

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/339390297>

STANNIS: Low-Power Acceleration of Deep Neural Network Training Using Computational Storage Devices

Conference Paper · July 2020

CITATIONS

0

READS

334

6 authors, including:



[Ali Heydarigorji](#)

University of California, Irvine

8 PUBLICATIONS 15 CITATIONS

[SEE PROFILE](#)



[Mahdi Torabzadehkashi](#)

NGD Systems

5 PUBLICATIONS 19 CITATIONS

[SEE PROFILE](#)



[Siavash Rezaei](#)

University of California, Irvine

11 PUBLICATIONS 48 CITATIONS

[SEE PROFILE](#)



[Hossein Bobarshad](#)

University of Tehran

24 PUBLICATIONS 243 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Computational storage [View project](#)



Structural Health Monitoring [View project](#)

STANNIS: Low-Power Acceleration of Deep Neural Network Training Using Computational Storage Devices

1st Ali HeydariGorji
EECS Department
University of California, Irvine
heydari@uci.edu

2nd Mahdi Torabzadehkashi
NGD Systems Inc.
mahdi.torabzadeh@ngdsystems.com

3rd Siavash Rezaei
Computer Science Department
University of California, Irvine
siavashr@uci.edu

4th Hossein Bobarshad
NGD Systems Inc.
hossein.bobarshad@ngdsystems.com

5th Vladimir Alves
NGD Systems Inc.
vladimir.alves@ngdsystems.com

6th Pai H. Chou
EECS Department
University of California, Irvine
phchou@uci.edu

Abstract—This paper proposes a framework for distributed, in-storage training of neural networks on clusters of computational storage devices. Such devices not only contain hardware accelerators but also eliminate data movement between the host and storage, resulting in both improved performance and power savings. More importantly, this in-storage processing style of training ensures that private data never leaves the storage while fully controlling the sharing of public data. Experimental results show up to 2.7x speedup and 69% reduction in energy consumption and no significant loss in accuracy.

Index Terms—deep neural network training, training distribution, task parallelization, computational storage, near data processing, privacy, Federated learning

I. INTRODUCTION

In the past decade, the amount of data generated has grown at an exponential rate. In 2018, over 2.5 quintillion bytes of data were generated every day [1]. That equals 2.5 million terabyte hard disk drives (HDD) every single day. The emergence of the Internet of Things (IoT) is further accelerating this growth of data generation. Data is not meaningful unless processed; moreover, data often need to be processed in real-time or near real-time. For example, in self-driving cars, a large amount of data needs to be processed quickly and locally. The rate of data generation is very high, and the required processing is very heavy. Thus, they can create a backlog of unprocessed data, which prevents the system from making a proper decision. Although new metrics such as Age of Information (AoI) can help preserve the freshness of data by discarding the old data and replacing it with new, valid data [2], the problem with processing still persists.

With the rise in the use of machine learning algorithms, many problems can be solved at the cost of high computation. Up until recent years, the conventional method to process the massive volume of data was to send it to data centers, process, and resend the results back to the client. This method is associated with common problems such as latency, identity management, and data protection [3]. With the considerable

progress in the design of low-power processors such as ARM® A series, the edge computing has emerged as an alternative solution. Although edge computing can be beneficial for small size applications, data centers remain the backbone of heavy computation tasks, including the training of neural networks. Based on the size of the network and the number of samples, the training procedure can take weeks, even on the most powerful processing machines currently existed. Distribution of the training can significantly reduce the overall time, but obstacles to effective distribution include synchronization of the models, data transfer, and power dissipation.

In this paper, we propose a distributed in-storage processing (ISP) framework to address the issues above. The ISP technology is to process data inside the storage devices without sending the data to the host. The ISP-enabled storage devices, known as Computational Storage Devices (CSD), are equipped with an internal processing engine that is capable of processing data in-place. This technology is studied in the literature and has been shown to be beneficial for big data and HPC applications [4]–[7]. This technology reduces the data movements, and consequently, it improves both performance and energy consumption. Additionally, in cases where the applications are connected to the storage devices via a network, data should move through this network to be processed. This long-range data movement raises some concerns regarding the security and the privacy of data and makes it vulnerable to privacy attacks. The ISP technology completely solves this problem by avoiding the data movement. The overall contributions of this paper are as follows:

- a novel low-power CSD named Newport with augmented processing power;
- a framework named Stannis to efficiently parallelize training tasks on clusters of CSDs;
- a tuning algorithm to maximize the utilization of a heterogeneous system;

- protecting privacy with a hybrid of private and public data in a storage system.

The rest of the paper is structured as follows. Section II reviews related work on neural network training and computational storage devices. Section III introduces the Newport hardware and software stack, followed by our distribution framework named Stannis. Section V presents our experimental results.

II. BACKGROUND AND RELATED WORK

This section provides a background on the background on deep neural networks and their efficient processing on novel parallel architectures.

A. Deep Neural Networks

Deep learning has achieved tremendous success and has become one of the most powerful modeling tools combined with neural network. The idea behind neural network is the imitation of human neurons and is developed based on the perceptron model. Deep learning stacks multiple layers of neural network, in which the output of the current layer is fed as the input to the next layer. As an example, the supervised learning trains the neural network by taking the training dataset as input and comparing the output with the corresponding output label. After several training iterations, the parameters in the neural network are updated and able to do the prediction using the same type of input. Thanks to the idea of chain rule, back propagation is developed to train neural network more efficiently. As the name of the algorithm suggests, the update of parameters starts from the last layer and propagate back to the previous layer based on the comparison between the output and labels. In order to optimize the training process, Stochastic Gradient Descent (SGD) is used to reduce the error. Many machine learning libraries have been developed for model developing using neural networks, including Tensorflow, Theano, and PyTorch. However, due to the size and complexity of neural networks and the huge amount of training data, the training period has grown drastically from hours to days and weeks.

B. Parallelization of Training

A practical approach to reducing the training time is to parallelize it on multiple devices. Two common methods for parallelization are the model parallel and data-parallel. In the former, each processing node is responsible for training a part of the neural network; in the latter, all processing nodes have a replica of the entire network and update it locally, but since different copies of the network get different updates, a synchronization method is needed. The initial version of Tensorflow, a well known library for machine learning applications, uses a parameter server approach to synchronize worker nodes [8]. Each worker sends parameter updates to the parameter server where all updates are accumulated, averaged, and then sent back to the workers. This method can cause either computation or communication congestion on the server-side. Pytorch [9] uses a similar approach with a more

enhanced allreduce algorithm [10]. In 2017, Horovod [11] for distributed training of neural networks has drawn attention with its superior scalability compared to Tensorflow's standard distribution framework. Horovod was developed based on Nvidia's NCCL ring-allreduce algorithm [12], where each node communicates only with two peer nodes and pass updates along in a circular fashion. This method is bandwidth optimal, and each node's communication bandwidth is independent of the number of the nodes. However, where Horovod falls short is the support for heterogeneous processors, a problem we will address in Section III.

C. Hardware Accelerators

Another way to decrease the overall time is to use more powerful devices. Due to its nature of parallel architecture, GPUs are much faster than regular CPUs in floating-point operations that commonly occur in training. For instance, Nvidia Tesla V100 can run a maximum of 100 TFlops whereas an Intel® Core i9-7980XE can output a maximum of 1.3 TFlops. In 2016, Google unveiled its machine-learning chip called Tensor Processing Unit (TPU), a massively paralleled Application Specific Integrated Circuit (ASIC) for tensor operations that can deliver up to 420 TFlops in its latest version [13]. However, the downside of both GPU and TPU is that they are application-specific processors and are not as good as CPU for general operations. Field programmable gate arrays (FPGAs) are also widely used in accelerating different applications, however, they have a long design time and need to be reconfigured for executing different operations [14].

D. In-Storage Processing

Before the era of NAND flash memories, the I/O speed on hard disk drives (HDD) was the main obstacle in accelerating the operations. Although NAND flash memories are faster than their magnetic counterparts, moving data from the storage to the DRAM for processing still remains as the bottleneck. ISP is an innovative technology that addresses this shortcoming by moving computation to the data instead of moving data to the computation. ISP allows tasks to be executed in place, i.e., in the storage devices, with no need to move huge volumes of data. Based on the processing units, the available CSDs can be categorized into two main groups: 1) CSDs that exploit the processing unit dedicated to SSD controller for ISP [15], [16], and 2) CSDs with a dedicated ISP engine [5], [7], [17]. In the first approach, despite the energy efficiency of exploiting the SSD controller's processing units for the purpose of ISP, the gained processing power is rather limited due to their relatively low performance; moreover, doing so can also negatively impact the performance of the SSD. In the second approach, dedicated processing units such as FPGAs [15] or embedded processors [18] overcome the limited processing power by augmenting the storage units with more computing resources. Although FPGAs can be power efficient with a great performance improvement potential, they suffer from several limitations that make them unsuitable for ISP purpose, including 1) they require an RTL implementation of the tasks,

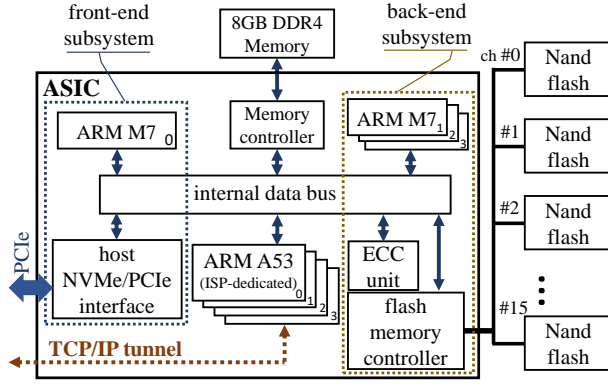


Fig. 1. Newport architecture

whose high-performance realization can be very challenging; 2) reconfiguration of FPGAs is needed for different tasks; 3) due to the lack of a file system structure, data cannot be accessed and used as a file concept. It has been shown that deploying a dedicated processor is a viable solution for ISPs [17]. In this paper, we propose a dedicated processor-based CSD and show the benefits of using it in the distributed training phase of learning algorithms.

III. CSD ARCHITECTURE

In-storage processing technology aims at augmenting storage devices with efficient processing engines, enabling them to process data locally without transferring it to a host system. Despite the simplicity of the concept, many of the previously proposed CSD architectures fall short of providing compelling flexibility and efficiency simultaneously.

In this research, we introduce Newport, an ASIC-based CSD architecture that provides a flexible and capable ISP environment for offloading and running user applications without modification. Newport is equipped with a full-fledged OS, which can support a wide range of programming models and languages. Newport is designed to embrace the ISP technology natively, which means its controller is composed of a dedicated ISP processing engine together with other conventional modules, all implemented in a single ASIC chip. Fig. 1 shows the hardware architecture of the Newport CSD, whose internal modules are categorized into three subsystems: front-end subsystem (FE), ISP-dedicated processing engine, and back-end subsystem (BE). Newport includes four ARM[®] M7 processors for running the conventional tasks of the storage device. The FE subsystem uses one of the four ARM[®] M7 processors and a host NVMe interface module. The interface module receives and depacketizes the NVMe commands, and the ARM[®] M7 processor checks the integrity of the commands and interprets them for the BE subsystem.

The CSD controller is connected to multiple NAND flash chips organized in 16 flash channels, which can do I/O operations in parallel. The BE subsystem manages all the flash channels and performs the I/O operations on the channels. The BE contains a flash memory controller unit, three ARM[®]

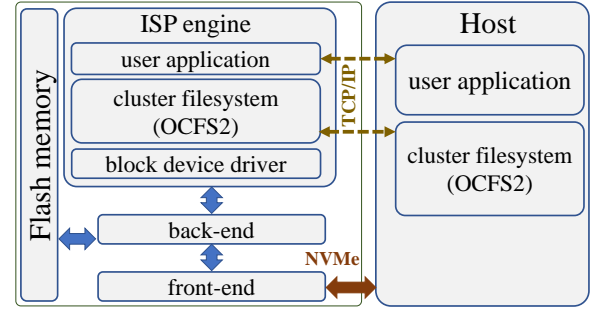


Fig. 2. Newport software stack

M7 processors, and an error correction code (ECC) unit for restoring the original data in case of bit errors from flash access. The BE also handles flash management tasks, including wear leveling, garbage collection, and flash memory logical-to-physical address translation layer (FTL). Besides these two subsystems, Newport is equipped with a unique and dedicated ISP engine composed of a quad-core ARM[®] Cortex A-53 processors and a complete software stack. The quad-core processor has access to an 8GB DRAM, which is shared among the processing units inside the CSD controller. The Software stack makes it possible to send ISP commands from the host to the Newport CSD and sending the ISP results back to the host.

All the subsystems in the Newport architecture are connected by a data bus, as shown in Fig. 1. Both the FE and ISP subsystems send flash read/write commands to BE, which is responsible for communicating with the flash memory chips to complete the flash I/O operations. In other words, while the data that is transferred to the host needs to go through the FE subsystem and a complex NVMe over PCIe link, the ISP engine bypasses the FE as well as the power-consuming NVMe link to read the data. Thus, the ISP subsystem has efficient and high-speed access to the data stored in the flash memory.

Fig. 2 shows Newport software stack architecture. The ISP engine runs a full-fledged Linux operating system (OS) both for running a wide spectrum of user applications in-place as well as developing the software stack. To provide filesystem-level access for user applications that run on the ISP engine, we have developed a *block device driver*, which makes an abstraction layer for flash I/O operations, so the OS on the ISP engine can mount the flash memory as a block device.

For communication purposes, a TCP/IP over PCIe tunnel is provided, which embeds TCP/IP packets inside the PCIe packets. Using this tunnel, user applications running on the host can communicate with the applications running on Newport. The TCP/IP tunnel is composed of a process running on the FE to transfer data from the ISP engine to the host and vice versa, a process running on the host which handles TCP/IP packetization on the host, and a similar process running on the Newport CSD to handle the packetization. These three processes work together to provide a TCP/IP connection

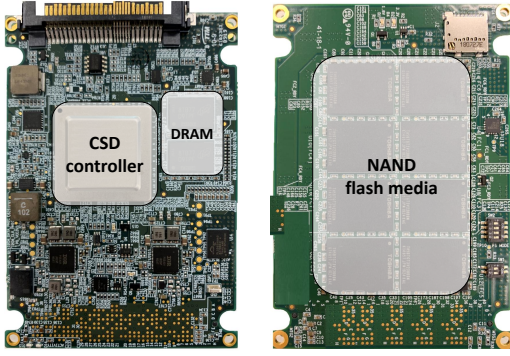


Fig. 3. Newport Prototype

between the ISP engine and the host. In the case of attaching multiple Newport CSDs to a host, the TCP/IP over PCIe tunnel spans over all the CSDs to provide a network covering all the CSDs and the host. Using the block device driver and the TCP/IP tunnel, user applications running on Newport have efficient and high-speed access to flash data, and can also communicate with the applications running on the host or other CSDs.

Since both the host and the ISP engine have concurrent access to the flash memory at the filesystem level, a synchronization mechanism should maintain the integrity of the filesystem metadata. To address this issue, we have ported Oracle Cluster Filesystem version 2 (OCFS2) on both the host and Newport CSD. OCFS2 has two agents that run on the host and the Newport CSD. These two agents communicate with each other through the TCP/IP tunnel to synchronize the filesystem metadata. We have prototyped a fully functional Newport CSD to show the feasibility of the proposed design and also to run experiments on a platform equipped with multiple Newport CSDs. Fig. 3 demonstrates the Newport prototype.

IV. SOFTWARE FRAMEWORK

System for TrAining of Neural Networks In Storage (*Stannis*) is a framework for the distribution of neural network training on homogeneous and heterogeneous systems. It overcomes Horovod’s inability to work efficiently on heterogeneous systems by workload scaling with consideration for transmission delay and data privacy.

Horovod shows great speedup on homogeneous systems, but on a heterogeneous system, the slowest processor becomes the bottleneck due to required synchronization in training. That is, a faster processing engine must wait for the slower ones. To address this problem, we try to equalize the processing time by setting different batch sizes for each processing engine, so that all processors wait for the least amount of time. In other words, the slower processor gets smaller batch size and thus, finishes the epoch in the same elapsed time as the more powerful processors.

Algorithm 1 shows the pseudo code for the proposed approach. It starts by running a series of benchmarks on

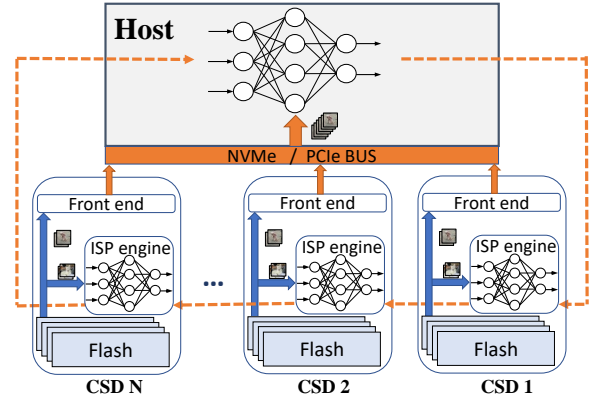


Fig. 4. Stannis Architecture

all processing engines to assess their processing speed and find the optimal batch size on each processor. Based on the results, the best batch size is selected for the slowest engine. Having the batch size for one device, we calculate the time it takes to finish one batch and find the best batch size on the other processing engines that can deliver almost the same elapsed time. We increase the batch size by a fraction ($\frac{1}{C}$) of the difference in time for the two processor, up until getting similar times. C is a constant that determines how much to adjust the batch size after each test. Larger C means more fine grained batch size update. We also consider the slowdown that occurs due to the synchronization process and allocates a ($\frac{\text{time}}{E}$) margin to the final time. We determined E by observing the slowdown pattern for the first number of processing nodes added to the system in our initial tests and defined it to give a fixed 20% margin to the results. Stannis also considers the access permissions of the data and assigns the private data to local ISP engine while sharing the public data with the ISP engine and the host processor. This method eliminates the transmission of private data over the network and to the host and increases the data protection level. One potential concern with varying the training batch sizes is the loss of accuracy. In practice, however, we observed that the range of changes in batch size makes no tangible impact on accuracy. While setting batch sizes can increase speed within one epoch, an imbalance in the datasets assigned to the computing nodes can stall faster nodes at the end of each epoch. After determining the optimal batch size, Stannis runs a load balancing algorithm to assign the proper number of input data to each node so all nodes finish after the same number of steps in one epoch. Since the host has access to more data than each individual CSD, it is convenient to determine the host’s dataset size based on the batch size ratios and the slower nodes dataset size. The following equation shows how to determine dataset size on host:

$$\begin{aligned} \text{steps}_{\text{per epoch}} &= \frac{\text{dataset}}{\text{batchsize}} \rightarrow \\ \text{dataset}_{\text{host}} &= \frac{\text{dataset}_{\text{card}}}{\text{batchsize}_{\text{card}}} \times \text{batchsize}_{\text{host}} \end{aligned} \quad (1)$$

In case the portion of the private data to be processed on Newport nodes are not equal, Stannis either uses more portion of the public data on the node with the smaller private database or duplicate the private data to maximize the rate of image per second. Stannis can run any network that is based on Keras or Tensorflow. Examples of such are VGG, ResNet, Inception, MobileNetV2, and DenseNet. Fig. 4 shows how the Stannis distributes and process data on system.

Distributing the training process means the central model receives updates less frequently that can cause accuracy loss. However, in [19], Goyal *et al.* show that this accuracy degradation can be avoided to a good extent by applying two strategies: a) a linear scaling up of the learning rate based on the number of passed epochs, and b) a warm-up strategy that uses lower learning rates at the start of training, which helps overcoming early optimization difficulties [20].

V. EXPERIMENTAL RESULTS

There are two scenarios for using Stannis in action. The first one is to run it on a server rack in a datacenter where the user data has been transferred to storage systems using end-to-end encryption to guarantee the privacy aspects. The second case considers Newport as a stand-alone system, capable of doing edge computing in an IoT environment. Examples of such cases are the autonomous driving cars where lots of sensor data is created every second that can be processed fully or partially in the car. For now, we stick to the first case and explore the second one in our future work with the addition of the Federated learning techniques.

To evaluate Stannis in the real world, we assembled an AIC 2U-FB201-LX server with an 8-core, 16-thread Intel® Xeon® Silver 4108 CPU with 32GB of DRAM and 24 Newport CSDs, each with 32 TB flash memory, giving us a storage server with a total of $24 \times 32 = 768$ TB capacity (see Fig. 5). We used an expanded TinyImageNet database [21] for neural network training and took 72000 images as public data and 12000 images as private distributed over all Newport



Fig. 5. A server with 24 Newport CSDs

TABLE I
PARAMETER TUNING FROM ALGORITHM 1

Network	Param	Flop	MAC	batch size		speed (img/sec)
				Host	Newport	Host / Newport
MobileNetV2	3.47M	7.16M	56M	315 / 25		31.05 / 3.08
NASNet	5.3M	10.74M	564M	325 / 15		47.31 / 2.80
InceptionV3	23.83M	47.82M	5.72G	370 / 16		30.80 / 1.85
squeezenet	1.25M	2.46M	861M	850 / 50		219.0 / 16.3

CSDs. We chose MobileNetV2 with 3.47 million parameters and 56 million multiply-and-accumulate (MAC) operations as our main neural network. To compare the speedup on different neural networks, we then ran the test for NASNet, InceptionV3, and SqueezeNet. The only concern in choosing a network and the batch size is the available DRAM on the systems. Large batch size on big networks can saturate the DRAM and thus stall the entire training process. The 6 GB DRAM on Newport proved to be enough for most of the test cases. The solution for cases with DRAM saturation was to choose a smaller batch size. Since the processing speed converges to a certain number after a certain batch size, this reduction in batch size would not affect the processing speed. For instance, the images-per-second speed for MobileNetV2 on Newport is about 3 images per second for all batch sizes greater than 16. This happens as a result of the full utilization of the processing engine when the task becomes computation intensive rather than communication intensive.

A. Performance Analysis

Stannis started with running the tuning algorithm for the MobileNetV2, which gave us the optimal batch sizes of 25 and 315 and the processing speed of 3 and 32.3 images per second for the Newport and the host, respectively. It then determined the number of input data corresponding to each node to avoid stall on individual cards and then ran the benchmarks on a mixture of host and number of cards. We ran the same test for NASNet, InceptionV3, and SqueezeNet as well. The optimized batch sizes and corresponding speed for different networks are shown in Table I. Fig. 6 shows the processing speed in the form of an image per second for different networks. There is a slowdown in all processing nodes in a distributed model, which is due to partial stalls when nodes are synchronizing the parameters. The slow down pace fades out, and the individual nodes performance converges to a certain speed after the number of nodes grows beyond 5-6 devices. The relative speedup for different networks is also shown in Fig. 7. The result shows that the smaller networks get better speedup than

Algorithm 1: Stannis's Tuning algorithm

Input: $IP_{Newport}, IP_{host}, C$
Output: $(BS_{host}, BS_{Newport})$
Function Tune($IP_{Newport}, IP_{host}, C$):
 for batch sizes in list of BS **do**
 run *benchmark* on Newport;
 update $BS_{Newport}$ to the best one;
 update $time_{Newport}$;
 end
 Let $E = \text{margin scale}$;
 while $(time_{host} \cdot time_{Newport}) < (time_{Newport} / E)$ **do**
 $BS_{host} += BS_{host} \times (time_{Newport} \cdot time_{host}) / C$;
 run *benchmark* on host ;
 get the $time_{host}$;
 end
 return $(BS_{Newport}, BS_{host})$;
End Function

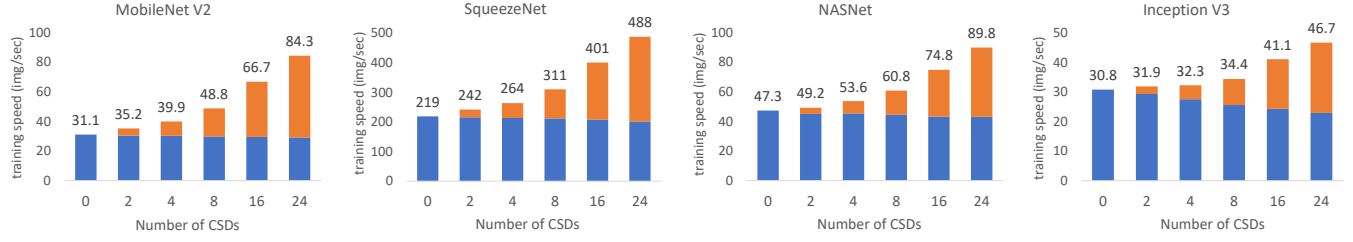


Fig. 6. Stannis Performance for different deep neural networks

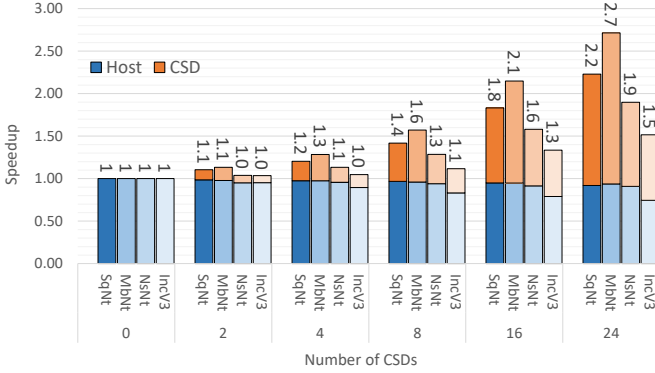


Fig. 7. Speedup for different deep neural networks

TABLE II
ENERGY CONSUMPTION

Number of CSDs	0	4	8	16	24
Energy per image (J)	13.10	8.30	6.84	5.05	4.02
Energy saving (%)	0%	37%	48%	62%	69%
FLOPS per watt	5.87M	7.05M	8.18M	10.37M	12.26M

the larger ones; this is because the more parameters to update, the more time it takes to synchronize the nodes. Another important parameter is the number of MACs. As Fig. 7 shows, SqueezeNet with 2.46M Flops gets less speedup compared to MobileNetV2 with 7.16M Flops, because it has 15x more MACs.

B. Power Analysis

The closest product to Newport that we could find on the market was the 11 TB Micron MTFDHAL11TATCW-1AR1ZAB SSD. We evaluate the power consumption of an AIC server with 24 11-TB Micron SSDs against the same system with 24 32-TB Newport CSDs. We used an off-the-shelf power meter to measure the input power to the entire server rack. Table II shows the energy per processed image for MobileNetV2. We skip the results for other networks since the power measurements were almost identical. Measurements show up to 69% saving in energy per processed image and 2x FLOPS per watt with 24 Newport CSDs compared to host alone.

C. Accuracy Analysis

To assess the accuracy degradation due to distribution of the training process, we ran a training session with 416,000 images on a single node and on six nodes to compare their accuracy and loss. The results show the loss increased from 1.1859 on a single node to 1.1907 on six nodes, or a 0.5% increase, and the same accuracy of 0.31 on both setups. Note that what is relevant here is not the absolute accuracy but the fact that our heterogeneous distributed setup yields the same accuracy as the centralized one.

VI. CONCLUSION

This paper introduces a new in-storage processing solution to accelerate deep neural network training. It achieves up to 2.7x speedup while reducing energy by up to 69%. Our solution consists of novel hardware named Newport, a computational storage device (CSD) that augments processing power to the systems and eliminates the need to move data to the host for processing. The hardware runs workload that is distributed by our software framework named Stannis, which extends Horovod to take advantage of heterogeneous processing units by optimizing the batch size and controlling the portion of data allotted to each processing node. Not only is this beneficial to both power and performance but more importantly it protects privacy by never moving private data out of the storage system. We plan to extend Stannis to a general framework that can parallelize any application on heterogeneous architectures and also develop a federated learning framework for training on mobile devices.

REFERENCES

- [1] "Becoming a data-driven CEO." [Online]. Available: <https://www.domo.com/solution/data-never-sleeps-6>
- [2] A. Javani, M. Zorgui, and Z. Wang, "Age of information in multiple sensing," 2019.
- [3] W. A. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," in *2011 44th Hawaii International Conference on System Sciences*. IEEE, 2011, pp. 1–10.
- [4] M. Torabzadehkashii, S. Rezaei, A. HeydariGorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Computational storage: an efficient and scalable platform for big data and hpc applications," *Journal of Big Data*, vol. 6, no. 1, p. 100, 2019.
- [5] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu et al., "Bluedbm: An appliance for big data analytics," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2015, pp. 1–13.

- [6] M. Torabzadehkashi, A. Heydarigorji, S. Rezaei, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Accelerating hpc applications using computational storage devices," in *21st International Conference on High Performance Computing and Communications*. IEEE, 2019, pp. 1878–1885.
- [7] B. Gu and et al., "Biscuit: A framework for near-data processing of big data workloads," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 153–165.
- [8] Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation OSDI*, 2016, pp. 265–283.
- [9] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.
- [10] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *Journal of Machine Learning Research*, vol. 15, pp. 1111–1133, 2014.
- [11] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018.
- [12] "Bringing HPC techniques to deep learning." [Online]. Available: <http://andrew.gibiansky.com/>
- [13] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU Platforms for Deep Learning," *arXiv e-prints*, p. arXiv:1907.10701, Jul 2019.
- [14] S. Rezaei, K. Kim, and E. Bozorgzadeh, "Scalable multi-queue data transfer scheme for fpga-based multi-accelerators," in *36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 374–380.
- [15] S. Kim, H. Oh, C. Park, S. Cho, S.-W. Lee, and B. Moon, "In-storage processing of database scans and joins," *Information Sciences*, vol. 327, pp. 183–200, 2016.
- [16] D. Park, J. Wang, and Y.-S. Kee, "In-storage computing for Hadoop MapReduce framework: Challenges and possibilities," *IEEE Transactions on Computers*, 2016.
- [17] M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Catalina: in-storage processing acceleration for scalable big data analytics," in *27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2019, pp. 430–437.
- [18] M. Torabzadehkashi, S. Rezaei, V. Alves, and N. Bagherzadeh, "Comptor: An in-storage computation platform for scalable distributed processing," in *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 1260–1267.
- [19] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training ImageNet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: <http://arxiv.org/abs/1706.02677>
- [20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: <http://arxiv.org/abs/1512.03385>
- [21] "Tiny imagenet visual recognition challenge." [Online]. Available: <https://tiny-imagenet.herokuapp.com/>