International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 11, Number 2, pages 516–532, July 2021

An In-Network Parameter Aggregation using DPDK for Multi-GPU Deep Learning

Masaki Furukawa
Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: furukawa@arc.ics.keio.ac.jp

Tomoya Itsubo
Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: itsubo@arc.ics.keio.ac.jp

Hiroki Matsutani
Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: matutani@arc.ics.keio.ac.jp

Received: February 16, 2021
Revised: May 5, 2021
Accepted: May 31, 2021
Communicated by Shinya Takamaeda-Yamazaki

Abstract

In distributed deep neural network using remote GPU nodes, communication occurs iteratively between remote nodes for gradient aggregation. This communication latency limits the benefit of distributed training with faster GPUs. In this paper, we therefore propose to offload the gradient aggregation to a DPDK (Data Plane Development Kit) based network switch between a host machine and remote GPUs. In this approach, the aggregation process is completed in the network using extra computation resources in the network switch and efficiently overlapped without increasing workload on remote nodes. The proposed DPDK-based switch supports reliable communication protocols for exchanging gradients data and can handle a part of MPI over TCP-based communication. We evaluate the proposed switch when GPUs and the host communicate with a standard IP communication over 40GbE, a PCI Express (PCIe) over 40Gbit Ethernet (40GbE) product and MPI communication over 10GbE, respectively. The evaluation results using a standard IP communication show that the aggregation is accelerated by 2.2-2.5x compared to the aggregation executed by a host machine. The results using the PCIe over 40GbE product show that the proposed switch outperforms the aggregation done by the host machine by 1.16x. The evaluations using MPI communication using Jetson Xaviers cluster show that the proposed switch provides up to 5.5-5.8x faster reduction operations than the conventional method.

1 Introduction

Distributed deep learning using multiple GPUs is widely used to reduce the training time. In data parallel training, training process has 1) GPU computation phase for back-propagation and 2) communication phase for parameter aggregation. The computation phase is efficiently accelerated by using GPUs. However, the communication phase is not accelerated, and its overheads increase with

GPU worker count. The performance of modern GPUs has improved significantly, the aggregation overheads limit the benefit of distributed training with remote GPUs [13]. This paper focuses on gradient aggregation in the communication phase of distributed training.

In this paper, we assume remote GPU environments using network-attached GPU technologies, where a host machine and GPUs are connected via Ethernet. In these remote GPU environments, since the aggregation process is performed on the host machine, the workload is imposed on the host machine. Moreover, the aggregation throughput depends on the performance of the host machine. We therefore propose a DPDK (Data Plane Development Kit) [1] based network switch which performs the aggregation in the network. The parameter aggregation is offloaded to the software network switch and executed by using its extra computing resources. The proposed DPDK-based switch supports a part of reliable MPI protocols over TCP-based communication for exchanging gradients. We evaluate the switch when GPUs and the host machine communicate with a standard IP communication over 40GbE, ExpEther 40G (a PCIe over 40GbE product) [15] and MPI communication over 10GbE, respectively. The evaluation results using 40GbE show that the proposed switch provides a higher throughput compared to the aggregation performed on the host machine. The evaluations using MPI communication in embedded GPU cluster consisting of power-efficient Jetson Xaviers [2] show that the aggregation throughput is accelerated by the proposed switch. These three case studies demonstrate that the proposed approach can be applied to the performance improvement of distributed training with multiple GPUs. [1]

This paper is organized as follows. Section 2 introduces distributed training approaches and remote GPU extension technologies. Section 3 proposes the DPDK-based network switch that performs the gradient aggregation. Section 4 describes the remote GPU configuration and the implementation. Section 5 evaluates the aggregation performance using the proposed switch. Section 6 concludes this paper.

2 Related Work

2.1 Distributed Training

The training process is divided into three phases: forward pass, backward pass, and optimization. In the forward pass, a prediction is performed for an input data. In the backward pass, the gradient for each parameter is calculated based on the prediction error. In the optimization, parameters are updated using these gradients so that the error becomes smaller. Generally, these calculations are performed on a GPU and can be parallelized by using multiple GPUs.

There are two approaches to parallelize the training using multiple GPUs: data parallel and model parallel [5]. In the data parallel, the same network is prepared for each GPU, and the training dataset is divided and applied to the neural network model of each GPU. On the other hand, in the model parallel, training is performed by dividing parameters on the network into each GPU. This paper employs the data parallel training approach, which is currently common in industry. The distributed training using multiple GPUs requires an aggregation process called All-Reduce. Figure 1 shows the flow of a single training iteration in the distributed training. In the distributed training, the gradients calculated by each GPU need to be aggregated between the backward pass and the optimization. By performing this All-Reduce process, a large-scale distributed training using a huge minibatch becomes possible. Then, the model parameters are updated based on these aggregated gradients in the optimization phase.

Several variants of the gradient descent optimization algorithm have been proposed. The commonly used basic optimization algorithm is SGD (Stochastic Gradient Descent). Here, it is assumed that minibatch β_i is assigned to corresponding GPU_i and it calculates a local gradient $g_i = \frac{\partial l_{\beta_i}}{\partial w}$, where l_β denotes the loss function for minibatch β. Thus, in SGD, the weights are updated as

[1] This work is an extended version of our previous work appeared in [9] by redesigning the proposed switch to support MPI over TCP-based communication and adding its case study using the Jetson Xavier cluster.

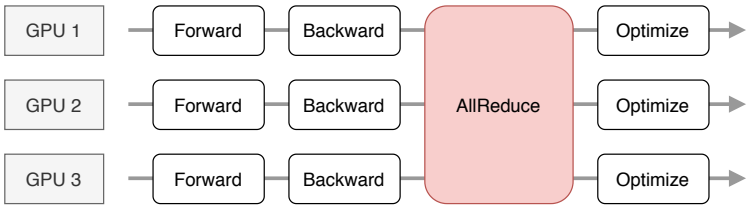


Figure 1: Single iteration in distributed training

follows:

$$w^{(t+1)} = w^{(t)} - \eta \cdot \sum_i \frac{\partial l_{\beta_i^{(t)}}}{\partial w^{(t)}} = w^{(t)} - \eta \cdot \sum_i g_i^{(t)},$$

where $w^{(t+1)}$, $w^{(t)}$, $g^{(t)}$, and η denote the next updated weights, the current weights, the current gradients, and the learning rate, respectively. Then, the next minibatch is trained. This paper focuses on the gradient aggregation process. We employ synchronous training, which is superior in terms of learning accuracy and convergence speed [7], but our proposal below is also useful for asynchronous training.

2.2 Network-Attached GPU Technology

Due to limitations such as the number of PCIe slots and power supply, the number of GPUs that can be equipped on a single machine is limited. To mitigate these limitations, there is an approach to connect GPUs remotely via networks. This approach is also effective in terms of GPU utilization and power efficiency. In this approach, such network-attached GPUs can be used transparently as well as directly-connected GPUs. The following describes two representative remote extension technologies.

ExpEther

ExpEther [15] [16] is a PCIe over Ethernet technology that makes GPUs available via Ethernet. Each GPU and a host machine are connected to Ethernet using a hardware ExpEther bridge. PCIe packets for GPUs are encapsulated into Ethernet frames by this bridge and transmitted to remote nodes. We employ the ExpEther 40G product as one case of remote GPU environment in this paper.

rCUDA (remote CUDA)

rCUDA [8] is widely used as a software service to use GPUs via networks. rCUDA is based on client-server model. In this approach, GPUs are connected to server machines via PCIe slots, and the server machines are connected to networks. Clients request the server machines for GPU tasks via TCP/IP communication. Then, the server machines instead of the clients perform the requested tasks using GPUs.

2.3 Communication Bottlenecks

In distributed training using remote GPUs, communication between multiple remote GPUs is required to aggregate gradients. The gradient aggregation is a sequential process. Moreover, as the number of GPUs increases, parallel computation parts become faster, but overheads of the aggregation process increase. Communication latency of the aggregation process therefore is one of the bottlenecks in distributed training. In [12], state-of-the-art DNN (Deep Neural Network) models are trained using a five-node GPU cluster with 10GbE connections. In this case, it is reported that the communication time is a significant part of the total training time. In [11], four parameter optimization algorithms, such as SGD and Adam, are accelerated in 10GbE FPGA-based network switch.

In addition, some methods to accelerate large-scale distributed DNN using GPU clusters are proposed [6] [14]. In [6], a GPU cluster consisting of 128 nodes with 1,024 Tesla P100 GPUs completed a training of ImageNet in 15 minutes. The nodes are connected with Infiniband FDR as a high-speed interconnect. As the underlying communication libraries, they use NCCL (NVIDIA Collective Communication Library) and MPI. Although these collective communication algorithms perform the aggregation efficiently, the communication overheads increase as the number of GPUs increases. In [6], it is reported that approximately 20% of the training time with 1,024 GPUs is spent for the communication.

Mellanox SHArP (Scalable Hierarchical Aggregation Protocol) technology [10] is known as a reduce communication offload technology. Reduction operations based on SHArP can be offloaded into Mellanox's Switch ASIC over the Infiniband network, which provides for low communication latency between machines equipped with GPUs. This paper targets communication of the network-attached GPUs cluster over Ethernet represented by ExpEther.

Computing node products each consisting of several GPUs (e.g., NVIDIA DGX series) have been used for high-performance computing purposes. This paper rather focuses on building low-cost clusters, where several GPUs and host CPU with limited PCIe slots are loosely-coupled with commodity technologies such as Ethernet.

3 DPDK Switch-Based Aggregation

3.1 Remote GPU Environment

This section proposes a DPDK-based acceleration of gradient aggregation in distributed deep learning. First, an assumed remote GPU configuration and bottlenecks in this configuration are described here. The remote GPU environment consists of a host machine, an Ethernet switch, and multiple remote GPU workers, and each component is connected via 40GbE / 10GbE network cables. As mentioned in Section 2.2, general-purpose machines cannot mount many GPUs directly at the same time due to the limited number of PCIe slots. In the remote GPU environments, the number of available GPUs is increased by introducing Ethernet switches between the host machine and GPU workers (please see Figure 3).

In this case, gradients data are aggregated by a CPU on the host machine for several reasons. First, the approach of using a GPU for aggregation causes unbalanced workload among GPUs. Second, the GPU does not have enough memory to hold all gradients data. Third, since the aggregation calculation is done by simple vector additions, the overhead of copying gradients to the GPU for the aggregation cannot be ignored. Communication latency between remote nodes during the aggregation limits the benefit of faster GPUs. Figure 2 shows a breakdown of the execution time using two ExpEther I/O boxes each equipped with a GPU (GeForce GTX 1080Ti). In this measurement, Pytorch is used as a deep learning framework, ResNet 152 is used as a DNN model, and SGD is used as an optimization algorithm. Gradient computation and Optimization are executed by two GPUs, while gradient aggregation is executed by a CPU on the host machine.

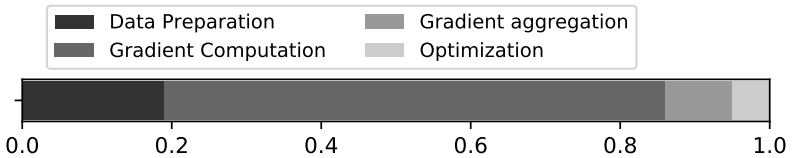


Figure 2: Breakdown of execution times in training phase

As shown in Figure 2, the gradient aggregation accounts for approximately 9% in this case. Although GPU computation is dominant with a few GPUs, the proportion of this communication increases with the number of GPU nodes, which lowers the training throughput.

3.2 Gradient Aggregation using Network Switch

In recent years, the parameter size of DNN models has been increasing with the benefit of faster GPUs. As the model size and the number of connected GPUs increase, aggregation throughput is limited by the network bandwidth, especially between an Ethernet switch and a host machine. We simply estimate the minimum transmission time in the network for each iteration as follows. Given a model size of M, with N GPU workers participating, and with B bandwidth interconnection, assuming gradients data from all GPU workers arrive at once. In this case, the minimum transmission time between the switch and each GPU is $2M/B$, while that time between the switch and the host machine is $2MN/B$. This means that the available bandwidth between the switch and each GPU is limited to B/N.

In this paper, we therefore propose to offload the gradient aggregation to the network switch in the network. Figure 3 illustrates the proposed system, where gradients are aggregated by the network switch that connects the host machine and each GPU. This approach prevents the transmission time from increasing proportionally to the number of GPUs. The aggregation using the network switch enables the wide bandwidth to be used efficiently. In addition, since the aggregation is done by the network switch, the saved compute resources at the host machine can be used for other computation tasks. In Section 4, we will implement the proposed switch by considering two GPU cluster cases: one with GTX 1080Ti GPUs and another with Jetson Xaviers that are power-efficient embedded GPUs. In Section 5, we will evaluate it in these cases.

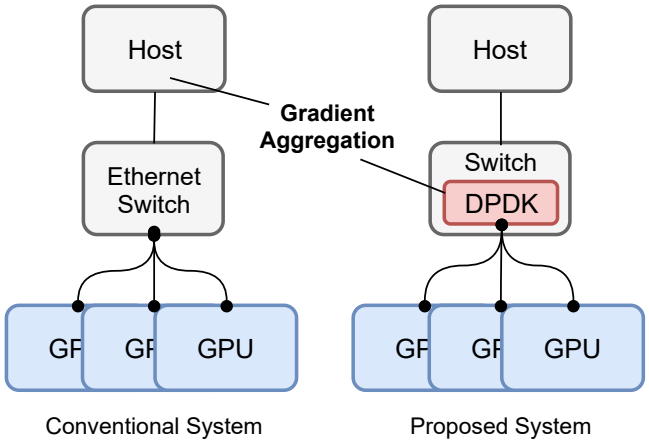


Figure 3: System overview. Gradients are computed by GPUs, then transmitted to a host machine. In our proposed system, gradient aggregation is done by DPDK-based intermediate switch.

3.3 DPDK-Based In-Network Aggregation

We offload gradient aggregation to a software network switch. Software switches are typically programmable and can add memory modules; so they are useful for the aggregation task that needs to synchronize gradients data. There are many ways to implement the software network switch. In this paper, we employ DPDK [1], which enables high-speed packet processing. Figure 4 shows an in-network aggregation using DPDK. In DPDK, the dedicated thread occupying a CPU core constantly polls NIC (Network Interface Card), and an application controls the NIC directly. In distributed training with remote GPU environments, all gradients data transmitted from each GPU node pass through the intermediate switch. Then, the gradients data are routed and aggregated at the same time by the proposed switch. By performing the aggregation in the switch, the gradients data are reduced by $1/N$ in the network, where N is the number of participating GPUs. Furthermore, the intermediate switch can structurally receive gradients data from remote GPUs all at once. This structural advantage is also beneficial in overlapping reduction operations with inter-node communication, which helps to reduce the aggregation time without increasing workload on remote nodes.

In this paper, the proposed switch is applied to the following three communication protocols for exchanging gradients: 1) UDP/IP communication, 2) a proprietary protocol for PCI Express over Ethernet and 3) MPI over TCP-based communication.

Our proposed method can be implemented at both the following: 1) a Linux-based software switch and 2) a host machine. In this paper, the proposed method is implemented at the software switch as one case of the above two implementation methods. When assuming that a host machine has an enough number of PCIe slots and each remote GPU is directly connected to the host machine, our proposed function would be implemented at the host machine.

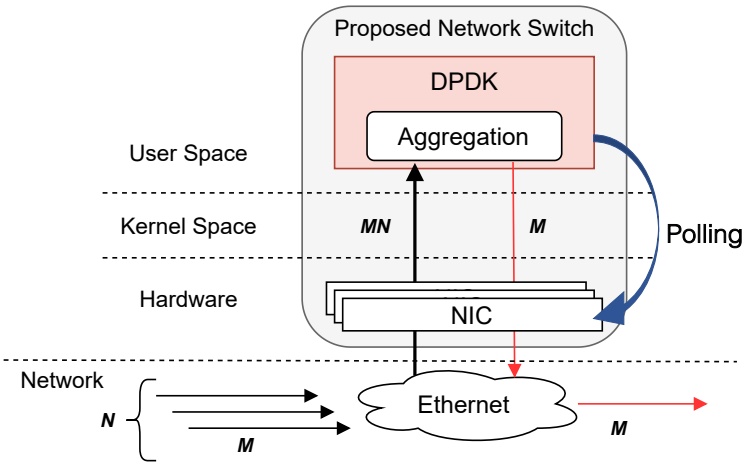


Figure 4: In-network aggregation using DPDK. N: GPU count, M: gradient size, $\rightarrow$: Gradients, $\rightarrow$: Aggregated gradients

4 Implementation

4.1 Remote GPUs Environment Setup

In this paper, we consider the following three cases as remote GPU environments:

Case 1) remote invocation of GPU tasks over a network, inspired by rCUDA, where the proposed system is verified using a standard TCP/IP communication.

Case 2) ExpEther 40G, where the proposed switch is applied to an actual remote GPU technology.

Case 3) an embedded GPU cluster consisting of Jetson Xavier, where the proposed switch is applied to the low-cost GPU cluster using MPI communication.

In Case 1, the proposed system using the proposed DPDK-based aggregation switch is implemented and verified using TCP/IP communication as a prototype implementation. In Case 2, we apply the proposed switch to the actual remote GPU technology using ExpEther 40G products. 40GbE network is used as an interconnect between a host machine and GPU nodes in these two cases. In Case 3, the proposed switch is extended and additionally evaluated for the power efficient embedded GPU cluster using MPI communication that is commonly used in practice. We employ 10GbE network for the interconnect in Case 3. We implement the environments using Pytorch as the deep learning framework.

4.2 Implementation Overview

4.2.1 Case 1

Figure 5 shows the configuration of Case 1 and its data flow. This environment consists of a host machine, worker machines equipped with a GPU, and the proposed DPDK-based aggregation switch. In Case 1, each learning iteration proceeds as follows: 1) The host machine extracts minibatches from a training dataset and distributes them to each GPU. 2) The GPU of each worker machine calculates gradients by applying the minibatch assigned by the host machine to its own DNN model. 3) The calculated gradients data are sent to the host machine. Please note that our proposed switch is implemented based on layer-2 switches, thus the destination IP address of gradients packets sent from each GPU devices is set to the host machine IP address. 4) These gradients are aggregated on the network switch instead of the host machine. 5) These target gradient packets are kept in the packet buffers and the local gradient values of payloads are overwritten to the aggregated gradients values. 6) The aggregated gradient packets are multicasted by the switch to each GPU, not to the host machine. 7) Instead of receiving the gradients from each GPU, the host machine receives a signal indicating completion of the aggregation from the switch. These steps can be overlapped to mitigate communication latency. In this implementation, we employ UDP/IP communication for the gradients data communication from the perspective of simplicity of implementing the protocol stack at user space of the proposed switch. TCP/IP communication is used for other communication. Considering the transmission efficiency and packet buffer size in DPDK libraries, the packet length containing gradients data is 1516 bytes, which include additional headers indicating the type of data and the sequence number of gradients. Packet loss is detected by checking the additional headers information. Please note that no packets loss occurs in our evaluation in this paper.

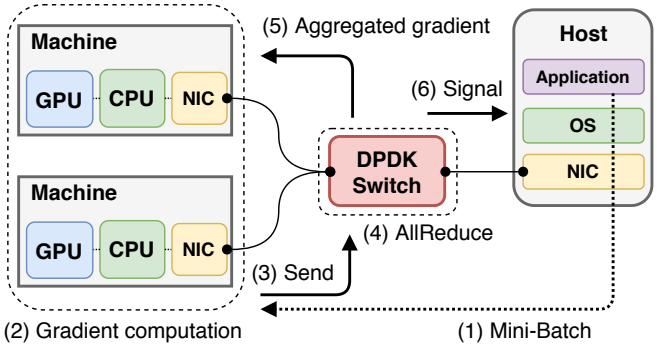
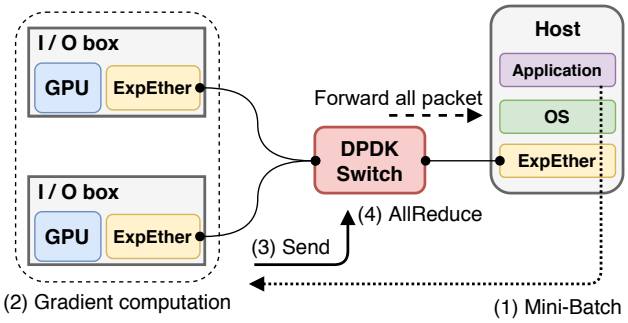


Figure 5: Configuration of Case 1 using remote invocation of GPU tasks

4.2.2 Case 2

Figure 6(a) shows the configuration of Case 2 and its data flow. This environment consists of a host machine, I/O boxes equipped with GPUs, and the proposed switch. Figure 6(b) shows the I/O box with one GPU mounted. The host machine and GPUs are connected via 40GbE network using ExpEther bridges. Figure 6(c) shows the 40G host adapter. The PCIe packets containing the calculated gradients data are encapsulated into Ethernet frames and transmitted to the host machine using proprietary protocols (e.g., transmission rate control, retransmission for lost packets). This reliable communication does not permit any changes to packets. Therefore, when manipulating the gradients packets, our proposed switch clones the original gradient packets and forwards them to the host machine in order to prevent communication interruptions and retransmissions with our current implementation. The flow from (1) to (4) is the same as that in Case 1.



(a) Configuration of Case 2 using ExpEther

(b) I/O box with one GPU

(c) Host adapter

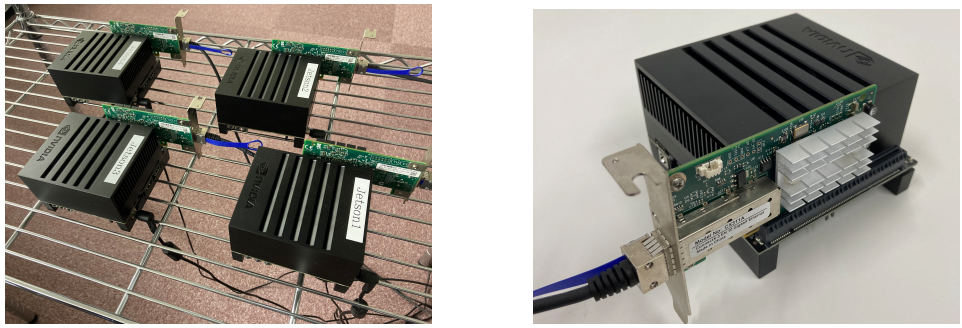
Figure 6: ExpEther 40G environment. ExpEther I/O boxes with GPUs (b) and host adapters (c) are connected to each other via DPDK-based 40GbE switch

4.2.3 Case 3

In Case 3, we use an embedded GPUs cluster consisting of Jetson Xavier as an example of low-cost GPU clusters. This environment consists of a host machine, Jetson Xaviers, and the proposed switch connecting them via 10GbE connections. Figure 7 shows the GPUs cluster consisting of four Jetson Xaviers, where a NIC is installed to the PCIe slot of each Jetson Xavier. We employ Open MPI [3] for gradients data communication, which is one of the most widely used communication protocols for high performance computing purposes. We run Open MPI using TCP-based communication over IP interfaces/networks. In this implementation, MPI_Send() and MPI_Recv() functions in Open MPI libraries are used for exchanging gradients data between nodes. The gradients calculated on each Jetson Xavier are transmitted to the host machine all at once using MPI_Send() and aggregated by the proposed intermediate switch. In our proposed system, the gradients data are manipulated by the switch in the network and do not reach the host machine. When using reliable communication, each GPU node and the host machine need to receive pseudo responses from the switch. Thus, the proposed switch is extended to support a part of TCP and MPI functions for gradients aggregation in this case. Instead of the host machine, the extended switch returns pseudo ACK responses for the received gradients packets to each GPU. Then, the switch sends a communication error as a completion signal to the host machine's error handler instead of forwarding the gradients data. The switch aborts communication after receiving the errors and moves to next processes.

4.3 DPDK-Based Aggregation Switch

Figure 8 illustrates an overview of the implemented DPDK-based aggregation switch. Figure 8(a) and Figure 8(b) show the 40GbE switch used in Case 1-2 and the 10GbE switch used in Case 3, respectively. DPDK applications receive packets directly in user space via a polling-based receiving mechanism called PMD (Poll Mode Driver). The received packets are placed in the packet buffer pre-allocated on Hugepage, and the application obtains the packet pointers using DPDK libraries. The aggregation switch mainly consists of a sending / receiving thread group and a gradient aggregation thread group. These threads are implemented as DPDK threads based on the polling mechanism, not regular Linux threads. The receiving threads guarantee the order of received packets, and perform packet classification processing using the headers. In the implemented 40GbE switch (Figure 8(a)), the packets including the gradients data to be aggregated are passed from the receiving threads to the aggregation threads. Then, the aggregation threads extract the gradients data from the received packets and temporarily store the data in gradient buffers allocated on Hugepage in advance.

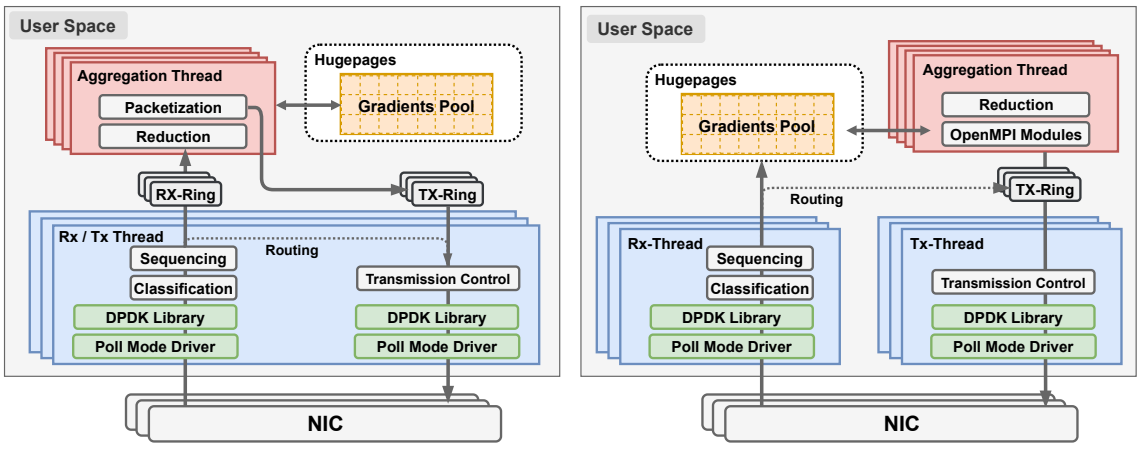


(a) An embedded GPUs cluster consisting of four Jetson Xaviers

(b) A Jetson Xavier with 10GbE NIC card

Figure 7: Environment of Case 3 using Jetson Xaviers.

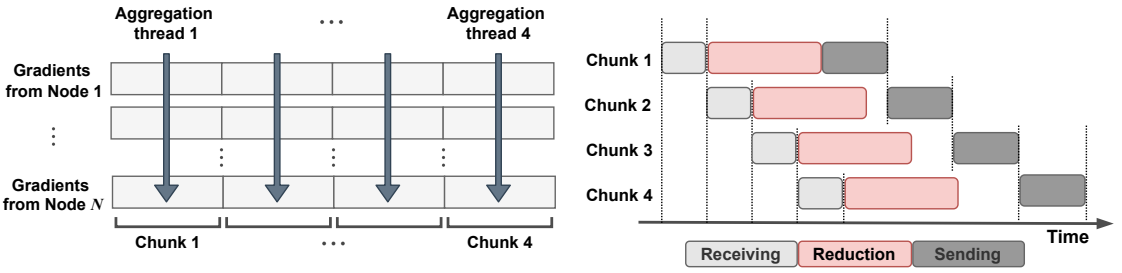
Assuming that the DNN model size is M and N GPU workers participate, an additional memory MN is required in the network switch for aggregation. In the implemented 10GbE switch (Figure 8(b)), the packets including the gradients data are directly stored based on Open MPI protocols by the receiving threads. This 10GbE switch is redesigned to support MPI over TCP-based communication. Open MPI modules to support the Open MPI communication for gradients data are implemented in the receiving threads and the aggregation threads. Note that these MPI modules are only a part of MPI functions used for gradient aggregation, not full MPI. Packets not to be aggregated are forwarded based on their header information and MAC address tables held by the threads of each port.



(a) 40GbE aggregation switch

(b) 10GbE aggregation switch

Figure 8: An overview of the DPDK-based 40GbE / 10GbE aggregation switch. (a): A sending/receiving thread is implemented for each NIC port. (b): A sending thread and a receiving thread are implemented for each NIC port and run independently. One logical core is assigned to each DPDK thread. Assuming that the number of NIC ports is P and the gradient reduction is executed by X aggregation threads, the total number of logical cores required in each switch is $P + 2X$ and $2P + X$, respectively.



(a) A way of organizing aggregation threads.

(b) A way of overlapping processes for gradient chunks

Figure 9: A way of gradient aggregation in the proposed switch. (a): Each gradient array is divided into chunks. Each aggregation thread works independently on the same chunk in all gradient arrays. (b): Communication processes and reduction operations for gradient chunks are overlapped each other.

There are several ways to organize threads to perform aggregation. Figure 9(a) illustrates a way of gradient aggregation used in this switch for gradient arrays from N GPU nodes. In this way, each gradient array obtained by each NIC port is divided into chunks of a predefined size. Each aggregation thread works on the same chunk in all gradient arrays. When chunks with a specific index are received from all GPU nodes, the aggregation thread assigned to that chunk starts aggregating. Then, when the aggregation operation is completed for the chunk, the aggregation thread overwrites the local gradient values with the aggregated gradient values and requests the corresponding sending / receiving threads to send the packets. Figure 9(b) shows a way of overlapping communication processes and reduction operations for gradient chunks to reduce the aggregation time. In actual environments, there is a delay in the time until gradient arrays are received from all GPU nodes. In particular, for larger gradients data, a longer waiting time occurs until the data are fully received. In this thread organization, data receiving / sending tasks and reduction operations can be efficiently overlapped with less inter-thread synchronization. Thus, this is a preferable way to organize aggregation threads.

5 Evaluations

5.1 Packet Processing Throughput

5.1.1 Case 1 and 2: 40GbE Switch

First, the proposed DPDK-based network switch is evaluated in terms of packet processing throughput. In this evaluation, the proposed switch is directly connected to NIC port of another machine with 40GbE cables. Each NIC is installed to PCIe 3.0 x16 slot of the machines. Test packets including gradients data are generated at a line rate of 40Gbps by using DPDK-Pktgen [4] and sent continuously from the packet generator machine to the proposed switch. The throughput is measured from the number of packets processed per second by the proposed switch. Table 1 shows the execution environment used for the proposed 40GbE switch. In our prototype implementation (Case 1), each packet contains 366 gradients and thus the packet length including a packet header and the payload is 1516 bytes. In the case of ExpEther, PCIe TLP (transaction layer packet) frames for GPUs are encapsulated into Ethernet frames. The maximum payload size of the TLP in our system used is 128 bytes, which is the same size used in [15]. In this case, the payload length of each gradients packet is 128 bytes (i.e., 32 gradients) and its header length is 64 bytes, and thus the total packet length is 192 bytes.

As a result of measuring the throughput in both cases, the average throughput of the proposed switch is 39.41Gbps and 26.01Gbps, respectively. Although test packets are continuously transmitted at 40Gbps, the 40GbE line rate is lower than 40Gbps when considering the Ethernet preamble and interframe gap inserted for each packet. Considering these overheads, the theoretical maximum

throughput in our environment is 39.48Gbps for Case 1 and 36.23Gbps for Case 2. In Case 1, the measured throughput of gradient aggregation is 99.8% of the maximum throughput, and thus almost the line rate is achieved. However, in Case 2, the measured throughput is decreased to 71.8% of the maximum. From these results, ExpEther's communication bandwidth is limited by the proposed switch due to the following two factors. First, the packets size used for forwarding the gradients data between ExpEther bridges is as small as 128 bytes in our environment. Second, our current implemented switch creates the clone packets of the gradients packets and forwards them to the host machine in order to prevent communication interruption and retransmission. This unnecessary one-copy overhead would be mitigated by considering ExpEther's proprietary protocols. Regarding the throughput, there is still enough room for improved implementations, such as using RSS (Receive Side Scaling) and multi-threading the receiving thread, which will be addressed in our future work.

Table 1: 40GbE DPDK-switch execution environment

OS	Ubuntu 18.04
CPU	Intel Xeon E5-2637 v3 @3.5GHz
Memory	512GB
Hugepages	1GB x4
NIC	Intel Ethernet CNA XL710-QDA2
DPDK	18.11.2
Pktgen	19.10.0

5.1.2 Case 3: 10GbE Switch

We evaluate the packet processing throughput of the proposed 10GbE switch used in Case 3. The proposed switch is directly connected to NIC ports of another machine running DPDK with 10GbE cables. The packet generator machine generates the packets used for transmitting gradients data in Open MPI library and sends them to the proposed switch at a line rate of 10Gbps using multiple DPDK threads. The throughput is measured by counting the number of the packets processed for a certain period time in the proposed switch. Table 2 shows the execution environment used for the proposed 10GbE switch.

Figure 10 shows the throughput for several different packet sizes. When Open MPI communication is used, gradients data are basically transmitted over Ethernet with a packet size of 1518 bytes. In this case, the measured throughput reaches the theoretical line rate with zero packet loss. Also, all packet sizes over 128 bytes almost reach the line rate. Note that the throughput with 64 bytes (gray bar) simply indicates switching throughput for packets that do not contain gradients data to be aggregated.

Table 2: 10GbE DPDK-switch execution environment

OS	Ubuntu 18.04
CPU	Intel Xeon E5-2637 v3 @3.5GHz
Memory	512GB
Hugepages	1GB x4
NIC	Intel Ethernet Converged Network Adapter X520-DA2
DPDK	19.11.6

5.2 Aggregation Performance

5.2.1 Case 1

In this section, the proposed DPDK-based switch is evaluated in terms of the execution time of gradient aggregation using remote invocation of GPU tasks. The evaluation environment consists of three remote GPU worker nodes, one host machine, and the switch connecting them. In this

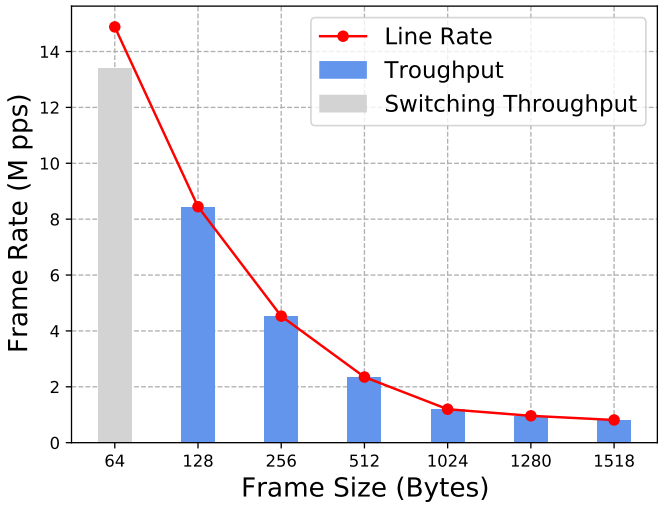


Figure 10: Throughput for gradient packets of the proposal switch

measurement, the execution time from when the packet including gradients data is received from any GPU node to when the aggregated gradients are completely transmitted to each remote GPU is measured. CIFAR 100 is used as a training dataset, and several state-of-the-art DNN models with different sizes are used for the evaluation. Table 3 shows DNN models and their gradient sizes used in our evaluation. To verify the effectiveness of the proposed switch, we compare the aggregation executed by the switch with the conventional aggregation done in an application layer of the host machine. Four threads are used for the aggregation task in both measurements. Table 4 shows the execution environments for the worker and host machines.

Table 3: DNN models

Model	Parameter size
GoogleNet	24.7 MB
VGG 19	80.2 MB
ResNet 50	94.0 MB
ResNet 101	170.0 MB
ResNet 152	232.6 MB

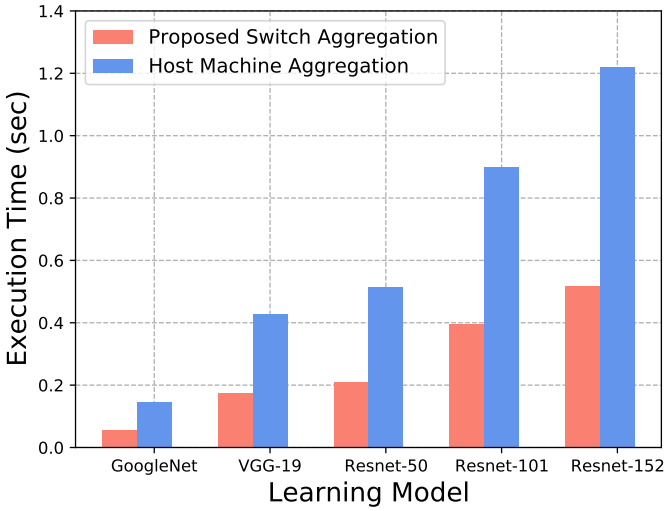


Figure 11: Execution time of gradient aggregation in Case 1

Figure 11 shows the execution times of the gradient aggregation for each model, which are average times of 100 iterations. In this graph, X-axis represents DNN models and Y-axis represents

their execution times. As a result, the aggregation in the network using the proposed switch (red bar) outperforms the aggregation on the host machine (blue bar) by 2.2-2.5x. These improvements are achieved by reducing unnecessary communication overheads using DPDK. By performing the aggregation using the network switch, the transmission time of gradients can be kept constant relative to the number of nodes. Therefore, it is considered that further benefits can be gained especially when using more GPUs.

Table 4: Worker and host execution environments

	Worker machine	Host machine
OS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS
CPU	Intel Xeon E5-2637 v3 @3.5GHz	Intel Core i7-4790 @3.6GHz
Memory	128GB	8GB
GPU	GeForce GTX 1080Ti (11GB RAM)	-
CUDA	10.0	10.0
Pytorch	1.3.0	1.3.0
NIC	Intel Ethernet CNA XL710-QDA2	Intel Ethernet CNA XL710-QDA2

5.2.2 Case 2

We evaluate aggregation execution time when using ExpEther as the actual remote GPU environment. In ExpEther, sophisticated communications based on proprietary protocols are performed between ExpEther bridges. These communications contain many shorter packets, which lowers the processing throughput of the proposed switch. In the proposed switch, the protocols for returning ACK packets or broadcasting aggregated gradients data to remote GPU devices without ExpEther hardware bridges are not implemented. Thus, we evaluate our proposed switch in a simple way. In this simple evaluation, the execution time of the aggregation is measured using 32,000 parameters consisting of 1,000 packets. This execution time is defined as the time until the addition operation for all gradients arrays from each GPU is completed. The evaluation environment consists of two I/O boxes each equipped with a GPU, one host machine, and the proposed switch. Figure 12 shows the comparison of the aggregation execution time. Note that this time does not include processing for the aggregated gradients. In this case, the aggregation performed on the switch is 1.16x faster than that on the host machine. In this evaluation, the parameter size used is small. In addition, packet forwarding for controlling ExpEther protocols increases unnecessary overheads. Thus, there is room for improvement in our measurement and implementation, but the evaluation results show that the proposed switch is useful for the actual remote GPU environment.

In this evaluation, the execution time of the aggregation is improved by approximately 1.16x. Based on the breakdown of training time shown in Section 3.1, the entire training time is shortened by approximately 1.25%. In this paper, only a few GPUs are used for the evaluation due to a facility limitation, but overheads of the aggregation process increase with GPU node count. In this case, there may be a possibility that the proposed switch would further reduce the entire training time in distributed training with more GPUs. Please note that since the aggregation is done by the network switch, the saved compute resources at the host machine can be used for other computation tasks.

5.2.3 Case 3

We evaluate aggregation execution time when using Open MPI in the Jetson Xaviers cluster. This evaluation environment consists of four Jetson Xaviers, one host machine, and the proposed 10GbE switch connecting them. In this evaluation, we measure the execution time on Jetson Xavier from the start of sending gradients data to the receipt of reduce completion responses. We compare the proposed switch-based aggregation with MPI_Reduce() function executed using the host machine. Table 5 shows the execution environments for the worker and host machines.

Figure 13 shows the average execution times of the aggregation for each model. In this graph, red bar and green bar represent the aggregation executed by the proposed switch with one aggregation

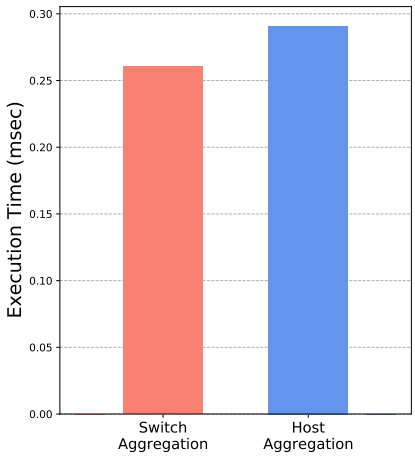


Figure 12: Execution time of gradient aggregation in Case 2

thread and that with four aggregation threads, respectively. Blue bar represents the aggregation executed by the host machine using MPI_Reduce() with a single process. As a result, the aggregation using the proposed switch with four aggregation threads (green bar) outperforms the aggregation using the host machine (blue bar) by 5.5-5.8x. This evaluation result shows that overlapping reduction operations with packets receiving by polling in the network switch helps to significantly reduce the aggregation time. Even when compared with the same number of the aggregation thread (blue bar and red bar), the proposed switch provides 1.6-1.7x faster aggregation than the conventional aggregation. Our proposed switch is based on the DPDK-based software switch that sending and receiving threads poll independently for each NIC port. On the other hand, in the case of the host machine, gradients data from all remote GPUs is received by a single process with interrupt processing for a single port. These differences result in the performance gap between the case of host machine and the proposed switch (1 aggregation thread).

Table 5: Worker and host execution environments in Case 3

	Worker machine	Host machine
OS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS
CPU	ARMv8 Processor rev 0 (v8l) @1.19GHz	Intel Xeon E5-2637 v3 @ 3.50GHz
Memory	32GB	128GB
GPU	512-Core Volta GPU with 64 Tensor Cores	-
pytorch	1.4.0	1.4.0
CUDA	10.0	10.0
Open MPI	2.1.1	2.1.1
NIC	Mellanox ConnectX-3 EN MCX311A-XCAT	Intel Ethernet Converged Network Adapter X520-DA2

5.2.4 Discussion

Our proposed switch is a Linux-based software switch, thus it has only several PCIe slots and several ports per NIC installed to them. In this case, the number of ports that can be scaled with a single proposed switch is at the most 8 or 16. When scaling the number of remote GPUs beyond these ports constraints, more than one level of the proposed switches will be required. However, communication bandwidth between the host machine and the top-level switch would be a bottleneck

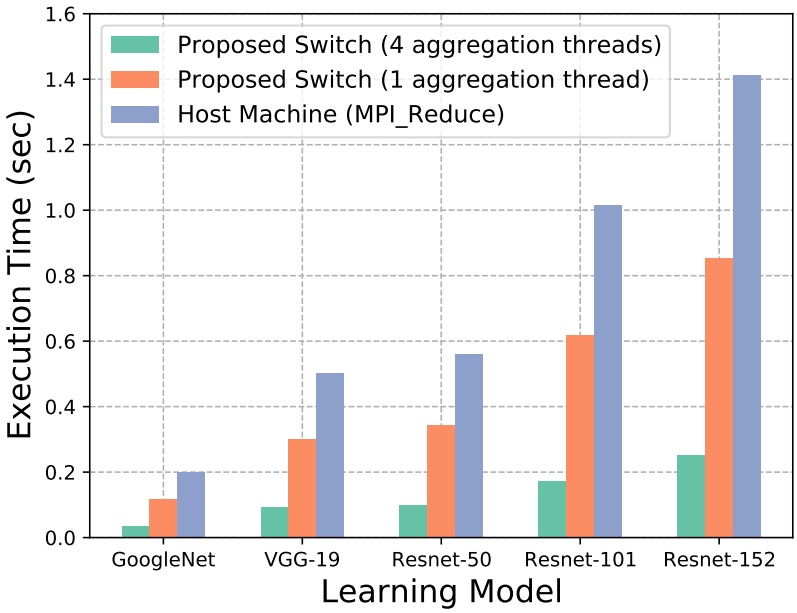


Figure 13: Execution time of gradient aggregation in Case 3

when deep levels of the proposed switches are used. GPU remote extension technologies such as ExpEther basically require a single host machine to control all remote GPUs as well as directly-mounted GPUs. Although the amount of gradients data can be reduced by introducing our proposed switches, communication cost for instructions to each GPU device and mini-batch data increases with the number of remote GPU nodes. Therefore, our proposed approach would have advantage for the relatively small remote GPU cluster consisting of up to a few levels of proposed switches.

6 Summary

In distributed training using remote GPUs, communication occurred between remote nodes when gradients are aggregated. This communication imposes a certain overhead for distributed training with multiple GPUs. In this paper, we focused on remote GPU environments using network-attached GPUs. In these environments, aggregation workload is imposed on a host machine, which lowers aggregation throughput. We therefore proposed to offload the gradient aggregation to a DPDK-based network switch between the host machine and remote GPUs. Our evaluation using UDP communication showed that the proposed switch-based aggregation outperformed the aggregation executed by the host machine by 2.2-2.5x. In the evaluation using ExpEther, although there was room for optimization, aggregation throughput was accelerated by 1.16x with our current implementation. The proposed switch was extended to support MPI over TCP-based communication for exchanging gradients data and evaluated using power efficient Jetson Xaviers cluster in Case 3. This additional evaluation showed that the proposed switch outperformed the conventional reduction operation by up to 5.5-5.8x.

In this paper, we implemented and evaluated our proposed switch by considering the GPU cluster using GTX 1080Ti GPUs and that using embedded GPUs, respectively. Although in this work, we used the limited number of GPUs due to our facility limitation, as a future work we are planning to evaluate scalability of the proposed switch using more GPUs in remote GPU environment.

References

[1] Data Plane Development Kit. https://www.dpdk.org/.

[2] Jetson AGX Xavier. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit/.

[3] Open MPI. https://www.open-mpi.org/.

[4] pktgen-dpdk. http://git.dpdk.org/apps/pktgen-dpdk/.

[5] Martin Abadi et al. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*, pages 265–283, Nov 2016.

[6] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes. *arXiv:1711.04325*, Nov 2017.

[7] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting Distributed Synchronous SGD. *arXiv:1604.00981*, Mar 2017.

[8] José Duato, Antonio J. Peña, Federico Silla, Rafael Mayo, and Enrique S. Quintana-Ortí. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS'10)*, pages 224–231, Jun 2010.

[9] Masaki Furukawa, Tomoya Itsubo, and Hiroki Matsutani. An In-Network Parameter Aggregation using DPDK for Multi-GPU Deep Learning. In *Proceedings of The Eighth International Symposium on Computing and Networking (CANDAR'20)*, Nov 2020.

[10] Richard L. Graham, Devendar Bureddy Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*, Nov 2016.

[11] Tomoya Itsubo, Michihiro Koibuchi, Hideharu Amano, and Hiroki Matsutani. Accelerating Deep Learning using Multiple GPUs and FPGA-Based 10GbE Switch. In *Proceedings of the International Conference on Parallel, Distributed and Network-Based Processing (PDP'20)*, pages 102–109, Mar 2020.

[12] Youjie Li, Jongse Park, Mohammad Alian, Yifan Yuan, Zheng Qu, Peitian Pan, Ren Wang, Alexander Gerhard Schwing, Hadi Esmaeilzadeh, and Nam Sung Kim. A Network-Centric Hardware/Algorithm Co-Design to Accelerate Distributed Training of Deep Neural Networks. In *Proceedings of the International Symposium on Microarchitecture (MICRO'18)*, pages 175–188, Oct 2018.

[13] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arbind Krishnamurthy. Parameter Hub: a Rack-Scale Parameter Server for Distributed Deep Neural Network Training. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'18)*, pages 41–54, May 2018.

[14] Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka, and Yuichi Kageyama. Massively Distributed SGD: ImageNet/ResNet-50 Training in a Flash. *arXiv:1811.05233*, Nov 2018.

[15] Jun Suzuki, Yoichi Hidaka, Junichi Higuchi, Yuki Hayashi, Masaki Kan, and Takashi Yoshikawa. Disaggregation and Sharing of I/O Devices in Cloud Data Centers. *IEEE Transactions on Computers*, 65(10):3013–3026, Oct 2016.

[16] Jun Suzuki, Yoichi Hidaka, Junichi Higuchi, Takashi Yoshikawa, and Atsushi Iwata. ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform. In *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI'06)*, pages 45–51, Aug 2006.