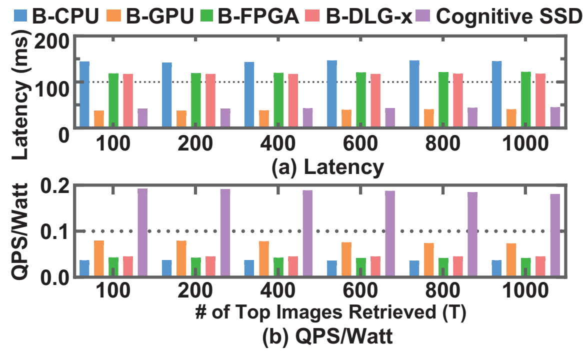

Fig. 11. System performance.

the URL address via ApacheBench (ab) [1]. The latency measurement indicates the time between issuing a request and the arrival of the result. The QPS is a scalability measuring metric characterizing the throughput of the system. The latency and QPS are affected by the software algorithm and the hardware performance of the system. Meanwhile, we utilized the metric of QPS per watt (QPS/Watt) to evaluate the energy-efficiency of the system.

Evaluation. We evaluated the performance of the Cognitive SSD system and four baselines under the assumption that data (weight/graph) cannot be accommodated in DRAM and must travel across the SSD cache, I/O interface, and DRAM before reaching a compute unit. The performance of the Cognitive SSD system and four baselines are shown in Fig. 11. With the increased number of top images retrieved, the retrieval time spent on the DLG-x accelerator will rise. It leads to increased retrieval latency and decreased QPS for the Cognitive SSD. Meanwhile, we also observe the 95% requests complete in time in experiments when write operations and garbage collection are inactive on the Cognitive SSD. Note that write operations and garbage collection are rare for the Cognitive SSD compared to read operations and usually occur offline. The workloads on the Cognitive SSD are read-only, which sustains the latency of the Cognitive SSD at a steady level with little fluctuation. Besides, in comparison to the B-CPU, the Cognitive SSD reduces latency by 69.9% on average. The performance improvement stems from the high-speed of data processing on the DLG-x accelerator compared to B-CPU. Due to the overhead of data movement caused by the bandwidth limitation of the I/O interface and onboard memory, the latency of B-FPGA and B-DLG-x is higher than B-GPU. Compared to the B-FPGA and B-DLG-x baselines, the Cognitive SSD reduces latency by 63.79% and 63.02% on average, which benefits from near-data processing. The average retrieval speed of B-GPU is 1.11x faster than Cognitive SSD because the execution of deep hashing costs more time on the resource-limited DLG-x compared to powerful GPUs. However, Cognitive SSD is more energy-efficient (QPS/Watt) than a GPU-integrated system by 2.44x, which is shown in Fig. 11(b). More importantly, the Cognitive SSD is implemented with FPGA and the operating frequency is only 100MHz. The performance will be better if the Cognitive SSD is implemented with ASIC or escalated operating frequency.

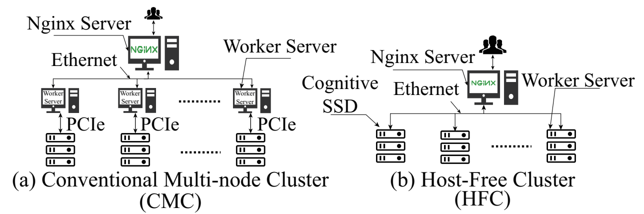

Fig. 12. The architecture of the conventional multi-node cluster(CMC)(a) and host-free cluster(HFC)(b).

5.6 The Cluster of Connected Cognitive SSDs

Experimental Setup. We evaluated the performance and the scalability of the Cognitive SSD when it scales into a multi-node cluster system. Fig. 12(a) shows the architecture of a conventional low-cost small-scale cluster system in a data warehouse. The cluster system consists of 10 worker servers and 1 Nginx [5] server. The Nginx server is responsible for load balancing. The worker nodes connect to the Nginx server by using TCP connections. In this case, we constructed four cluster system baselines by extending above four baselines: **BC-CPU**, **BC-GPU**, **BC-FPGA**, and **BC-DLG-x**. We issued requests to measure the QPS and the latency per request of the Cognitive SSD based cluster system when multiple users are accessing the web service shown in Fig. 10 concurrently via the ab tool.

Evaluation. Fig. 13(a) illustrates that when concurrent users equal to 400, all evaluated schemes rise slowly in experiments, which is limited by the thread of worker server and the node number of clusters. Meanwhile, the variation of peak QPS is due to the change of the performance bottleneck in different solutions. For example, the saturation performance of BC-GPU and CMC is constrained by the thread of the server while that of BC-CPU, BC-FPGA, and BC-DLG-x is determined by the latency of deep hashing inference and data movement. Fig. 13(b) shows that the QPS and latency per request of four baselines and CMC also change with the scale of the node cluster when the number of concurrent users reaches 400. As the number of nodes increases, the QPS gradually increases to the peak value, and the latency gradually decreases owing to improvement of service parallelism. When the nodes increase, the load-balancing mechanism of Nginx prevents a large hotspot formation in the cluster, which greatly increases the waiting time of requests.

We also measured the power consumption of the CMC system while running the CBIR service and compared it to BC-CPU, BC-GPU, BC-FPGA, and BC-DLG-x. When the cluster system is active, as shown in Fig. 14, the power consumption of a single node in the BC-CPU and BC-GPU are respectively 1.52x and 2.70x higher than a single node in the CMC. Similarly, Fig. 15 indicates the power of BC-CPU and BC-GPU is 1.45x and 2.46x than that of a CMC. Meanwhile, we also compared the QPS/Watt of CMC with other four baseline in Fig. 13(c). The energy-efficiency (QPS/Watt) of CMC

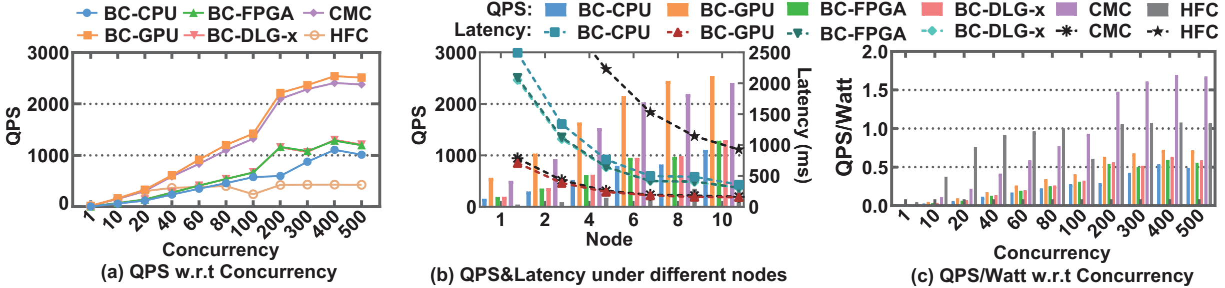

Fig. 13. Performance comparison.

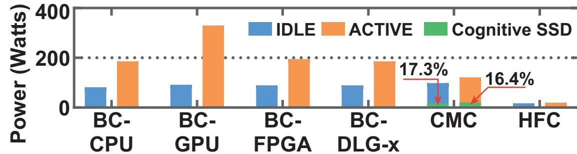

Fig. 14. Power dissipation of a single node.

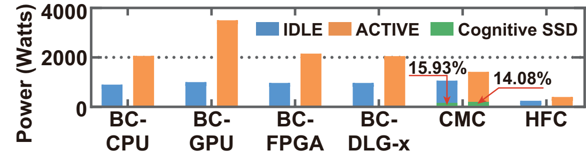

Fig. 15. Power dissipation of cluster.

is higher than the other four baselines because of short data movement path and the energy-efficient DLG-x accelerator. When the cluster system is IDLE, the power consumption is slightly higher than BC-CPU and BC-GPU because of the power consumption of the Cognitive SSD prototype is higher than the enterprise SSD.

It is noted that Fig. 15 indicates the power consumption of the Cognitive SSD only occupies about 15.93% (IDLE) and 14.08% (ACTIVE) of the entire system in the CMC architecture. The Cognitive SSD contains one Dual 1GHz ARM Cortex-A9 core, which could run embedded Linux system and has lower power consumption compared with the Intel Xeon CPU. Thereby, as shown in Fig. 12(b), to further reduce power consumption, we proposed the architecture of the host-free cluster (HFC) system, where the Cognitive SSD is directly connected to the Ngnix server via TCP connection, and the embedded Linux system runs a simple NAND flash management daemon and crow web framework.

We measured the performance of the host-free cluster system under the same experimental setup, which is illustrated in Fig. 13, Fig. 14, and Fig. 15. Fig. 14 shows that the power dissipation of a single node in the host-free cluster system is reduced by up to 89.2%, 93.9%, and 83.6% compared with that of BC-CPU, BC-GPU, and the original CMC when system is active. Considering the host server contains two Intel Xeon E5-2630 CPU that outperforms the dual Cortex-A9 in the Cognitive SSD, thereby, we measured the QPS per watt (QPS/Watt) to illustrate the energy-efficiency of the HFC system. The result (Fig. 13(c)) shows that when system concurrency is low, HFC delivers better energy-efficiency than the other four baselines and even better than the CMC architecture. The reason is that using high-performance machines to handle infrequent requests results in low energy-efficiency. Therefore, Fig. 13(c) witnesses the energy-efficiency of HFC relatively decreases with the increasing concurrency of the system. The performance growth of HFC under different node numbers also project that the level-off throughput is limited by the embedded CPU power instead of the DLG-x. In analysis, the HFC system will have much lower power consumption and higher performance if the Cortex-A9 processor is replaced by the latest Cortex-A series, e.g., a quad-core or octo-core Cortex-A75. Therefore, connecting the Cognitive SSD directly via interconnects contributes to much higher energy efficiency in Cognitive SSD system and guaranteed service throughput as well.

6 Conclusion

In this paper, we have introduced the Cognitive SSD, a near-data deep learning device that actively performs low latency, low power and high accuracy unstructured data retrieval. We have designed and implemented the Cognitive SSD with a direct flash-access deep hashing and graph search accelerator, to combat the complex software stack and inefficient memory hierarchy barriers in the conventional multimedia data retrieval systems. Our prototype demonstrates that the Cognitive SSD reduces latency by 69.9% on average compared to CPU, and more than 34.4% and 63.0% power saving against CPU and GPU respectively. Furthermore, the Cognitive SSD can scale to a multi-SSD system and significantly reduces the cost and power overhead of large-scale storage nodes in data centers. The demo of the retrieval system based on Cognitive SSD is available at [3] and part of the source code is available at [2].

Acknowledgments

We thank our shepherd, Joseph Tucek, and the anonymous ATC reviewers for their valuable and constructive suggestions. We thank the professor Jiafeng Guo of the CAS key lab of network data science and techology for his supports and suggestions. This work was supported in part by the National Natural Science Foundation of China under Grant 61874124, Grant 61876173,Grant 61432017, Grant 61532017, Grant 61772300 and YESS hip program No.YESS2016qnrc001.

References

[1] ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4. http://httpd.apache.org/docs/2.4/programs/ab.html.

[2] The Cognitive SSD. https://github.com/Cognitive-SSD.

[3] The Cognitive SSD Platform. http://cognitivessd.vicp.io:10110/.

[4] Flexible I/O tester. https://fio.readthedocs.io/en/latest/fio_doc.html#moral-license.

[5] NGINX. https://www.nginx.com/.

[6] Nvm express. https://nvmexpress.org/.

[7] The OpenSSD Project. http://openssd.io.

[8] Vivado. https://www.xilinx.com/support/download.html.

[9] Learning multiple layers of features from tiny images. Technical report, 2009. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf.

[10] The biggest data challenges that you might not even know you have, May 2016. https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/.

[11] Micron nand flash. page 239, 2017. https://www.micron.com/products/nand-flash.

[12] ZC706 Evaluation Board for the Zynq-7000 XC7z045 SoC User Guide (UG954). page 115, 2018. https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf.

[13] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 1–14, New York, NY, USA, 2009. ACM. http://doi.acm.org/10.1145/1629575.1629577.

[14] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson. Near-data processing: Insights from a micro-46 workshop. *IEEE Micro*, 34(4):36–42, July 2014. https://ieeexplore.ieee.org/document/6871738.

[15] Matias Bjørling, Javier González, and Philippe Bonnet. Lightnvm: The linux open-channel SSD subsystem. In *15th USENIX Conference on File and Storage Technologies (FAST'17)*, pages 359–374, 2017. https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling.

[16] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman. Active flash: Out-of-core data analytics on flash storage. In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, April 2012. https://ieeexplore.ieee.org/document/6232366.

[17] Deng Cai. A revisit of hashing algorithms for approximate nearest neighbor search. *CoRR*, abs/1612.07545, 2016. http://arxiv.org/abs/1612.07545.

[18] Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollow, Rajesh K. Gupta, and Steven Swanson. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 385–395, Washington, DC, USA, 2010. IEEE Computer Society. https://doi.org/10.1109/MICRO.2010.33.

[19] Intel IT Center. Big data 101: Unstructured data analytics. page 4. https://www.intel.com/content/www/us/en/big-data/unstructured-data-analytics-paper.html.

[20] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 269–284, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2541940.2541967.

[21] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 367–379, Piscataway, NJ, USA, 2016. IEEE Press. https://doi.org/10.1109/ISCA.2016.40.

[22] Sangyeun Cho, Chanik Park, Hyunok Oh, Sungchan Kim, Youngmin Yi, and Gregory R. Ganger. Active disk meets flash: A case for intelligent ssds. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 91–102, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2464996.2465003.

[23] Hyeokjun Choe, Seil Lee, Seongsik Park, Sei Joon Kim, Eui-Young Chung, and Sungroh Yoon. Near-data processing for machine learning. *CoRR*, abs/1610.02273, 2016. http://arxiv.org/abs/1610.02273.

[24] Arup De, Maya Gokhale, Rajesh Gupta, and Steven Swanson. Minerva: Accelerating data analysis in next-generation ssds. In *Proceedings of the 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '13, pages 9–16, Washington, DC, USA, 2013. IEEE Computer Society. http://dx.doi.org/10.1109/FCCM.2013.46.

[25] 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 *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1221–1230, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2463676.2465295.

[26] Cong Fu and Deng Cai. EFANNA : An extremely fast approximate nearest neighbor search algorithm based on knn graph. *CoRR*, abs/1609.07228, 2016. http://arxiv.org/abs/1609.07228.

[27] Cong Fu, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with navigating spreading-out graphs. *CoRR*, abs/1707.00143, 2017. http://arxiv.org/abs/1707.00143.

[28] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(12):2916–2929, December 2013. https://doi.org/10.1109/TPAMI.2012.193.

[29] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 Object Category Dataset, March 2007. http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001.

[30] Jaeseung Ha. crow: Crow is very fast and easy to use C++ micro web framework, June 2018. https://github.com/ipkn/crow.

[31] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2647868.2654889.

[32] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. Bluedbm: An appliance for big data analytics. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 1–13, New York, NY, USA, 2015. ACM. http://doi.acm.org/10.1145/2749469.2750412.

[33] Sang-Woo Jun, Andy Wright, Sizhuo Zhang, Shuotao Xu, and Arvind. Grafboost: Using accelerated flash storage for external graph analytics. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, pages 411–424, Piscataway, NJ, USA, 2018. IEEE Press. https://doi.org/10.1109/ISCA.2018.00042.

[34] Yangwook Kang, Yang-Suk Kee, Ethan L. Miller, and Chanik Park. Enabling cost-effective data processing with smart ssd. *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2013. ftp://ftp.cse.ucsc.edu/pub/darrell/kang-msst13.pdf.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. http://doi.acm.org/10.1145/3065386.

[36] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0). *CoRR*, abs/1610.02455, 2016. http://arxiv.org/abs/1610.02455.

[37] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1711–1717. AAAI Press, 2016. http://dl.acm.org/citation.cfm?id=3060832.3060860.

[38] K. Lin, H. Yang, J. Hsiao, and C. Chen. Deep learning of binary hash codes for fast image retrieval. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 27–35, June 2015. http://ieeexplore.ieee.org/document/7301269/.

[39] V. E. Liong, Jiwen Lu, Gang Wang, P. Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2475–2483, June 2015. http://ieeexplore.ieee.org/document/7298862/.

[40] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *2016 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pages 2064–2072, June 2016. http://ieeexplore.ieee.org/document/7780596/.

[41] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an rdma-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (USENIX ATC'17)*, pages 773–785, 2017. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu.

[42] Youyou Lu, Jiwu Shu, and Weimin Zheng. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*, pages 257–270, 2013. https://www.usenix.org/conference/fast13/technical-sessions/presentation/lu_youyou.

[43] Jian Ouyang, Shiding Lin, Zhenyu Hou, Peng Wang, Yong Wang, and Guangyu Sun. Active ssd design for energy-efficiency improvement of web-scale data analysis. In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, pages 286–291, Piscataway, NJ, USA, 2013. IEEE Press. http://dl.acm.org/citation.cfm?id=2648668.2648739.

[44] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. Sdf: Software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 471–484, 2014. http://doi.acm.org/10.1145/2654822.2541959.

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015. http://dx.doi.org/10.1007/s11263-015-0816-y.

[46] Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, Yanqin Jin, Yang Liu, and Steven Swanson. Willow: A user-programmable ssd. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 67–80, Berkeley, CA, USA, 2014. USENIX Association. http://dl.acm.org/citation.cfm?id=2685048.2685055.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. http://arxiv.org/abs/1409.1556.

[48] Yongseok Son, Nae Young Song, Hyuck Han, Hyeonsang Eom, and Heon Young Yeom. A user-level file system for fast storage devices. In *Proceedings of the 2014 International Conference on Cloud and Autonomic Computing*, ICCAC '14, pages 258–264, Washington, DC, USA, 2014. IEEE Computer Society. https://doi.org/10.1109/ICCAC.2014.14.

[49] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei Li. C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 123:1–123:6, New York, NY, USA, 2016. ACM. http://doi.acm.org/10.1145/2897937.2897995.

[50] Devesh Tiwari, Simona Boboila, Sudharshan S. Vazhkudai, Youngjae Kim, Xiaosong Ma, Peter J. Desnoyers, and Yan Solihin. 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, pages 119–132, Berkeley, CA, USA, 2013. USENIX Association. http://dl.acm.org/citation.cfm?id=2591272.2591286.

[51] Devesh Tiwari, Sudharshan S. Vazhkudai, Youngjae Kim, Xiaosong Ma, Simona Boboila, and Peter J. Desnoyers. Reducing data movement costs using energy efficient, active computation on ssd. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, HotPower'12, pages 4–4, Berkeley, CA, USA, 2012. USENIX Association. http://dl.acm.org/citation.cfm?id=2387869.2387873.

[52] Hung-Wei Tseng, Qianchen Zhao, Yuxiao Zhou, Mark Gahagan, and Steven Swanson. Morpheus: Creating application objects efficiently for heterogeneous computing. *SIGARCH Comput. Archit. News*, 44(3):53–65, June 2016. http://doi.acm.org/10.1145/3007787.3001143.

[53] Jianguo Wang, Dongchul Park, Yang-Suk Kee, Yannis Papakonstantinou, and Steven Swanson. Ssd in-storage computing for list intersection. In *Proceedings of the 12th International Workshop on Data Management on New Hardware*, DaMoN '16, pages 4:1–4:7, 2016. http://doi.acm.org/10.1145/2933349.2933353.

[54] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for Similarity Search: A Survey. *arXiv:1408.2927 [cs]*, August 2014. http://arxiv.org/abs/1408.2927.

[55] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family. In

Proceedings of the 53rd Annual Design Automation Conference, DAC '16, pages 110:1–110:6, New York, NY, USA, 2016. ACM. http://doi.acm.org/10.1145/2897937.2898003.

[56] Louis Woods, Zsolt István, and Gustavo Alonso. Ibex: An intelligent storage engine with support for advanced sql offloading. *Proc. VLDB Endow.*, 7(11):963–974, July 2014. http://dx.doi.org/10.14778/2732967.2732972.

[57] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *Int. J. Comput. Vision*, 119(1):3–22, August 2016. http://dx.doi.org/10.1007/s11263-014-0748-y.

[58] Huei-Fang Yang, Kevin Lin, and Chu-Song Chen. Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(2):437–451, February 2018. https://doi.org/10.1109/TPAMI.2017.2666812.

[59] Jiacheng Zhang, Jiwu Shu, and Youyou Lu. Parafs: A log-structured file system to exploit the internal parallelism of flash devices. In *2016 USENIX Annual Technical Conference (USENIX ATC'16)*, pages 87–100, 2016. https://www.usenix.org/conference/atc16/technical-sessions/presentation/zhang.

[60] Jie Zhang, Miryeong Kwon, Donghyun Gouk, Sungjoon Koh, Changlim Lee, Mohammad Alian, Myoungjun Chun, Mahmut Taylan Kandemir, Nam Sung Kim, Jihong Kim, and Myoungsoo Jung. Flashshare: Punching through server storage stack from kernel to firmware for ultra-low latency ssds. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, pages 477–492, Berkeley, CA, USA, 2018. USENIX Association. http://dl.acm.org/citation.cfm?id=3291168.3291203.

[61] Liang Zheng, Yi Yang, and Qi Tian. SIFT meets CNN: A decade survey of instance retrieval. *CoRR*, abs/1608.01807, 2016. http://arxiv.org/abs/1608.01807.