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

Masaki Furukawa, Tomoya Itsubo, and Hiroki Matsutani
Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: {furukawa,itsubo,matutani}@arc.ics.keio.ac.jp

***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 distributed deep learning using the remote GPUs, workload of gradient aggregation is imposed on a host machine. In this paper, we therefore propose to offload the gradient aggregation to a DPDK (Data Plane Development Kit) based network switch between the host machine and remote GPUs. In this approach, the aggregation process is completed in the network using extra computation resources in the network switch. We evaluate the proposed switch when GPUs and the host communicate with a standard IP communication and a PCI Express (PCIe) over 40Gbit Ethernet (40GbE) product, 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. This approach is thus useful for distributed training with multiple GPUs.**

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 [1]. 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 technology, where a host machine and GPUs are connected via 40GbE. 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) [2] 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. We evaluate the switch when GPUs and the host machine communicate with a standard IP communication and ExpEther 40G (a PCIe over 40GbE product) [3], respectively. The evaluation results show that the proposed switch provides a higher throughput compared to aggregation performed on the host machine. This approach can be applied to the performance improvement of distributed training with multiple GPUs.

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 [4]. 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 a 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.

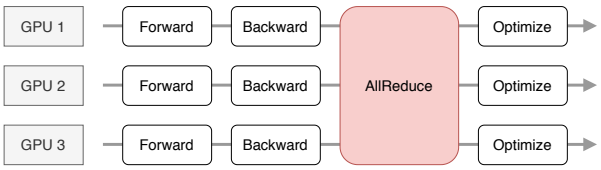

Figure 1. Single iteration in distributed training

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 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 [5], 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 [3] [6] 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 ExpEther 40G product as one case of remote GPU environment in this paper.

rCUDA (remote CUDA). rCUDA [7] is widely used as a software service to use GPUs via a network. 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 the network. Clients request the server machines for a GPU processing via TCP/IP communication. Then, the server machines instead of 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. Furthermore, 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 [8], 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 [9], 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 [10] [11]. In [10], 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 overhead increases as the number of GPUs increases. In [10], 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 [12] is known as a reduce communication offload technology. Reduction operations based on SHArP can be offloaded into Infiniband switches 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-Based Switch Design

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 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, gradient data is aggregated by the CPU on a 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.

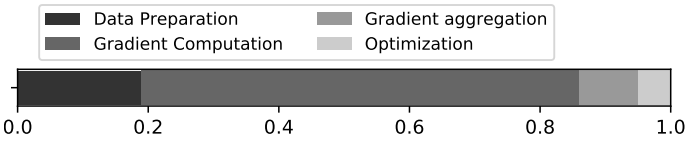

Figure 2. Breakdown of execution times in training phase

As shown in Figure 2, the gradient communication 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 arrives 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.

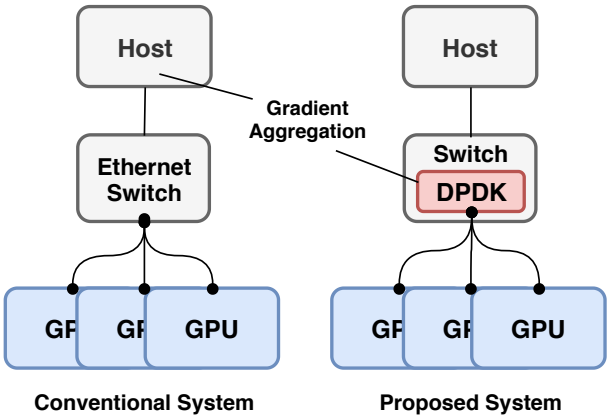

Figure 3. System view. Gradients are computed by GPUs, then transmitted to a host machine. In our proposed system, gradient aggregation is done by DPDK-based switch.

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 gradient data. There are many ways to implement the software network switch. In this paper, we employ DPDK [2], 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 gradient data transmitted from each GPU node passes through the proposed switch. Then, the gradient data is routed and aggregated at the same time by the switch. The proposed switch reduces the gradient data in the network by $1/N$, where N is the number of participating GPUs. By performing the aggregation in the network, the workload on a host machine is significantly mitigated.

4. Implementation

4.1. Remote GPUs Environment Setup

In this paper, we consider the following two 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.

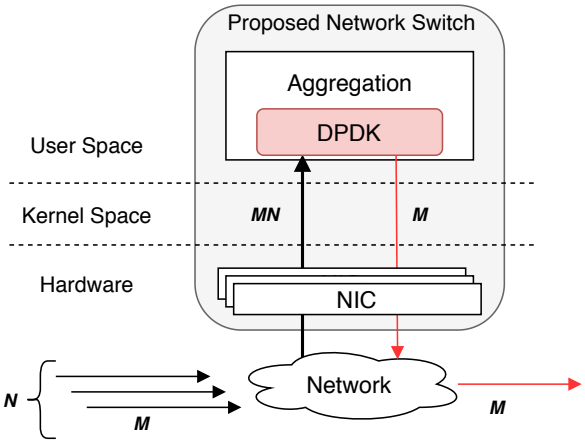

Figure 4. In-network aggregation using DPDK. N: GPU count, M: gradient size, $\rightarrow$: Gradient, $\rightarrow$: Aggregated gradient

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

We employ 40GbE network as an interconnect between a host machine and GPU nodes. We implement the environments using Pytorch as the deep learning framework in both cases.

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 network 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 the gradient by applying the minibatch assigned by the host machine to its own DNN model.
3) The calculated gradient data is sent to the host machine.
4) These gradients are aggregated on the network switch instead of the host machine.
5) The aggregated gradient is multicasted by the switch to each GPU, not to the host machine.
6) Instead of receiving 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 use UDP/IP communication for the gradient data communication, and TCP/IP communication for other communication. Considering the transmission efficiency and the packet buffer size in DPDK switch, the packet length containing gradient data is 1516 bytes, which include additional headers indicating the type of data and the sequence number of gradients.

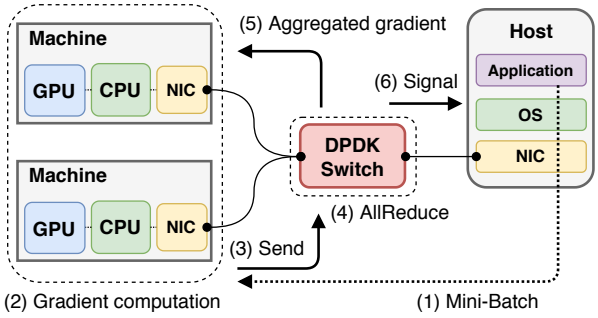

Figure 5. Case 1 using remote invocation of GPU tasks

4.2.2. Case 2. Figure 6 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 7(a) shows the I/O box with one GPU mounted. The host machine and GPUs are connected via 40GbE network using ExpEther bridge. Figure 7(b) shows the 40G host adapter. The PCIe packet containing the calculated gradient data is encapsulated into an Ethernet frame 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, we forward all packets to the host machine when gradients are aggregated on the switch. This processing is done for enabling the trace of packets. The flow from (1) to (4) is the same as that in Case 1.

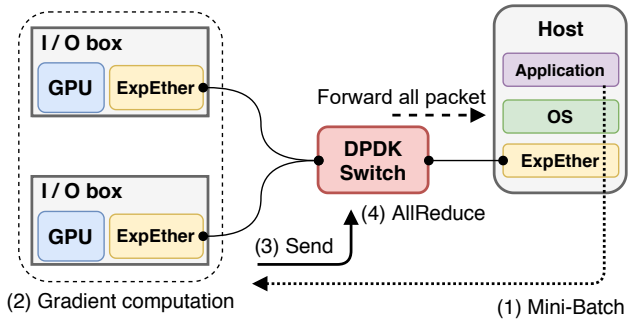

Figure 6. Case 2 using ExpEther

4.3. DPDK-Based Aggregation Switch

Figure 8 illustrates an overview of the implemented DPDK-based aggregation switch. The DPDK application receives packets directly in user space via a polling-based receiving mechanism called PMD (Poll Mode Driver). The received packet is placed in the packet buffer pre-allocated on Hugepage, and the application obtains the packet pointer 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, not regular Linux threads. Receiving threads guarantee the order of the received packets, and

(a) I/O box with one GPU

(b) Host adapter

Figure 7. ExpEther 40G environment. ExpEther I/O boxes with GPUs (a) and host adapter (b) are connected to DPDK switch via 40GbE.

perform packet classification processing using the header. The packet including the gradient data to be aggregated is passed from receiving threads to aggregation threads. The aggregation thread extracts the gradient data from the received packet and temporarily stores the data in the gradient buffer allocated on Hugepage in advance. Assuming that the DNN model size is M and N GPU workers participate, an additional memory MN is required in the network switch. Packets not to be aggregated are forwarded based on the header information and the MAC address table held by the thread of each port.

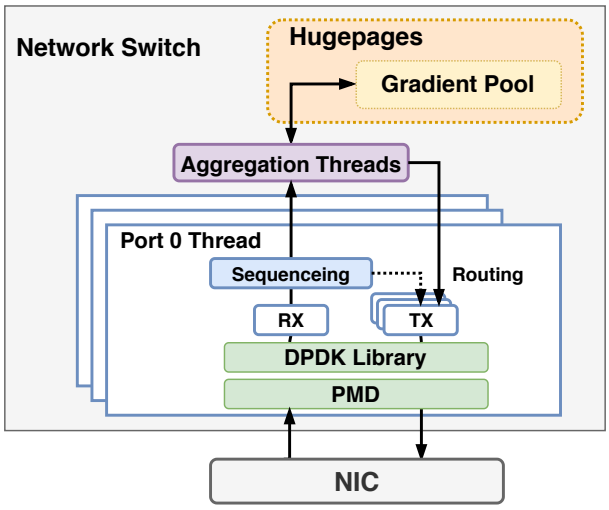

Figure 8. An overview of DPDK-based aggregation switch. A sending / receiving thread is implemented independently for each NIC port.

There are several ways to organize threads to perform aggregation. Figure 9 illustrates the way of gradient aggregation used in this switch for three gradient arrays. 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 packetizes the aggregated gradient data based on MAC address table and requests the corresponding sending / receiving threads to send the packet. In actual environments, there is a delay in the time until gradient arrays are received from all GPU nodes. In particular, for larger gradient data, a longer waiting time occurs until the data is fully received. In this thread organization, data receiving tasks and aggregation operations can be efficiently overlapped with less inter-thread synchronization. Thus, this is a preferable way to organize aggregation threads.

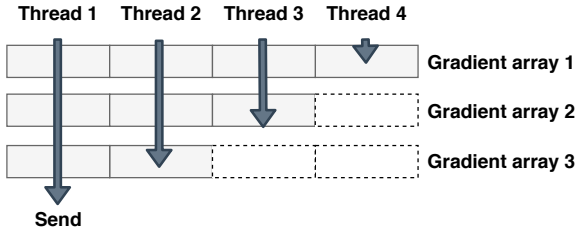

Figure 9. A way of gradient aggregation. Each gradient array is divided into chunks. Each aggregation thread works on the same chunk in all gradient arrays.

5. Evaluations

5.1. Packet Processing Throughput

First, the proposed DPDK-based network switch is evaluated in terms of the 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 machines. Test packets including gradients data are generated at a line rate of 40Gbps by using DPDK-Pktgen [13] and sent continuously from the packet generator machine to the proposed switch. The aggregation 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 DPDK 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. When ExpEther is used (Case 2), each packet contains 32 gradients 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, it is considered that the

network bandwidth is limited by the proposed switch when ExpEther is used in our current implementation. 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. 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	ver 18.11.2
Pktgen	ver 19.10.0

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 measurement, the execution time from when the packet including gradient 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 2 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. Table 3 shows the execution environments for the worker and host machines.

TABLE 2. 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 10 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.

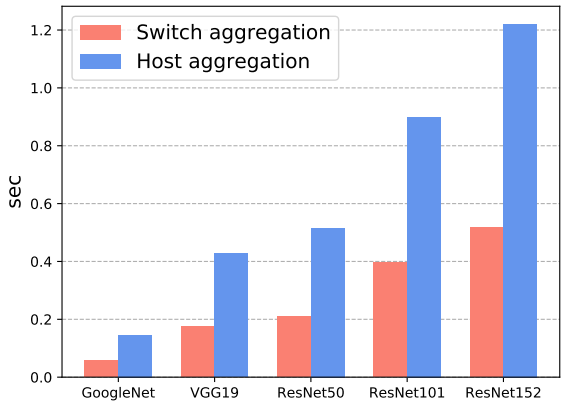

Figure 10. Execution time of the aggregation in Case 1

TABLE 3. WORKER AND HOST EXECUTION ENVIRONMENTS

	Worker machine	Host machine
OS	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	ver 10.0	
Pytorch	ver 1.3.0	
NIC	Intel Ethernet CNA XL710-QDA2	

5.2.2. Case 2. We evaluate the 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 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 gradient 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 11 shows the comparison of the aggregation execution time. Note that this time does not include processing for the aggregated gradient array. 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 aggregation throughput 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

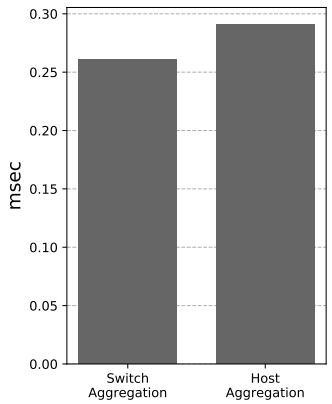

Figure 11. Execution time of the aggregation in Case 2

a few GPUs are used for the evaluation, but more GPUs are generally used. Overheads of the aggregation process increase with GPU node count. Therefore, 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.

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 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 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.

As a future work, we are evaluating scalability of the proposed switch using more GPUs in remote GPU environment. We are also planning to demonstrate the performance improvement of entire training using low-cost GPU clusters composed of power-efficient embedded GPUs such as NVIDIA Jetson series.

References

[1] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. 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)*, May 2018, pp. 41–54.

[2] "Data Plane Development Kit," https://www.dpdk.org/.

[3] J. Suzuki, Y. Hidaka, J. Higuchi, Y. Hayashi, M. Kan, and T. Yoshikawa, "Disaggregation and Sharing of I/O Devices in Cloud Data Centers," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 3013–3026, Oct 2016.

[4] M. 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)*, Nov 2016, pp. 265–283.

[5] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting Distributed Synchronous SGD," *arXiv:1604.00981*, Mar 2017.

[6] J. Suzuki, Y. Hidaka, J. Higuchi, T. Yoshikawa, and A. Iwata, "ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform," in *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI'06)*, Aug 2006, pp. 45–51.

[7] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. 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)*, Jun 2010, pp. 224–231.

[8] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. G. Schwing, H. Esmaeilzadeh, and N. S. 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)*, Oct 2018, pp. 175–188.

[9] T. Itsubo, M. Koibuchi, H. Amano, and H. 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)*, Mar 2020, pp. 102–109.

[10] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes," *arXiv:1711.04325*, Nov 2017.

[11] H. Mikami, H. Suganuma, P. U-chupala, Y. Tanaka, and Y. Kageyama, "Massively Distributed SGD: ImageNet/ResNet-50 Training in a Flash," *arXiv:1811.05233*, Nov 2018.

[12] R. L. Graham, D. B. P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, L. Levi, A. Margolin, T. Ronen, A. Shpiner, O. Wertheim, and E. 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.

[13] "pktgen-dpdk," http://git.dpdk.org/apps/pktgen-dpdk/.