Exhaustive Study of Hierarchical AllReduce Patterns for Large Messages Between GPUs

Conference Paper · May 2019

DOI: 10.1109/CCGRID.2019.00057

2 authors, including:

Rio Yokota
Tokyo Institute of Technology
80 PUBLICATIONS **1,027** CITATIONS

Exhaustive Study of Hierarchical AllReduce Patterns for Large Messages Between GPUs

Yuichiro Ueno
Department of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
ueno.y.ai@m.titech.ac.jp

Rio Yokota
Global Scientific Information and Computing Center
Tokyo Institute of Technology
Tokyo, Japan
rioyokota@gsic.titech.ac.jp

***Abstract*—Data-parallel distributed deep learning requires an AllReduce operation between all GPUs with message sizes in the order of hundreds of megabytes. The popular implementation of AllReduce for deep learning is the Ring-AllReduce, but this method suffers from latency when using thousands of GPUs. There have been efforts to reduce this latency by combining the ring with more latency-optimal hierarchical methods. In the present work, we consider these hierarchical communication methods as a general hierarchical Ring-AllReduce with a pure Ring-AllReduce on one end and Rabenseifner's algorithm on the other end of the spectrum. We exhaustively test the various combinations of hierarchical partitioning of processes on the ABCI system in Japan on up to 2048 GPUs. We develop a performance model for this generalized hierarchical Ring-AllReduce and show the lower-bound of the effective bandwidth achievable for the hierarchical NCCL communication on thousands of GPUs. Our measurements agree well with our performance model. We also find that the optimal large-scale process hierarchy contains the optimal small-scale process hierarchy so the search space for the optimal communication will be reduced.**

***Index Terms*—Hierarchical, AllReduce, Large Message, GPU, InfiniBand, NVLink, NCCL, Deep Learning**

I. Introduction

Deep learning allows very large models to be trained on very large datasets, so the computation time can quickly become intractable. Furthermore, many training runs need to be performed to find the optimal hyperparameters such as the learning rate, momentum, drop out, and weight decay. Distributed parallel computing is one solution to this large demand in computational power. The simplest approach to harness the power of a large cluster of computers is to run the hyperparameter search in parallel. However, being able to run a single training job on multiple nodes/GPUs could allow faster convergence to an optimum set of hyperparameters due to better feedback during the search [1].

Large-scale parallel deep learning is performed on the largest supercomputers today. For example, the winners of the ACM Gordon Bell Prize for 2018 used more than 25,000 NVIDIA Tesla V100 GPUs to train a deep neural network to improve the predictive capability of climate models [2]. Other finalists of the prize for 2018 also used deep learning for extracting data from electron microscopes [3].

A popular benchmark for large-scale distributed deep learning is the training of ResNet-50 on the ImageNet-1K dataset on thousands of processes. Goyal *et al.* achieved a top-1 validation accuracy of 76.3% in 1 hour using 256 P100 GPUs [4]. Akiba *et al.* achieved 74.9% in 15 minutes using 1,024 P100 GPUs [5]. Jia *et al.* achieved 75.8% in 6.6 minutes using 2,048 P40 GPUs [6]. Mikami *et al.* achieved 75% in 3.7 minutes using 2,176 V100 GPUs [7]. Ying *et al.* achieved 76.3% in 2.2 minutes using 1,024 TPUs [8].

Training of deep neural networks on distributed systems can be done either by distributing the data and replicating the model or distributing the model and replicating the data among the distributed processes. The former is called data-parallel and typically requires a course grain AllReduce collective communication of the gradients among all processes. The latter is called model-parallel and requires a fine grain point-to-point communication of the activation/gradients between layers. Data-parallelism has been the more popular approach due to its scalability and ease of implementation. In this work, we will focus on the collective communication used in the data-parallel approach. Even though the large volume of computation typically hides the latency in the communication of deep learning, it has been reported that communication could become a bottleneck past 1,024 GPUs [2].

As the number of processes participating in the collective communication grow, the optimal algorithm changes. For example, using the Ring-AllReduce on thousands of processes is suboptimal, although it is known to be faster for sending large data to a small number of processes. Combining the collectives of the NVIDIA Collective Communications Library (NCCL) [9] and MPI is known to improve performance [10]. Furthermore, using the collectives of NCCL two-dimensionally also works well [7]. In this way, grouping rings into a few hierarchical collective operations seems to give a better performance, but the optimal dimensions of this hierarchical communication depend on the network topology, the number of GPUs per node, the bandwidth within the node and between the nodes, the number of processes, and the message size. We aim to provide a strategy for choosing the optimal hierarchical communication for deep learning workloads.

Computational resource of AI Bridging Cloud Infrastructure (ABCI) was awarded by "ABCI Grand Challenge" Program, National Institute of Advanced Industrial Science and Technology (AIST). This work is supported by JST CREST Grant Number JPMJCR1687, Japan.

Our contributions are as follows:

- We consider the hierarchical AllReduce communication as a hybrid method with the Ring-AllReduce at one end and Rabenseifner's Algorithm at the other end. We show that both of these algorithms at the far end are suboptimal for typical deep learning workloads.
- We develop a model that predicts the performance of the hierarchical AllReduce by the number of dimensions, the number of processes and the message size and verify its accuracy on InfiniBand-connected multi-GPU per node supercomputers.
- We implement the hierarchical AllReduce on both NVLink and InfiniBand using NCCL for the elementary communication at every level of the hierarchical AllReduce. We test this for a wide configuration of hierarchical topologies and the number of processes to show the effect of the low-latency of NCCL.
- Through performance modeling and systematic experimentation, we determine a strategy for choosing the optimal hierarchical AllReduce for any network topology, number of GPUs per node, bandwidth within the node and between the nodes, number of processes, and message size.

II. Distributed Deep Learning

In this section, we describe the general properties of the communication and parallelization of deep neural network training. For a thorough review of this topic, we refer the reader to [11].

A. Parallelism of Deep Learning

Distributed parallel computers can accelerate deep learning by exploiting the two types of parallelism available. One is to parallelize over the model and the other is to parallelize over the data. In this work, we focus on the data-parallelism.

In data-parallel distributed deep learning, the data is distributed and the parameters are replicated. Multiple processes work on a different subset of the entire data set, and aggregate this information at the end of each iteration to update the shared parameters among the processes. Instead of speeding up the time it takes to process a single data, it processes multiple data in a given amount of time.

B. Distributed Stochastic Gradient Descent (SGD)

In data-parallel distributed deep learning, different processes use different training data, and the different gradients that result from this are used to update the parameters. There are two well-known updating methods: the asynchronous method (Asynchronous SGD) and the synchronous method (Synchronous SGD).

1) Asynchoronous SGD: Asynchronous SGD can have a number of different implementations. A typical one would be to use parameter servers, which are servers dedicated to keeping track of the parameters. The worker processes, in charge of the training, calculate the forward propagation and backward propagation and sent the resulting gradients to the parameter server. The parameter servers will then update the parameters according to the received gradient. The parameter server then sends the updated parameters to the worker process. When the worker process receives the updated parameters, the worker uses them to calculate the forward and backward propagation.

2) Synchronous SGD: Synchronous SGD typically does not use parameter servers. The dataset is distributed to all processes and they all serve as workers, which do forward and backward propagation on the distributed datasets. Then, at the end of each iteration, an AllReduce collective communication is performed to calculate the average of the gradients among all processes. This gives the equivalent result to using a large mini-batch on a single process, but the calculation will be faster because the data is actually processed in parallel. Under the assumption that the parameters on all processes are initialized with the same value, this method allows the model to remain consistent between the processes without sending the parameters. One drawback of this approach is that the effective mini-batch could become exceedingly large, which is known to have a detrimental effect on the accuracy [12].

3) Characteristic of Communication in Synchronous SGD: In the current work, we focus on Synchronous SGD. As mentioned earlier, this requires us to perform an AllReduce collective communication between all processes at every iteration. When the model size increases, the message size of this AllReduce communication also increases. One of the well-known convolutional neural networks is ResNet-50, and this requires around 100MB communication in each iteration. Many AllReduce implementations switch between different algorithms depending on the number of processes and message size. However, the range of message size they consider is usually only until 8MB [13], which is much smaller than the typical message size in deep learning communications. This motivates us to conduct a more in-depth investigation of optimal AllReduce operations for deep learning workloads.

III. AllReduce Operation

In this section, we discuss the details of the AllReduce collective communication and its optimal implementation for deep learning. An AllReduce communication performs a reduction operation on each element of an array on all processes and the result of the reduction is stored on all processes.

A. Algorithms of AllReduce

There exist many algorithms for the AllReduce, but we will focus on two of them that have optimal bandwidth requirements; the Ring-AllReduce Algorithm [14] and Rabenseifner's Algorithm [13] in the present work.

1) The Ring-AllReduce Algorithm: The Ring-AllReduce algorithm forms a logical ring among all the processes involved in the communication. Each process only communicates with the two adjacent processes in the logical ring topology. If the total number of processes is p, the array to be reduced is partitioned into p chunks. When the i-th process sends the i-th chunk to the next $(i+1)$-th process, the same i-th process will receive the $(i-1)$-th chunk from the $(i-1)$-th process.

After receiving the chunk it will perform a reduction with its own corresponding chunk, and then pass it on to the next $(i+1)$-th process. When $i = p-1$, $i+1 = p$ will wrap around to 0, and when $i = 0$, $i-1 = -1$ will wrap around to $p-1$. When this step is done $p-1$ times, every process will have a different chunk that has the contribution of all processes. At this stage, the Reduce-Scatter operation is complete. In order to complete the AllReduce operation, one must now perform an AllGather operation, which can be done in a similar manner using the ring in $p-1$ steps. Therefore, the total number of communications is $2(p-1)$. If the total message size is n, each chunk size will be n/p, so the total volume of communication will be $2(p-1)n/p$.

2) Rabenseifner's Algorithm: Let us suppose the total number of processes is $p = 2^k$. Rabenseifner's Algorithm performs the AllReduce by communicating with processes that have a distance of $1, 2, 4, \cdots, 2^{k-1}$, therefore requiring only $O(\log p)$ steps. In the first step, each process exchanges data with the neighboring processes. During this exchange, it splits the array into two halves; and performs a Reduce-Scatter similar to a ring with two processes. In the next step, pairs are formed among the processes that are two apart. The array that was split during the first step is further split again, and two halves are exchanged in a similar fashion. This kind of recursive halving for the Reduce-Scatter can be combined with a recursive doubling for the AllGather to compose an AllReduce. The Reduce-Scatter is done with pairs of processes with a distance of $1, 2, 4, \cdots, 2^{k-1}$, whereas the AllGather does the reverse with pairs having a distance of $2^{k-1}, \cdots, 4, 2, 1$. The total number of exchanges for this case is $2k$. The total volume of communication is $2n(\sum_{i=1}^{k} 2^{-i}) = 2(p-1)n/p$.

3) Pipelining of AllReduce: Both the Ring-AllReduce and Rabenseifner's algorithm first perform a Reduce-Scatter, which consists of receiving data, performing the reduction on it, and sending it to the next process. The sending of one chunk of data can be overlapped with the reduction of the received chunk, so the cost of the reduction can be entirely hidden. Because of the need to partition the chunks again to overlap the computation with communication, the number of communications increases and the latency does as well.

4) Comparison AllReduce Algorithms: Let the total number of processes be $p = 2^k$ and let the length of the array be n. Then, the total amount of communication per process is $2(p-1)n/p$ for both the Ring-AllReduce and Rabenseifner's Algorithm. The main difference is that the Ring-AllReduce only communicates between adjacent processes but it does so $2(p-1)$ times, whereas Rabenseifner's Algorithm communicates with processes that are exponentially far but only for $2\log_2 p$ times. Therefore, the Ring-AllReduce can avoid contention on most network topologies [14]. We can use a very simple performance model using the latency α and inverse bandwidth β of the network, where the point-to-point communication time T can be written as [13]:

$$T(n) = \alpha + n\beta$$

Under this model, Rabenseifner's algorithm is always faster than the Ring-AllReduce, since the volume of communication is same but the latency of Rabenseifner's Algorithm is smaller.

B. Implementations of AllReduce

NVIDIA Collective Communications Library (NCCL) [9] and CUDA-aware MPI [15] are the two most popular AllReduce implementations between GPUs.

1) NVIDIA Collective Communications Library: NCCL is designed for communication between GPUs in a multi-GPU environment, where the GPUs are connected through NVLink or PCI-Express internally, and are connected through InfiniBand or Ethernet externally. The AllReduce collective of NCCL is implemented using the Ring-AllReduce algorithm. NCCL can detect the NVLink and InfiniBand connections, and form multiple ring topologies to extract the full bandwidth of the fabric automatically. Therefore, NCCL is used as a *de facto* standard library for distributed deep learning.

2) CUDA-aware MPI: CUDA-aware MPI is an extension of MPI, where it can handle the address of GPU memory space. The user does not need to explicitly call the memory copies between host and device, so it is easy to use. CUDA-aware MPI can use GPUDirect RDMA, which enables DMA between GPU and PCI-Express devices (*e.g.* InfiniBand HCA). MPI has many different implementations for the AllReduce so that the optimal one can be selected for a given number of processes and message size. There has been some research to bring its performance closer to that of NCCL [16].

IV. Hierarchical AllReduce

In this section, we describe a hierarchical AllReduce algorithm, which combines the best of both worlds from Rabenseifner's Algorithm and the Ring-AllReduce Algorithm.

A. Hierarchical Characteristic of Rabenseifner's Algorithm

Rabenseifner's Algorithm can be thought of as a hierarchical version of the Ring-AllReduce Algorithm, where the Ring-AllReduce is performed on pairs of two recursively. When the number of processes is two, the two algorithms are identical.

Let us now consider the case of four processes, where each of them have rank = $\{0, 1, 2, 3\}$. In this case, Rabenseifner's Algorithm will perform two steps of Reduce-Scatter and two steps of AllGather, with a total of four steps to perform an AllReduce. The first step of the Reduce-Scatter form the pairs (pair-1: $\{0, 1\}$, pair-2: $\{2, 3\}$). After the pairs exchange the first half and second half of the buffer, rank $\{0, 2\}$ will receive the first half of the reduced array, while rank $\{1, 3\}$ will receive the second half of the reduced array. Next, we perform the reduction between pairs, by doing the same operation for pairs (pair-1: $\{0, 2\}$, pair-2: $\{1, 3\}$). This will result in rank = $\{0, 2, 1, 3\}$ having the total reduced values the first, second, third, and fourth chunk, respectively. Then, performing AllGather with the pairing in the opposite order will result in an efficient AllGather algorithm. The second Reduce-Scatter step of Rabenseifner's Algorithm for four processes, is identical to the Reduce-Scatter step of Rabenseifner's

Algorithm for two processes, which means it is also identical to a Ring-Reduce-Scatter for two processes. To summarize, Rabenseifner's Algorithm for four processes will perform the below operations:

1) a Ring-Reduce-Scatter for pairs: $\{0,1\},\{2,3\}$
2) a Ring-Reduce-Scatter for pairs: $\{0,2\},\{1,3\}$
3) a Ring-AllGather for pairs: $\{0,2\},\{1,3\}$
4) a Ring-AllGather for pairs: $\{0,1\},\{2,3\}$

B. Extension to the Hierarchical Ring-AllReduce Algorithm

Rabenseifner's Algorithm can be thought of as a hierarchical Ring-AllReduce Algorithm that performs the Ring-AllReduce on pairs of two in a multi-layer fashion. A generalization of this is to have the rings at each layer have more than two processes. This allows a solution to the following issues:

1) From the perspective of Rabenseifner's Algorithm, it can alleviate the contention that would otherwise occur in an original Rabenseifner's Algorithm for networks topologies that don't have a full bisection bandwidth. Also, the distance between pairs in Rabenseifner's Algorithm grows exponentially, but we can also reduce this exponential growth by using more processes per layer.
2) From the viewpoint of a Ring-AllReduce Algorithm, it can reduce the number of communications and therefore reduce the latency. The hierarchy keeps the ring at any given layer from becoming too long and can reduce the number of communications within a ring.

A schematic of the hierarchical Ring-AllReduce on 128 processes with $4\times8\times4$ hierarchical configuration and the 5 steps of communication are shown in Fig. 1. In this figure, we assume that each node has four GPUs. Step 1 is the intra-node Reduce-Scatter and Step 2 is the inter-node Reduce-Scatter where each GPU in the node participates in a separate Ring-Reduce-Scatter between eight processes. The message size of Step 2 is $1/4$ of Step 1, since it uses the result of the Reduce-Scatter at Step 1. Step 3 further performs an AllReduce between 4 processes, where the message size is now $1/4\times1/8 = 1/32$ of Step 1. Steps 4 and 5 perform an AllGather between the same processes as the Reduce-Scatter in Steps 2 and 1, respectively.

C. Analysis of the Hierarchical Ring-AllReduce Algorithm

In this section, we model the communication time T of the hierarchical Ring-AllReduce algorithm using $T(n) = \alpha + n\beta$ [13] where n is the message size. Let h be the number of hierarchical layers, and p be the total number of processes. The number of processes per layer can be multiplied to yield the total number of processes

$$p = p_1 \times p_2 \times \cdots \times p_h.$$

We consider the difference in the utilization of the bandwidth at different layers and we note the inverse bandwidth at the i-th layer as β_i. Then, the point-to-point communication time T_i for a message size n at that i-th layer is

$$T_i(n) = \alpha + n\beta_i.$$

By assuming that the time for the reduction can be hidden by partitioning the message to $p\times s$ parts, the total communication time $T_{i,all}$ for the Ring-AllReduce of message size n with pipelining at that i-th layer would be

$$T_{i,all}(n) = 2s\alpha(p_i - 1) + 2\beta_i\frac{n}{p_i}(p_i - 1).$$

The total communication time for all layers can then be calculated as

$$\begin{aligned} T(n) &= \sum_{i=1}^{h} T_{i,all}\left(\frac{n}{\prod_{j=1}^{i-1} p_j}\right) \\ &= 2s\alpha\left(\sum_{i=1}^{h}(p_i) - h\right) + 2n\sum_{i=1}^{h}\beta_i\left(\frac{p_i - 1}{\prod_{j=1}^{i} p_j}\right). \end{aligned}$$

Let us consider the following decomposition of processes:

$$p = \left(p_{a_1}\times\cdots\times p_{b_1}\right)\times\left(p_{a_2}\times\cdots\times p_{b_2}\right)\times\cdots\times\left(p_{a_m}\times\cdots\times p_{b_m}\right).$$

We assume that for a given layer with processes $\left(p_{a_i}\times\cdots\times p_{b_i}\right)$, the inverse bandwidth $\beta = B_i$ is constant within that layer. Under these conditions, the interconnect is decomposed into hierarchical layers, and the bandwidth may differ between the layers. (*e.g.* Inter-node and intra-node communication could be considered as different layers in the hierarchical process decomposition).

The total communication time of the hierarchical communication can be written as

$$\begin{aligned} T(n) &= 2s\alpha\left(\sum_{i=1}^{h}(p_i) - h\right) + 2n\sum_{i=1}^{m} B_i \sum_{j=a_i}^{b_i}\left(\frac{p_j - 1}{\prod_{k=1}^{j} p_k}\right) \\ &= 2s\alpha\left(\sum_{i=1}^{h}(p_i) - h\right) + 2n\sum_{i=1}^{m} B_i\left(\frac{1}{\prod_{k=1}^{b_{i-1}} p_k} - \frac{1}{\prod_{k=1}^{b_i} p_k}\right). \end{aligned}$$

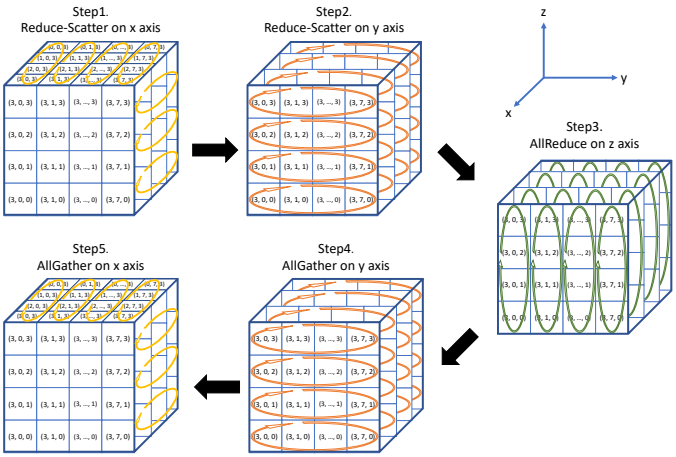

Fig. 1. A schematic of the hierarchical Ring-AllReduce on 128 processes with $4\times8\times4$ configuration.

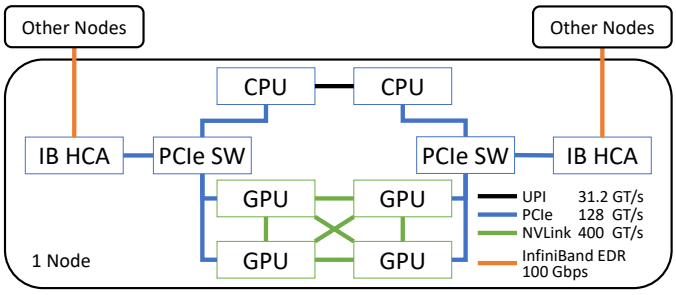

Fig. 2. A schematic of the ABCI supercomputer node configuration.

Since $\prod_{k=1}^{b_0} p_k = 1$, the total communication time can be rewritten as

$$T(n) = 2s\alpha\left(\sum_{i=1}^{h}(p_i) - h\right) + 2n\sum_{i=1}^{m} B_i\left(\frac{1}{\prod_{k=1}^{i-1} P_k} - \frac{1}{\prod_{k=1}^{i} P_k}\right)$$
$$= 2s\alpha\left(\sum_{i=1}^{h}(p_i) - h\right) + 2n\sum_{i=1}^{m} B_i\left(\frac{P_i - 1}{\prod_{k=1}^{i} P_k}\right).$$

where $\left(p_{a_i} \times \cdots \times p_{b_i}\right) = P_i$ and $\prod_{k=1}^{0} P_k = 1$. This is the same as the hierarchical Ring-AllReduce with m layers, except for the difference in the α term. This shows the hierarchical decomposition in the layers of same bandwidth does not affect the bandwidth term in the communication time. The equivalence of the bandwidth term in the Ring-AllReduce and Rabenseifner's Algorithm comes from this.

D. Hierarchical Ring-AllReduce Algorithm on a GPU-accelerated InfiniBand Cluster

A schematic of the node configuration for the ABCI supercomputer at AIST is shown in Fig. 2. Each node has two CPUs, four NVIDIA Tesla V100 GPUs, and two InfiniBand EDR HCAs. The GPUs are connected via NVIDIA NVLink2 for high-bandwidth and low-latency communication.

Training a deep neural network on this kind of node configuration will use each GPU for the forward and backward propagation with an AllReduce between GPUs. We will consider the effect of using the hierarchical AllReduce algorithm for such cases.

Let us assume that the AllReduce happens on P nodes and $4 \times P$ processes. We consider the case where there are two layers in the communication hierarchy – NVLink2 and InfiniBand EDR. In this case, the total communication time for the AllReduce is

$$T_{hier}(n) = 2s\alpha(2+P) + 2n\left(B_{nvlink}\left(\frac{3}{4}\right) + B_{IB/2}\left(\frac{P-1}{4P}\right)\right).$$

where B_{nvlink} is the inverse bandwidth of NVLink2, and $B_{IB/2}$ is the inverse bandwidth of half the InfiniBand EDR because four GPUs share two InfiniBand EDR HCAs.

The theoretical bandwidth of NVLink2 is almost four times that of InfiniBand EDR. There are six possible ring orderings for the GPUs, and two patterns will share one link, so intra-node communication is $4 \times 6/2 = 12$ times faster than inter-node communication. Therefore, we can rewrite the equation as:

$$T_{hier}(n) = 2s\alpha(2+P) + 2n\left(\frac{B_{IB}}{12}\left(\frac{3}{4}\right) + 2 \times B_{IB}\left(\frac{P-1}{4P}\right)\right)$$
$$= 2s\alpha(2+P) + 2nB_{IB}\left(\frac{9}{16} - \frac{1}{2P}\right). \quad (1)$$

A flat AllReduce without any hierarchy does not require all GPUs to use InfiniBand, so the bandwidth improves to

$$T_{ring}(n) = 2s\alpha(4P-1) + 2nB_{IB\times 2}\left(\frac{4P-1}{4P}\right)$$
$$= 2s\alpha(4P-1) + 2nB_{IB}\left(\frac{1}{2} - \frac{1}{8P}\right). \quad (2)$$

Therefore, it can be seen that the hierarchical AllReduce improves the α (latency) term, but for the B (inverse bandwidth) term Ring-AllReduce is smaller for $P > 6$, so there is a trade-off.

V. Experiment

To evaluate the performance of our hierarchical Ring-AllReduce algorithm for deep learning workloads, we perform two experiments; a microbenchmark and actual training of image classification, on the ABCI supercomputer at AIST.

Fig. 2 shows a schematic of a node on ABCI. Each rack has 34 nodes, where the intra-rack network is full-bisection, while the inter-rack has 1/3 of the full-bisection.

A. Implementation

Our implementation of the hierarchical Ring-AllReduce uses Reduce-Scatter, AllGather, and AllReduce (for innermost layer) collectives of NCCL, which is optimized for NVIDIA GPUs, NVIDIA NVLink, and InfiniBand. The version of compilers and libraries we used are shown in Table I.

B. Microbenchmark result

We use 128MiB single-precision floating point numbers per process, which is similar to the one used in ResNet-50 training, and we change the number of processes from 4 to 2048, and its hierarchical decomposition. We assume that the deep learning framework allocates input and output arrays separately. Furthermore, we allow the input array to be overwritten to store intermediate results. If the input array needs to be preserved, the communication at each layer must copy the output array back to the input array.

TABLE I
Software used in the experiment.

Software	Version
GCC	4.8.5
CUDA	9.2
cuDNN	7.4
Open MPI	2.1
NCCL	2.3.5-2
Chainer	v6.0.0b3

1) Determining the Parameters of the Model: In order to estimate the communication time, we determine the parameters, the latency α, the inverse of bandwidth β and the granularity of pipelining s. α and β are measured by *osu_latency* [17] with GPUDirect RDMA-enabled Open MPI. s is determined from the implementation of NCCL. As a result, we found that the values are $\alpha = 4.80[\mu s]$, $B_{IB}^{-1} = 9301.19[MB/s]$ and $s = 2$.

2) Scalability with Respect to Processes: The communication time of the Ring-AllReduce with and without hierarchy is shown in Fig. 3. The results show the best case for the different number of layers in the hierarchy. To keep the communication topology symmetric, we use only 32 out of the 34 nodes per rack.

a) Evaluation of the Performance Model: Fig. 3 shows that the results with the performance model match the communication time, except for the flat Ring-AllReduce results of processes 8 to 256. In this case, the communication time is slower than the estimation time. This means that there is room for improvement in NCCL's ability to detect the network topology, in particular on the ABCI which has two InfiniBand HCAs on each node. Setting the environment values correctly could improve this problem.

b) Performance of the Flat Ring-AllReduce: The four processes case is a special case since it can use the high-bandwidth performance of NVLink2, without depending on the bandwidth of InfiniBand. Furthermore, when the number of processes is bigger than 256, the communication time is proportional to the number of processes. The proportional term of the performance model (2) is the latency term therefore this slowdown suffers from the latency issues of many processes.

c) Performance of the Hierarchical Ring-AllReduce: From Fig. 3, we see that the flat Ring-AllReduce case is bandwidth-bound for processes 8 to 128, so if both InfiniBand HCAs on the node are utilized, its performance could potentially exceed that of the hierarchical Ring-AllReduce. However, beyond 256 processes the latency causes the performance of the flat Ring-AllReduce to degrade. On the other hand, the hierarchical communication does not suffer from latency issues as can be seen from both the performance model and the actual measurements. Therefore, we show that the hierarchical communication is an appropriate large-message AllReduce algorithm for many GPU processes.

3) Intra-rack Performance of the Hierarchical Ring-AllReduce: The performance of different hierarchical topologies on up to 128 processes (a rack) is shown in Fig. 4. Table II shows the effective bandwidth for the corresponding runs. The fastest hierarchical topology was $4 \times 8 \times 4$, which was 1.39 times faster than the flat Ring-AllReduce.

a) The optimal number of first layer processes: Cases that the number of first layer processes is four are faster than the others due to the high-bandwidth and low-latency performance of NVLink2. When the number of first layer processes is two, the communication is executed in NVLink2 only, but it is slower than cases whose number is four. This reason is NVLink2 has a fully-connected topology so if the number is two, we cannot exploit all links of the topology. When the number of first layer processes is 8 or 128, the speed of the constructed ring is limited by InfiniBand. In these cases, two InfiniBand HCAs are available but due to the topology detection issue of NCCL, the performance is likely underestimated.

b) The Performance of Deeper Hierarchy: From Fig. 4, the second bar from the right is Rabenseifner's Algorithm, which is the almost slowest case. There are two probable causes:

1) Due to the deeper hierarchy.
   The latency of multiple kernel launches, and also the creation of communication threads per NCCL communicator.
2) Due to the smaller message sizes in the deeper layer.
   The message sizes of the deeper layer are reduced by the shallower layers.

The overhead by the first cause is likely to be small, because the communication time of the second layer is almost equal when the number of hierarchy differs (*e.g.* $4 \times 4 \times 8$ and $4 \times 4 \times 4 \times 2$).

Fig. 5 shows the attainable bandwidth for pure NCCL communication when the number of processes and message size are varied. To isolate potential network problems, we used only one GPU per node and only one ring for the communication (NCCL_MAX_NRINGS=1). First of all, as the message size of the AllReduce decreases the performance decreases for all number of processes. This indicates that NCCL currently is not yet optimized for workloads like Rabenseifner's Algorithm, where the message size decreases exponentially at each stage. This is why Rabenseifner's Algorithm has the worst performance, though it is algorithmically superior.

4) Inter-rack Performance of the Hierarchical Ring-AllReduce: The results for the experiments on more than one rack (over 128 processes) are shown in Fig. 6. ABCI has full-bisection within the rank, but $1/3$ of the full-bisection across racks.

TABLE II
BUS BANDWIDTH OF INTRA-RACK HIERARCHICAL ALLREDUCE. (NP = 128)

Topology	Bus Bandwidth [MB/s]			
	1st dim	2nd dim	3rd dim	4th dim
128	11547.97			
4×32	83446.91	4348.85		
$4 \times 4 \times 8$	85237.64	4229.91	4121.85	
$4 \times 8 \times 4$	85956.47	4648.48	3766.94	
$4 \times 16 \times 2$	81777.71	4140.13	2375.11	
$4 \times 4 \times 4 \times 2$	81196.07	4188.34	3417.55	3285.54
$4 \times 8 \times 2 \times 2$	83191.72	4640.75	2705.13	3063.70
8×16	11343.85	4463.43		
$8 \times 4 \times 4$	11239.52	4551.33	4011.04	
$8 \times 4 \times 2 \times 2$	11362.74	4366.68	2978.18	3195.68
Rabenseifner	30279.93	27978.80	3314.66	2998.27
4-Rabenseifner	92155.60	3330.65	3060.10	3014.57

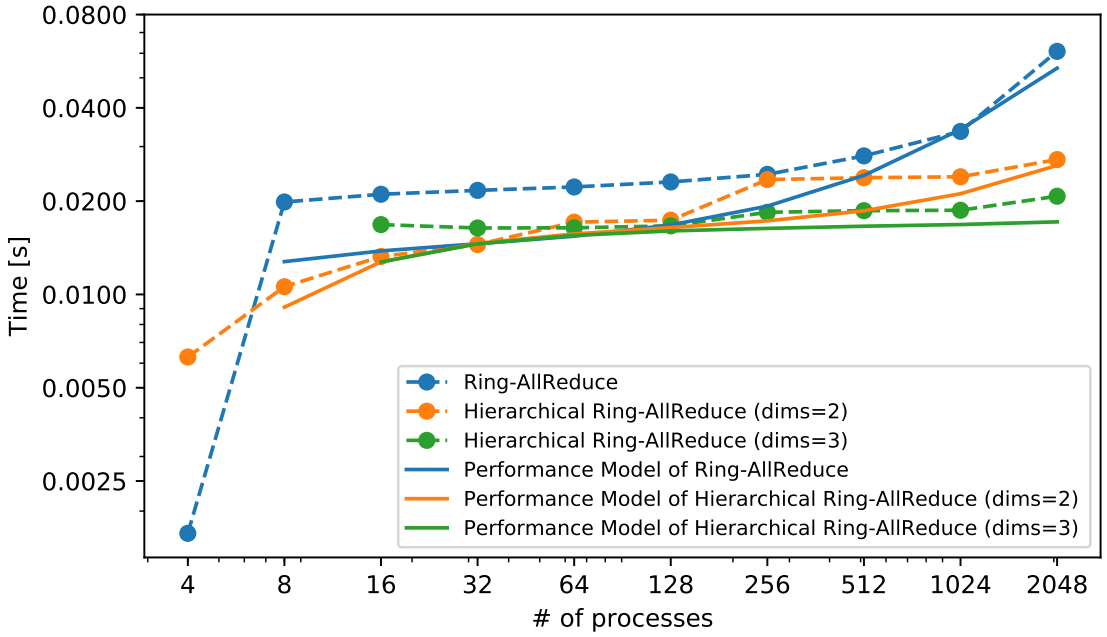

Fig. 3. Communication time of the Ring-AllReduce with and without hierarchy, and its estimation time by the model.

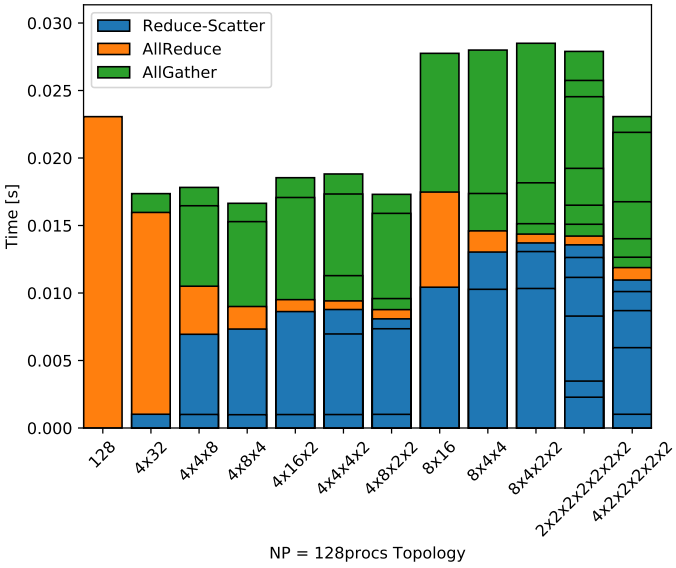

Fig. 4. Communication time of the hierarchical Ring-AllReduce. The total number of processes is 128. The different colors correspond to the Reduce-Scatter, AllReduce, and AllGather. Each layer is split by the border.

a) Evaluation of 256 *processes:* For 256 processes, the best hierarchical topology was $4\times8\times4\times2$, which is approximately 1.42 times faster compared to the flat Ring-AllReduce. The first three intra-rank layers are $4\times8\times4$, which agrees with the best topology for the intra-rank AllReduce experiments. The next best cases are $4\times8\times2\times4$ and $4\times8\times8$, which are both topologies that do not clearly separate intra-rack and inter-rack. A schematic of the communication topology for the 8×8 case is shown in Fig. 7. The communication in the third layer has both intra-rack and inter-rack communications,

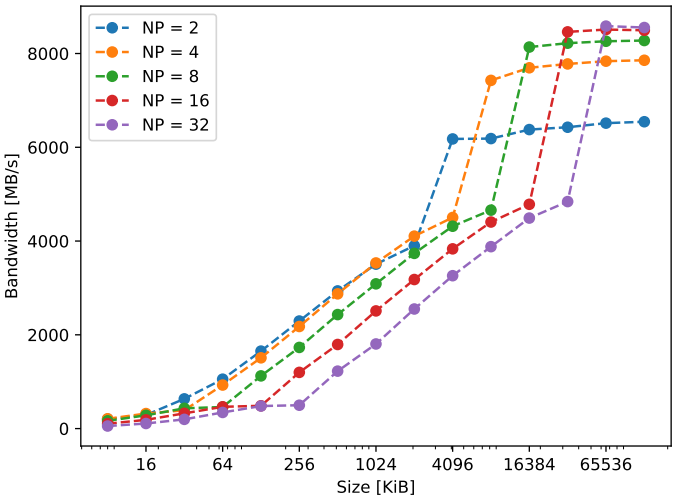

Fig. 5. The attainable bandwidth for pure NCCL communication when the number of processes and message size are varied. We used one GPU per node and one ring for communication to isolate potential network problems.

which limits the number of processes that communicate inter-rack. Therefore, it should put less stress on the 1/3 full-bisection inter-rack bandwidth.

TABLE III
BUS BANDWIDTH OF INTER-RACK HIERARCHICAL ALLREDUCE. (NP = 256)

Topology	Bus Bandwidth [MB/s]			
	1st dim	2nd dim	3rd dim	4th dim
$4\times8\times8$	84179.02	4680.55	2115.40	
$4\times8\times2\times4$	84346.88	4674.03	2922.87	2682.33
$4\times8\times4\times2$	82676.35	4680.52	3448.45	2441.33

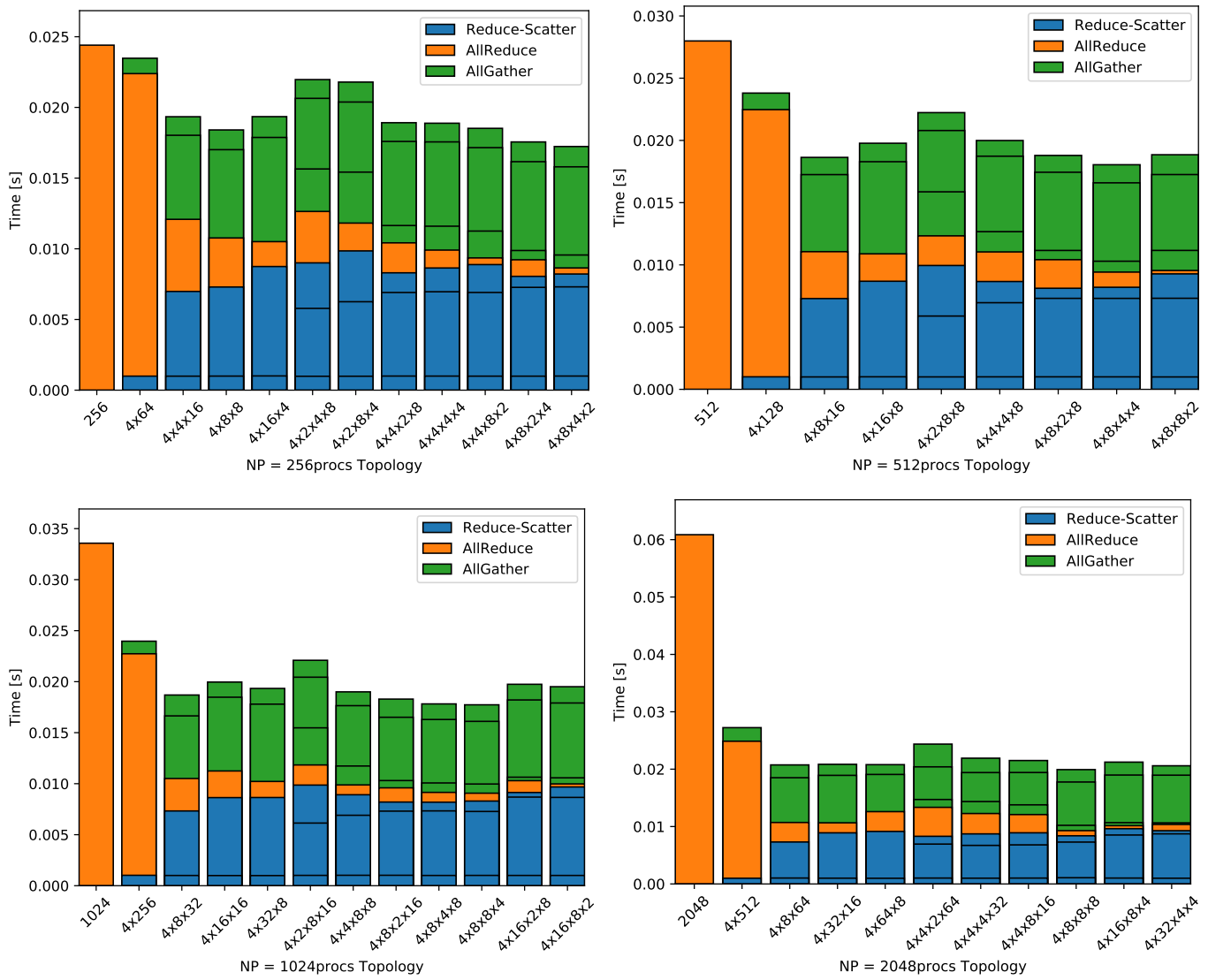

Fig. 6. Communication time of the hierarchical Ring-AllReduce. The total number of processes is 256, 512, 1024, and 2048. The different colors correspond to the Reduce-Scatter, AllReduce, and AllGather. Each layer is split by the border.

Table III shows the effective bandwidth in these inter-node experiments. We expect that the fourth layer bandwidth of $4\times8\times4\times2$ is lower than the third layer one of $4\times8\times8$ due to its message size and inter-rack communication, but as a result, the bandwidth of the third layer is lower than the fourth layer one. In particular, the interconnect of ABCI is implemented via a tree with multi-layer switches, so the inter-rack communication contends with the intra-rack communication. This is known as the Hot-spot problem [18]. Therefore, care must be taken when partitioning the layers so that each layer has a mixture of inter-rack and intra-rack communications.

b) Evaluation of 512 *processes:* When the total number of processes is 512, the fastest communication time is achieved when a hierarchical topology of $4\times8\times4\times4$ is used. The communication time is 1.55 times faster than the flat Ring-AllReduce in this case. This is an extension of the fastest hierarchical topology for 256 processes, and also agrees with the fastest case for 128 processes.

c) Evaluation of 1024 *processes:* For 1024 processes, $4\times8\times8\times4$ was the best hierarchical topology, and was 1.89 times faster than the flat Ring-AllReduce. This topology is an extension of the third best topology for 256 processes, which does not separate inter-rack and intra-rack communications. The second best topology for 1024 processes was $4\times8\times4\times8$, and was 1.88 times faster than the flat Ring-AllReduce. Similar to the 512 process case, this is an extension of the topology where the intra-rack and inter-rack communications are separated.

d) Evaluation of 2048 *processes:* For 2048 processes, $4\times8\times8\times8$ was the best topology, and was 3.05 times faster than the flat Ring-AllReduce. This is an extension of the best topology for 1024 processes.

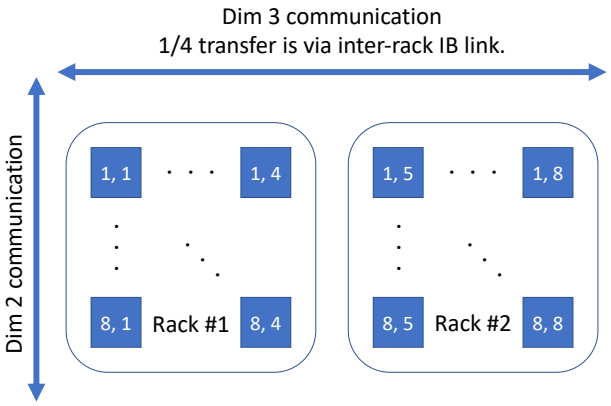

Fig. 7. The relation between the second layer and the third layer for a 4×8×8 topology. The second layer is only intra-rack, but the third layer has 1/4 inter-rack and 3/4 intra-rack communication.

5) How to Determine the Best Hierarchical Topology: In this section, we show how to determine the best hierarchical topology from the experiments on a small number of processes by identifying the universal behavior of the hierarchical AllReduce. Among the different hierarchical topologies that we tested, the top-5 are shown in Fig. 8.

We found that the optimal topology for a larger number of processes is likely to be a simple extension of a smaller scale topology either by:

1) Adding another layer of hierarchy.
2) Increasing the number of final layer processes.

For 128 processes the best topology was $4 \times 8 \times 4$, and for 256 processes the best topology was $4 \times 8 \times 4 \times 2$. This is an example of rule 1. The best topology for 512 processes was $4 \times 8 \times 4 \times 4$, so it also follows rule 2.

As Fig. 5 shows, the effective bandwidth decreases for deeper hierarchy because the message size becomes very small at the end of the hierarchical communication. This suggests that shallower hierarchies with longer rings are preferable as long as each layer is using the same interconnect (*e.g.* NVLink, InfiniBand, etc.). Therefore, which rule to use depends on the balance between the decrease in effective bandwidth for deeper hierarchy, and the increase in latency of having longer rings per layer. This behavior depends on the message size and the number of processes.

Which layer needs to be made longer depends on the case, and we see this in the 1024 case where the best topology is 4×8×8×4, where the third layer is extended (and not the fourth). There is also the effect of InfiniBand congestion, which the current performance model cannot predict accurately.

C. Image Classification result

To evaluate our hierarchical Ring-AllReduce in the actual training, we profile one iteration of ResNet-50 training with the ImageNet-1K dataset. We execute 300 iterations of training and show the median of the profile. To avoid the effect of the straggler process, we add *MPI_Barrier* before the AllReduce. The topologies that we use are $4 \times 8 \times 4$, $4 \times 8 \times 4 \times 2$ and $4 \times 8 \times 4 \times 4$, for NP=128, 256 and 512, respectively.

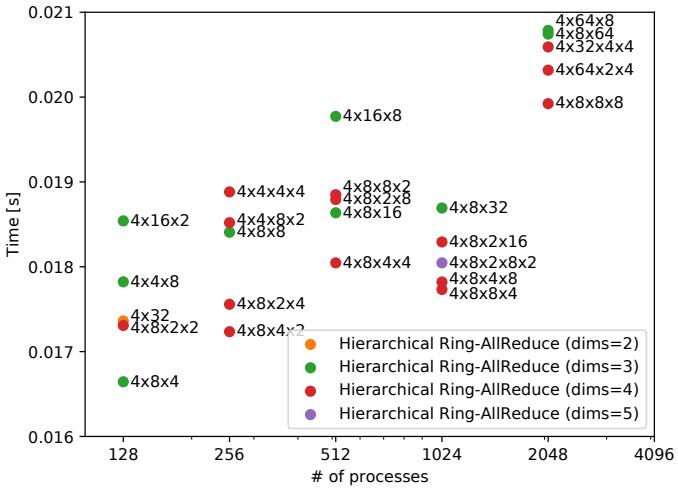

Fig. 8. The top-5 hierarchical topologies for each number of processes.

Table IV shows that using the hierarchical Ring-AllReduce improves the parallel efficiency, computed as a ratio of the total iteration time except for the communication related time range (*i.e.* Pack, AllReduce and Unpack). Compared 256 and 512 processes case, the slowdown of the flat Ring-AllReduce is almost 40%, but the hierarchical Ring-AllReduce is only 14%. Furthermore, the degradation of the parallel efficiency is also.

VI. Related Work

There have been many attempts to reduce the latency of AllReduce by using a hierarchical approach. Horovod [10] uses NCCL and MPI in a hierarchical manner to improve the scalability of AllReduce for deep learning workloads. It has been used to scale deep learning applications to over 25,000 GPUs on Summit [2]. Summit has six GPUs per node, so they first use intra-node NCCL collectives, and use four GPUs for the inter-node MPI collectives. This number four matches the number of InfiniBand HCAs. They use a highly optimized MPI implementation provided by the vendor of Summit, which uses a tree AllReduce to minimize the latency.

With respect to related work on the ABCI supercomputer, there has been a similar work to optimize deep learning

TABLE IV
Profiles of a ResNet-50 / ImageNet-1K Iteration.

	NP = 128		NP = 256		NP = 512	
	Ring	Hier-Ring	Ring	Hier-Ring	Ring	Hier-Ring
Forward	28.6	29.1	28.9	28.9	28.9	28.9
Backward	58.5	58.5	58.6	58.4	58.5	58.5
Pack	1.2	1.2	1.2	1.2	1.2	1.2
AllReduce	18.4	13.7	20.8	15.9	29.3	18.1
Unpack	0.4	0.4	0.4	0.4	0.4	0.4
Update	7.8	7.3	7.7	6.9	7.6	7.1
Total [ms]	114.9	110.2	117.6	111.7	125.9	114.2
Efficiency [%]	0.826	**0.861**	0.81	**0.843**	0.755	**0.827**

communication [7]. Mikami *et al.* adaptively change the mini-batch size during the training, and were able to train ImageNet for very large mini-batch sizes without any degradation of the validation accuracy. They used a two-level hierarchical AllReduce to improve the communication time. However, they did not investigate whether two-levels were optimal.

There have also been efforts to train ImageNet at supercomputer scale using the third generation TPUs [8]. Ying *et al.* used techniques such as Distributed Batch Normalization and Input Pipeline Optimization to train ImageNet in 2.2 minutes without loss of accuracy. They also used a 2-D Ring-AllReduce algorithm to effectively use the 2-D torus interconnect between the TPUs. Their algorithm is optimal for the particular interconnect of TPUs, but is not necessarily optimal on fat-tree-based other supercomputers like Summit and ABCI.

VII. Discussion and Conclusions

A. Limitations

NCCL is not originally designed to do hierarchical Ring-AllReduce communications, and we ended up launching multiple NCCL collectives. This requires the NCCL communications of small messages and causes decreases of bandwidth. There is room for improvement by modifying NCCL itself. This enables us to cache the registered memory for InfiniBand communication between layers, share the communication threads, merge the CUDA kernels to one.

We have only run our experiments on the ABCI supercomputer. Therefore, the effects of different fabric and network topology are not obvious from our studies. Future work will require runs on multiple systems with different interconnect, to investigate the effect of our hierarchical Ring-AllReduce algorithm and the validity of our performance model.

B. Conclusion

In this work, we have implemented a hierarchical Ring-AllReduce algorithm using NCCL, and on the ABCI supercomputer at AIST using various configurations of the hierarchical topology. We observed a significant increase in performance when using the optimal number of layers with the optimal number of processes per layer. We performed an exhaustive search on the hierarchical topology to find this optimal configuration. We found that the optimal topologies on a large number of processes contain the optimal topologies on a small number of processes as a subgraph. This allowed us to construct a performance model that predicts the optimal topology for a large number of processes without actually running on a large number of processes.

C. Future work

This work has the potential to be extended to workloads other than deep learning. We would also like to provide implementations that do not depend on NCCL. Also, more explanation of the empirical optimal topologies can be provided.

Acknowledgment

We are grateful to Akira Naruse (NVIDIA) and Hitoshi Sato (AIST) for useful discussions. Author's internship project at Preferred Networks, Inc. helped with the in-depth understanding of collective communications.

References

[1] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2546–2554.

[2] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, Prabhat, and M. Houston, "Exascale Deep Learning for Climate Analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 51:1–51:12.

[3] R. M. Patton, J. T. Johnston, S. R. Young, C. D. Schuman, D. D. March, T. E. Potok, D. C. Rose, S.-H. Lim, T. P. Karnowski, M. A. Ziatdinov, and S. V. Kalinin, "167PFlopss Deep Learning for Electron Microscopy: From Learning Physics to Atomic Manipulation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 50:1–50:11.

[4] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *arXiv:1706.02677 [cs]*, Jun. 2017.

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

[6] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes," *arXiv:1807.11205 [cs, stat]*, Jul. 2018.

[7] H. Mikami, H. Suganuma, P. U-chupala, Y. Tanaka, and Y. Kageyama, "ImageNet/ResNet-50 Training in 224 Seconds," Nov. 2018.

[8] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image Classification at Supercomputer Scale," *arXiv:1811.06992 [cs, stat]*, Nov. 2018.

[9] "NVIDIA Collective Communications Library (NCCL)," https://developer.nvidia.com/nccl, May 2017.

[10] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," *arXiv:1802.05799 [cs, stat]*, Feb. 2018.

[11] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis," *arXiv:1802.09941 [cs]*, Feb. 2018.

[12] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "ImageNet Training in Minutes," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 1:1–1:10.

[13] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, Feb. 2005.

[14] P. Patarasuk and X. Yuan, "Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, Feb. 2009.

[15] "MPI Solutions for GPUs," https://developer.nvidia.com/mpi-solutions-gpus, May 2013.

[16] A. A. Awan, C.-H. Chu, H. Subramoni, and D. K. Panda, "Optimized Broadcast for Deep Learning Workloads on Dense-GPU InfiniBand Clusters: MPI or NCCL?" in *Proceedings of the 25th European MPI Users' Group Meeting*, ser. EuroMPI'18. New York, NY, USA: ACM, 2018, pp. 2:1–2:9.

[17] "MVAPICH :: Benchmarks," http://mvapich.cse.ohio-state.edu/benchmarks/.

[18] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *2008 IEEE International Conference on Cluster Computing*, Sep. 2008, pp. 116–125.