Towards Scalable Analytics with Inference-Enabled Solid-State Drives

Minsub Kim, Jaeha Kung, and Sungjin Lee

Abstract—In this paper, we propose a novel storage architecture, called an Inference-Enabled SSD (IESSD), which employs FPGA-based DNN inference accelerators inside an SSD. IESSD is capable of performing DNN operations inside an SSD, avoiding frequent data movements between application servers and data storage. This boosts up analytics performance of DNN applications. Moreover, by placing accelerators near data within an SSD, IESSD delivers scalable analytics performance which improves with the amount of data to analyze. To evaluate its effectiveness, we implement an FPGA-based proof-of-concept prototype of IESSD and carry out a case study with an image tagging (classification) application. Our preliminary results show that IESSD exhibits 1.81× better performance, achieving 5.31× lower power consumption, over a conventional system with GPU accelerators.

Index Terms—Solid-state drives, in-storage processing, deep neural networks, convolutional neural networks

1 Introduction

Recently, Deep Neural Network (DNN) has received serious attention since it produces high-quality results for difficult problems. High computation overhead of DNN algorithms was considered a huge barrier for their wide use in various applications, but this problem has alleviated by leveraging AI accelerators (e.g., GPU, FPGA, and ASIC) which are able to speed up DNN algorithms using specialized hardware [1].

The rapid advance of AI accelerators may move a bottleneck in serving DNN services from computation to *data transfer*. For flexible management of resources in data centers, large-scale systems are typically designed to separate application logics from data. However, such separation inevitably requires application servers where DNN services run to undergo remote data access over the network. Compared to the processing power of accelerators that has increased by 15 times over recent five years [2], the network throughput hasn't been improved well – 7.6 times for the same period [3]. Considering the data-intensive nature of DNN algorithms, this poor network speed would eventually be an obstacle in providing fast DNN services.

To address the above problem, this paper proposes a novel storage architecture, called an *Inference-Enabled SSD* (*IESSD*). By employing and utilizing DNN inference accelerators inside a solid-state drive (SSD), IESSD eliminates frequent data movements between applications and data storage, thereby boosting up DNN analytics performance. A key benefit of IESSD is that it can deliver scalable analytics performance that improves proportional to the amount of data to be analyzed. Since accelerators and data are combined into the same device, as the number of SSDs deployed increases (i.e., the amount of data stored increases), the ability of analyzing data improves accordingly. IESSD is more efficient in terms of energy and cost because it does not require us to attach another application servers to the network, which are costly and power-hungry.

The idea of placing accelerators near data is not new and has been investigated by many researchers [4], [5], [6]. Existing studies, however, have focused on offloading primitive math operations (e.g., matrix multiplication [4]) to an SSD or accelerating part of database queries (e.g., filtering [5] and join [6]) on the storage side. They neither take into account technical issues that arise when DNN inferences run on a resource-constrained SSD nor present proper designs to better integrate DNN accelerators with an SSD controller.

In this paper, we analyze the performance of an inference-enabled SSD which embraces FPGA fabrics to accelerate DNN inferences. Although modern SSDs employ multiple embedded cores (e.g., 4× 1 GHz ARM cores [7]) and DNN operations are accelerated by FPGA, the naive adoption of DNN inference engines cannot deliver desired performance, failing to efficiently utilize full hardware resources available in the SSD controller.

We observe that, owing to limited SSD core speed, a preprocessing step of a DNN inference accounts for a non-trivial proportion of the total inference time. We address this problem by distributing preprocessing tasks across multiple cores and running them with hardware in a pipelined manner. Second, implementing the whole inference algorithm in FPGA cannot fully utilize the high speed of FPGA. By adopting HW/SW partitioning that splits the algorithm into two parts, we better utilize hardware accelerators, improving the inference throughput. Finally, we found that, depending on DNN models, offloading some operations to the host side is a reasonable choice to scale up the inference speed.

We have implemented IESSD, along with three optimization techniques, in Xilinx's FPGA board with a custom flash card. As a case study, we build a full-fledged image tagging (classification) system that uses DNN inference engines to extract tags from image files stored in SSDs. We compare IESSD with a GPU-based AI acceleration system. Our experimental results confirm that IESSD's performance linearly scales with the amount of data stored in SSDs. In particular, when the number of IESSDs attached to a storage server exceeds 39, IESSD outperforms the GPU-based system. Moreover, ESSD exhibits 5.31× better energy efficiency over the typical system.

This paper is organized as follow. In Section 2, we describe an overall architecture of IESSD. After analyzing the performance of a naive IESSD design with an image tagging system, we detail optimization techniques to enhance IESSD in Section 3. Section 4 presents preliminary results and, in Section 5, we conclude with future work.

2 Inference-Enabled SSD Architecture

Fig. 1 illustrates an overall architecture of an AI acceleration system with SSDs. Application servers are responsible for executing user services, including DNN services, which fetch data from storage servers through the network, perform operations on the data, and return results to clients. Application servers may use GPU to accelerate service logics. A storage server is like a container of a set of SSDs which are connected to application servers via high-speed connections like iSCSI, PCIe, and Ethernet. One of the representative products in the market is JBOF [8] and all-flash array (AFA) systems. JBOFs and AFAs employ many SSDs, for example, up to 576 SSDs [8].

IESSD is designed as a replacement of a typical SSD in a storage server, but it also provides an ability of processing data. The overall design of IESSD is similar to that of an enterprise-class SSD; it has multi-core ARM CPUs to execute flash management algorithms (e.g., an FTL), along with large DRAM to keep metadata (e.g., a mapping table) and to cache data. IESSD uses FPGA as an accelerator because of its high performance, low-power consumption, and

- *The authors are with the Department of Information & Communication Engineering, DGIST, Daegu 42988, South Korea. E-mail: {appleeji, jhkung, sungjin.lee}@dgist.ac.kr.*

Manuscript received 28 May 2019; accepted 1 July 2019. Date of publication 20 July 2019; date of current version 3 Apr. 2020.
(Corresponding author: Sungjin Lee.)
Digital Object Identifier no. 10.1109/LCA.2019.2930590

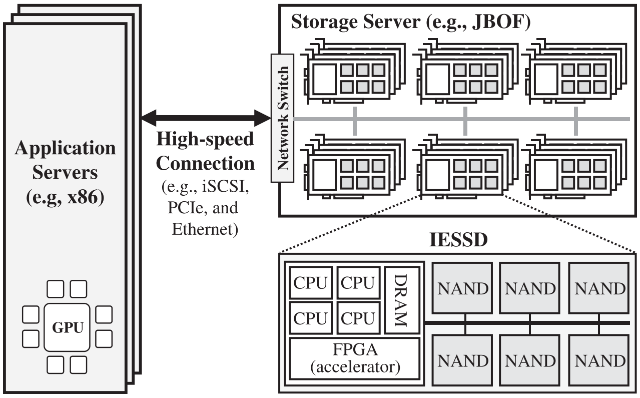


Fig. 1. Overall architecture of AI acceleration system with IESSDs.

reconfigurability, but different types of accelerators (e.g., GPU and ASIC) can be used. Owing to high computation overheads of a learning process, the current design of IESSD is focused on accelerating inference computation only. Efficient implementation of inference logics in FPGA is not our primary goal. Thus, we use Xilinx's ChaiDNN [9] which is less efficient in performance, but provides popular hardware IPs for DNN inferences. ChaiDNN is compatible with the Caffe framework, supporting various inference models (e.g., GoogleNet [10] and AlextNet [11]).

IESSD handles conventional I/O commands (e.g., read or write LBAs) in the same way as a typical SSD. When application-specific inference commands come (e.g., extracting tags from image files), IESSD makes use of the same ARM cores and FPGA logics available in the SSD controller to deal with them and only returns a result to an application server. It is important to define flexible interfaces between IESSDs and a server so that various application-specific commands can be supported. Prior studies already took into account this issue [5], so we do not consider it in the paper.

3 Case Study: Image Tagging System

Even though various DNN applications can run with IESSD, in this preliminary work, we carry out a case study of IESSD with a popular DNN-based application, image tagging. The image tagging system is composed of two main phases: (1) *tagging* and (2) *indexing* [12]. The tagging phase performs inference on images using DNN models and extracts associated tags; the indexing phase stores image/tag information on a database so that clients quickly retrieve relevant images they look for.

In general, the tagging and indexing are done in a streamed manner whenever new images are being uploaded to a photo storage [12]. Thus, it involves only a few I/O traffic to update a database and to store new images in storage servers. However, once existing DNN models are updated, a '*re-tagging*' process has to be performed to create recent and accurate tags for previously stored images.[1] This re-tagging process issues many reads to storage servers, creating a large volume of network traffic between SSDs and application servers. Since image files are accumulated over time, the amount of data that needs to be transferred and analyzed for new models keeps increasing.

With IESSD, almost all of the network traffic is removed because inference operations are performed internally within SSDs, and only image tags are delivered to application servers. Since accelerators are deployed with SSDs, the inference performance scales with the amount of data accumulated. Our initial results, however, reveal that the image tagging throughput of IESSD is much lower than our expectation: 30 images/sec. Even considering the reduction of the network traffic, it is 64.5 times slower than that of GPU-based inferencing on the application server side, making in-SSD inference less attractive. To understand the main causes of such slow performance, we analyze the detailed performance profile of the in-SSD inference.

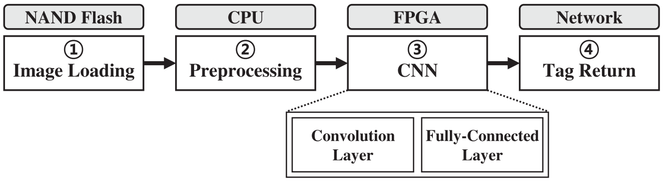


Fig. 2. Four main steps of the image tagging process and entities that perform at each step in the baseline IESSD design.

3.1 Performance Analysis of In-SSD Image Tagging

Fig. 2 details a tagging process with four steps, (1) *image loading*, (2) *image preprocessing*, (3) *convolutional neural network (CNN)*, and (4) *tag return*. Images are loaded from NAND flash during the image loading step and then are passed to the image preprocessing module run by CPU. The preprocessing module manipulates the image data so that they can be efficiently processed by CNN accelerators. Finally, image tags inferred by CNN are delivered to an application server via the network.

Fig. 3 reports the breakdown of the elapsed time at each step on four different CNN models: GoogleNet, AlexNet, ResNet-50, and VGGNet. The ILSVRC2012 validation set is used to assess the models. The image loading and tag return steps account for a negligible proportion (less than 2 percent) of the total inference time, thanks to the high internal bandwidth of NAND array and a small tag size, respectively. On the other hand, the preprocessing spends about 25 percent of the inference time, on average, while CNN spends 72 percent. In typical DNN frameworks (e.g., Caffe and TensorFlow), the preprocessing is implemented in software since its impact on performance is negligible, but it turns out to be significant when running in ARM CPUs. Even worse, in spite of being accelerated by FPGA, CNN takes a non-trivial portion of time. The above observation leads us to develop three optimization techniques, (1) preprocess pipelining, (2) HW/SW partitioning, and (3) host offloading, which make it possible to maximize the SSD resource utilization.

3.2 Preprocessing and CNN Pipelining

We can take several approaches to reduce the preprocessing time. The most effective way might be implementing the entire preprocessing tasks in hardware. This solution, however, requires

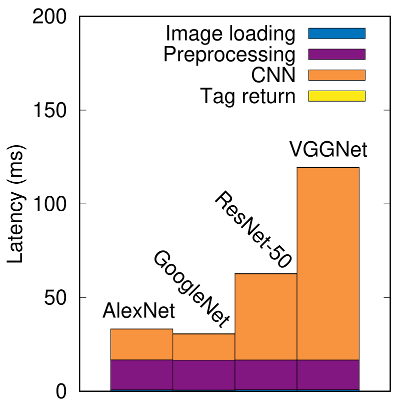


Fig. 3. Latency of each step.

[1] As an example, consider that an older DNN model that was only able to identify the type of mammals (e.g., cats or dogs) is improved to recognize specific species (e.g, American shorthair or Bengal cats). To reflect such a new model, previously stored images should be read and tagged again.

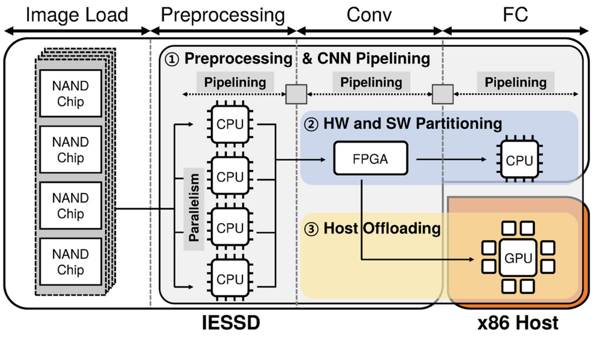


Fig. 4. Three optimization techniques in IESSD.

additional hardware resources, along with custom logics that should be able to decode various image sources (e.g., jpeg, png, and gif). Implementing part of the preprocessing tasks which are independent upon image types (e.g., resizing and normalization) may be feasible in terms of cost reduction.

IESSD improves the preprocessing speed in a cost-efficient way by (1) taking advantage of multicore CPUs and by (2) performing the preprocessing and CNN tasks in a pipelined manner. As in Fig. 4, IESSD reads desired images from NAND and distributes them across multiple cores so that the preprocessing tasks can be done in parallel. Modern SSDs are equipped with multiple CPUs (e.g., a quad-core CPU [7]), so it can reduce the preprocessing time considerably. The preprocessed data are then sent to the CNN module executed by FPGA. This pipelined execution of the preprocessing task in CPUs and the CNN task in FPGA enables us to further improve the inference throughput by hiding the shorter task behind the longer one.

Taking the examples of GoogleNet and AlexNet where the preprocessing is a burden, the inference throughputs improve by up to 1.99 and 2.22 times, respectively (see Fig. 6). This is because, by leveraging multiple cores, the effective preprocessing times per image are reduced to 4.5 and 4.7 milliseconds, and these are completely overlapped with the CNN processing time through the pipelining.

3.3 HW and SW Partitioning

The preprocess/CNN pipelining makes the CNN task a new bottleneck. To shorten the CNN time, we consider partitioning the CNN task into SW and HW. The CNN is composed of two main layers, a convolution (CONV) layer and a fully-connected (FC) layer (see Fig. 2). The CONV layer mostly performs arithmetic operations (convolutions) on input feature maps, which means that it is a computation-bound task that can be efficiently accelerated by FPGA. The FC layer, however, involves many accesses to DRAM, and the hardware performance is constrained by slow memory latency rather than computation. When both layers are implemented in FPGA, the maximum frequency of the resulting hardware is limited to 200 MHz, as it is bottlenecked by memory latency. On the other hand, the standalone CONV implementation in the hardware better utilizes FPGA resources and achieves higher effective frequency of 500 MHz with double-pumped DSPs [9], [13].

This leads us to split and run the two layers separately: that is, the CONV layer on FPGA and the FC layer on CPU. Note that the software FC layer is designed to leverage multiple CPU cores as well. Such HW/SW separation, however, does not result in the best performance all the time, because the software FC layer offers different performance depending on its DNN model. If the FC layer is simple and thin (e.g., GoogleNet), it spends only a few CPU cycles, not badly affecting inference throughput. A deep FC layer (e.g., AlexNet), however, requires lots of CPU cycles, slowing down the entire inference process. This indicates that HW/SW partitioning should be done with a careful consideration of a given model.

Step	AlexNet	GoogleNet
Load	134.2 KB	134.2 KB
Preproc	301.9 KB	294 KB
Conv	36 KB	4 KB
FC	4 KB	4 KB

Fig. 5. Output size after each step.

3.4 Host Offloading

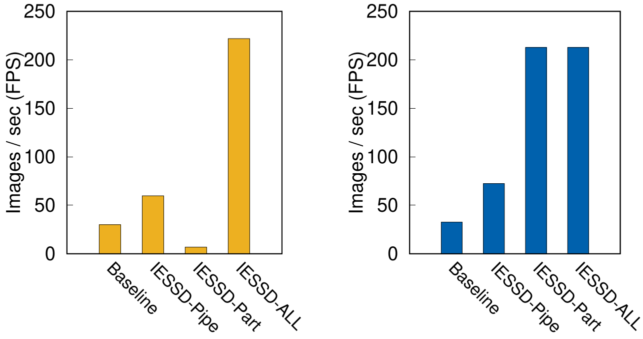
A deep FC layer cannot be efficiently executed by both FPGA and ARM CPUs, and thus it potentially acts as a main bottleneck. To address this, if it is necessary, IESSD offloads and runs it on a host side where a more powerful accelerator (e.g., GPU as in Fig. 1) is available. The idea behind this is based on two facts. First, the amount of data that has to be transferred to the FC layer is small enough. Given a image file of 134.2 KB, the intermediate output data from the CONV layer, which are then supplied to the FC layer as an input, reduce to 4 or 36 KB (see Fig. 5). Thus, offloading the FC layer to the host does not create heavy network traffic. Second, FC operations are lighter than the entire inference process and thus can be efficiently processed. Our preliminary experiments show that a consumer-class GPU (GTX 1080 Ti) is able to cope with requests sent from 147 SSDs simultaneously, achieving the throughput of 32,840 images/sec. It is 16.9x faster than 1,936 images/sec when the entire inference process is processed by GPU. It is worth noting that, to support the host offloading, the SSD and the host in IESSD use the same CNN model, sharing identical weight values.

4 Evaluation

We evaluated the performance of IESSD in two aspects: (1) standalone IESSD performance when each of the three optimization techniques was disabled or enabled and (2) aggregate performance when many IESSDs were grouped together in a storage server. The standalone performance was measured using the proof-of-concept prototype of IESSD implemented in Xilinx's ZCU102 with our custom flash card. ZCU102 has a quad-core ARM Cortex-A53 CPU running at 1.4 GHz and 4 GB DRAM. This HW specification is almost identical to those of commercial SSDs. Owing to the limited number of prototypes, we carried out simulations to analyze the aggregate performance, varying various parameters, such as the number of IESSDs, network throughput, and host GPU performance. All the numbers for the simulation were measured from real devices. Images from ILSVRC2012 were used as a benchmark.

Standalone IESSD Performance. We compared four different configurations of IESSD, `Baseline`, `IESSD-Pipe`, `IESSD-Part`, and `IESSD-ALL`. `Baseline` represents the naive implementation of IESSD with no optimization techniques. `IESSD-Pipe` employs the pipelining technique, and `IESSD-Part` is `IESSD-Pipe` with the HW/SW partitioning. Finally, `IESSD-ALL` is IESSD with all the optimizations.

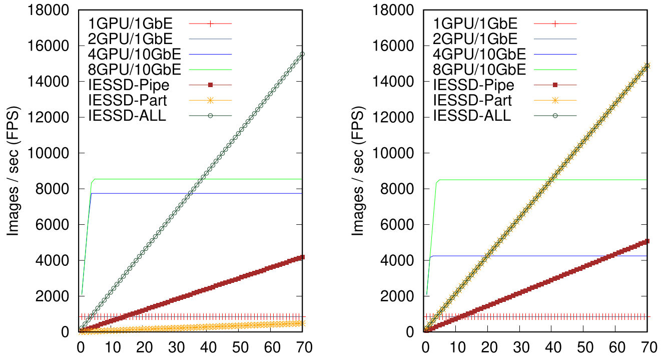
Fig. 6 shows that `IESSD-ALL` exhibited the best performance, achieving 221.9 and 212.7 throughput (images/sec) for AlextNet and GoogleNet, respectively. `Baseline` showed the worst performance, except for `IESSD-Part` in AlexNet. Compared to `Baseline`, `IESSD-ALL` showed 7.3× and 6.5× improved throughputs. `IESSD-Pipe` improved the inference throughputs by 199 and 222 percent for AlexNet and GoogleNet. As expected, `IESSD-Pipe` benefited from the use of four ARM CPUs that run in parallel with FPGA accelerators. `IESSD-Part` showed conflicting results. For AlexNet, it showed lower performance compared to `IESSD-Pipe`. This was owing to the deep FC layer of AlexNet which was a burden on ARM CPUs. Conversely, GoogleNet exhibited



(a) AlexNet

(b) GoogleNet

Fig. 6. Single IESSD performance.



(a) AlexNet

(a) GoogleNet

Fig. 7. Aggregate performance. The X-axis is the number of SSDs.

excellent performance with `IESSD-Part` thanks to its thin FC layer. In particular, cutting and offloading the FC layer to the host side didn't give any benefit in GoogleNet because ARM CPUs were able to cope with the FC operations efficiently.

For larger models, ResNet-50 and VGGNet, `IESSD-ALL` showed similar performance trends, exhibiting 5× and 3× higher throughputs than `Baseline` (not shown in Fig. 6). However, the absolute throughputs of ResNet-50 and VGGNet were 70 and 45 FPS, respectively. These low throughputs were due to high computation overheads of the large models.

Aggregate IESSD Performance. We evaluated the aggregate inference throughput of a storage server employing up to 70 SSDs. We compared two types of storage servers, one with typical SSDs and the other with IESSDs. In the typical setup, application servers fetch data from the storage server over network and perform the inference using GPU. To understand the impact of the network and GPU performance, we used four HW configurations, one GPU with 1 GbE NIC (`1 GPU/1 GbE`), two GPUs with 1GbE (`2 GPU/1 GbE`), 4 GPUs with 10 GbE (`4 GPU/10 GbE`), and 8 GPUs with 10 GbE (`8 GPU/10 GbE`). For IESSD, three different configurations, `IESSD-Pipe`, `IESSD-Part`, and `IESSD-ALL`, were evaluated.

Fig. 7 shows estimated inference throughputs depending on the number of SSDs. For AlexNet, `1 GPU/1 GbE` exhibited the worst performance that didn't scale with SSDs. Even with 2 GPUs (`2 GPU/1G bE`), AlexNet showed the same throughput as with one GPU (`1 GPU/1 GbE`), and this indicated that 1GbE NIC acted as a bottleneck. When the network throughput was increased to 10 Gb/s (`4 GPU/10 GbE`), huge improvement was observed. Doubling GPUs (`8 GPU/10 GbE`), however, was not helpful because the network became the bottleneck again, failed to supply enough data to eight GPUs. Note that an SSD model we used for the setting with typical SSDs was Samsung's 960 EVO that offered 273 MB/s read throughput. Thus, the 10 GbE NIC was not saturated until 5 SSDs were aggregated in RAID. IESSD exhibited scalable performance with more IESSDs. Particularly, `IESSD-ALL` outperformed `8 GPU/10 GbE` when more than 39 IESSDs were employed. Considering that recent JBOF and AFA employ many SSDs (up to 576 [8]), we can expect the same benefit in real-world settings.

We observed rather different trends in GoogleNet. Given the same network throughput, adding more GPUs improved overall inference throughput. This was because, unlike AlexNet, GoogleNet was more computation intensive. Note that the recent CNN models tend to have thin FC layers similar to GoogleNet. Putting accelerators inside SSDs, however, was beneficial even in such a case; when the number of IESSDs exceeded 40, `IESSD-ALL` outperformed `8 GPU/10 GbE`.

One might argue that by adding high-speed NICs (e.g., 40 GbE) and more GPUs, the conventional storage server would sufficiently outperform IESSD even if many SSDs with accelerators are employed. Another may argue that, considering its low throughputs for large models, IESSD would be useful for only limited applications. We expect this wouldn't be the case, as the accelerator design in IESSD also has enough headroom for its throughput improvement. Our HW IPs are built using the C-based high-level synthesis tool (i.e., ChaiDNN [9]), so they are not providing the maximum performance as a hardware accelerator. If they are reimplemented with a low-level HW language (e.g., HDL), we are able to provide much higher throughput with a smaller number of IESSDs. Detailed investigation with faster accelerators is left for future studies.

Power Consumption. Finally, we evaluated expected benefit in saving power. Using Xilinx's XPE tool, the power consumption of each IESSD is estimated to be 5.4W. If the host offloading is enabled, the power consumption increases to 6.03W and 5.67W for AlexNet and GoogleNet, respectively, since it partially uses host GPU. On the other hand, when one GPU is entirely used for the inference, it consumes 93.63W and 143.33W for AlexNet and GoogleNet. Based on those numbers, it is estimated that IESSD consumes 1.78× and 5.31× less power than 8 GPUs at the cross points where IESSD overtakes `8 GPU/10 GbE` (i.e., 39 and 40 IESSDs for AlexNet and GoogleNet in Fig. 7). This confirms that, for the same performance, IESSD will achieve better power efficiency.

5 Conclusion

To overcome the limitations of traditional AI acceleration systems, we proposed the Inference-Enabled SSD which is capable of performing DNN inference near data with FPGA accelerators. We evaluated the effectiveness of IESSD, in terms of performance, scalability, and power consumption, using an image tagging application. Our results confirmed that, when 39-40 IESSDs were employed in a storage server, IESSD outperformed the existing GPU-based AI acceleration system, achieving up to 5.31× power savings. As future work, we plan to implement IPs using HDL with better HW designs (e.g., prefetching/pipelining for the FC layer to hide memory latency). We also explore the pros and cons of various accelerators (e.g., mobile GPUs, TPUs, and ASIC) for in-storage inference.

Acknowledgments

This research was supported by the NRF grant funded by the Korea government (Ministry of Science and ICT) (NRF-2017R1E1A1A01077410) and the DGIST RD Program of the Ministry of Science and ICT (18-EE-01). Jaeha Kung was supported by the Ministry of Science and ICT (MSIT) under the ITRC support program (IITP-2019-2018-0-01433).

References

[1] N. P. Jouppi, et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.
[2] J. Huang, "A new AI era dawns," *Presented at the GPU Technol. Conf.*, Silicon Valley, 2018.

[3] N. N. Group, Nielsen's law of Internet bandwidth, Jan. 6, 2018. [Online]. Available: http://www.useit.com/alertbox/980405.html

[4] S. W. Jun, et al., "In-storage embedded accelerator for sparse pattern processing," in *Proc. High Perform. Extreme Comput.*, 2016, pp. 1–7.

[5] B. Gu, et al., "Biscuit: A framework for near-data processing of big data workloads," in *Proc. Int. Symp. Comput. Archit.*, 2016, pp. 153–165.

[6] S. Kim, et al., "In-storage processing of database scans and joins," *Inf. Sci.*, vol. 327, pp. 183–200, 2016.

[7] Hitachi Vantara, "Hitachi Accelerated Flash 2.0," White Paper, 2018.

[8] Y. T. Jin, et al., "Performance analysis of NVMe SSD-based all-flash array systems," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, 2018, pp. 12–21.

[9] Xilinx, Inc., "HLS based deep neural network accelerator library for Xilinx Ultrascale+ MPSoCs," 2019. [Online]. Available: https://github.com/Xilinx/CHaiDNN

[10] C. Szegedy, et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[11] A. Krizhevsky, et al., "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[12] J. Fu, et al., "Advances in deep learning approaches for image tagging," *APSIPA Trans. Signal Inf. Process.*, vol. 6, 2017, Art. no. e11.

[13] M. Reiche Myrgård, "Acceleration of deep convolutional neural networks on multiprocessor system-on-chip," Master Thesis, Department of Information Technology, Division of Computer Systems, Uppsala University, Uppsala, Sweden, 2019.