EFLOPS: Algorithm and System Co-design for a High Performance Distributed Training Platform

Jianbo Dong, Zheng Cao, Tao Zhang, Jianxi Ye, Shaochuang Wang, Fei Feng
Li Zhao, Xiaoyong Liu, Liuyihan Song, Liwei Peng, Yiqun Guo, Xiaowei Jiang
Lingbo Tang, Yin Du, Yingya Zhang, Pan Pan, Yuan Xie
Alibaba Group
{jianbo.djb, zhengzhi.cz, t.zhang, jianxi.ye}@alibaba-inc.com

***Abstract*—Deep neural networks (DNNs) have gained tremendous attractions as compelling solutions for applications such as image classification, object detection, speech recognition, and so forth. Its great success comes with excessive trainings to make sure the model accuracy is good enough for those applications. Nowadays, it becomes challenging to train a DNN model because of 1) the model size and data size keep increasing, which usually needs more iterations to train; 2) DNN algorithms evolve rapidly, which requires the training phase to be short for a quick deployment. To address those challenges, distributed training platforms have been proposed to leverage massive server nodes for training, with the hope of significant training time reduction.**

Therefore, *scalability* is a critical performance metric to evaluate a distributed training platform. Nevertheless, our analysis reveals that traditional server clusters have poor scalability for training due to the traffic congestions within the server and beyond. The intra-server traffic on the I/O fabric can result in severe congestions and skewed quality of service as high performance devices are competing with each other. Moreover, the traffic congestions on the Ethernet for inter-server communication could also incur significant performance degradation.

In this work, we devise a novel distributed training platform, *EFLOPS*, that adopts an algorithm and system co-design methodology to achieve good scalability. A new server architecture is proposed to alleviate the intra-server congestions. Moreover, a new network topology, *BiGraph*, is proposed to divide the network into two separate parts, so that there is always a direct connection between any nodes from different parts. Finally, accompany with BiGraph, a topology-aware allreduce algorithm is proposed to eliminate the traffic congestion on the direct connection. The experimental results show that eliminating the congestions on network interface can gain up to 11.3× communication speedup. The proposed algorithm and topology can provide further improvement up to 6.08×. The overall performance of ResNet-50 training achieves near-linear scalability, and is competitive to the top-rankings of MLPerf results.

I. Introduction

Deep neural networks (DNNs) have made big achievements recently due to its effectiveness on the artificial intelligence (AI) applications, such as image recognition [1], [2], [3], speech recognition [4], natural language processing (NLP) [5], [6], [7], and so forth. For better accuracy, the size of DNN models and training dataset is increasing rapidly. Unfortunately, the big models along with huge datasets inevitably lead to a tremendous increase of training time, which could be several weeks, or even months. To assure a quick model deployment, it is critical to improve the training performance by reducing the training time dramatically.

In general, the contemporary techniques for better training performance could be classified into two categories, *scale-up* and *scale-out*. The scale-up techniques focus on the performance of a single device. For instance, various special purpose processors dedicated to accelerating neural networks have been developed in both industry [8], [9], [10], [11], [12] and academia [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. Nonetheless, as the end of Dennard's scaling and Moore's Law is running out of steam, the performance evolution of a single device slows down and is far away from meeting AI applications' requirements. Therefore, the scale-out techniques, such as the distributed system with data and model parallel training, have been proposed to improve training performance by exploring the intrinsic parallelisms [24], [25], [26], [27], [28]. In this work, we concentrate on the scale-out techniques for the distributed training.

Scalability is one of the most important metrics to evaluate a distributed system. By leveraging massive servers for distributed training, there have been a *de facto* competition to pursue the world record of the least training time for ResNet-50 [29], [30], [31], [32], [33], [34], from one hour to 132 seconds. The lastest result of MLPerf [35] shows that it only takes 76.8 seconds to train ResNet-50 with 2,048 TPUs [8].

Even though the traditional server clusters in the data center could be adopted directly for training, they are suffering from traffic congestions within servers and beyond. On one hand, intra-server traffic on the I/O fabric can cause severe congestions and skewed QoS when several high performance devices, like graphics processing units (GPUs) and network interface controllers (NICs), are competing with each other. On the other hand, the network congestions among servers can lead to long tail latencies in communication. Training tasks are extremely sensitive to tail latencies since their communication traffic is mostly composed of sets of collective operations (e.g., allreduce and allgather) to be executed in sequence. Thus significant performance degradation can be incurred by the congestions on Ethernet.

The intra-server congestions have been partially addressed by NVIDIA's NVLink and NVSwitch solution [36]. Comparing to PCIe, they can provide an order of magnitude higher bandwidth (300GB/s v.s. 32GB/s) for GPU-GPU communication, and allow 16 GPUs to communicate with each other within a server [37]. Thus, they provide a congestionless

2378-203X/20/$31.00 ©2020 IEEE
DOI 10.1109/HPCA47549.2020.00056

intra-server network to maximize the training performance. However, NVLink and NVSwitch are dedicated to NVIDIA's GPUs, which are neither flexible nor extensible to support other accelerators. They also have poor scalability due to their physical limitation on the signal's effective propagation distance. Moreover, the technology does not take care of inter-server communication. Therefore, it is not a solution for large-scale distributed training.

Distinct from NVLink-NVSwitch, in this work we focus on the innovation on the PCIe and Ethernet NIC that have been standardized for decades. We revise the system architecture of general purpose clusters and propose a high performance distributed training system, namely EFLOPS, to mitigate the aforementioned traffic congestions and thus achieve near-linear scalability. For intra-server communication, we introduce a new network to deal with the communication congestion among high performance peripherals.

For system-wise scalability and compatibility, we propose to direct the intra-server traffic outside the chassis and reuse the existed *Top of Rack* (TOR) switches for device-level communications. To this end, we install the same number of GPUs and NICs, and bind one GPU and one NIC as a pair under the same PCIe switch. We denote one GPU-NIC pair as a *sub-node*. In this way, a virtual switch, *Top of Server*, is formed, dedicating for the communications among sub-nodes within a server. As a result, there is no direct communications among the sub-nodes via I/O fabric so that the intra-server congestions are alleviated.

For inter-server communication, we propose a new network topology, *BiGraph*, in which switches are evenly divided into two separate parts and the two parts are interconnected with CLOS network [38] and as a complete bipartite graph. Different from Fat-Tree's *Leaf-Spine* architecture, we denote the two parts as *upper* and *lower* part. In other words, there is always a direct link between any two switches from different parts. For each switch, regardless of which part it comes from, a group of sub-nodes are attached. This is different from Fat-Tree topology since only *leaf* switch in Fat-Tree is the access point for server nodes. The idea behind the BiGraph is two fold. First of all, there are abundant direct links across the two parts of BiGraph, which provide more options to find some less congested (or even free) links for new connections. Secondly, the shortest path is deterministic for those cross-part connections. As a consequence, the controllability of network traffic can be enhanced in the software libraries.

To fully utilize the abundant network resources, we propose a topology-aware allreduce algorithm, *Halving-Doubling with Rank Mapping* (`HDRM`), that has been implemented in our self-developed communication library. With an in-depth analysis on Halving-Doubling algorithm [39], we have observed that the total number of connections required in each step of communication exactly matches the number of links in BiGraph. Therefore, it opens up an opportunity to build a one-to-one mapping from connections to physical links, by exploiting the controllability of network traffic. In `HDRM`, we re-rank the sub-nodes to ensure that one connection takes up a link exclusively. Therefore, network congestions are eliminated and superb communication performance can be achieved.

In summary, the major contributions of our work are as follows.

- We propose EFLOPS as the architecture for high-performance training, which revises traditional architecture of data center servers and network topology to mitigate the traffic congestions within servers and beyond.
- We propose BiGraph topology and the topology-aware allreduce algorithm `HDRM`, which can achieve near optimal communication performance.
- We have deployed EFLOPS by the off-the-shelf products and developed custom communication library to enable `HDRM`. We run MLPerf benchmark on EFLOPS and the competitive results demonstrate its promising benefits.

II. Related Works

The scalability of distributed training systems are bounded by the communication cost. Prior studies have proposed several approaches to address this issue, including 1) increasing computation time, 2) decreasing communication time, and 3) maximizing the overlapping between computation and communication to hide communication latency.

The computation time, relative to communication time, can be extended with larger batch sizes [40]. Nevertheless, as the batch size grows, the effect of SGD will be affected adversely so that it can incur noticeable accuracy loss. Furthermore, the batch size is usually limited by the capacity of underlying devices, such as GPU memories, or HBM on TPUs [8], [9], [11]. Therefore, this method has limited effect on increasing the computation time.

On the other hand, the communication time could be reduced by gradient compression techniques and faster network fabrics. Since it's challenging to accomplish reasonable compression ratio via lossless compression techniques [41], lossy compression techniques are proposed [42], [43]. However, these methods may result in accuracy loss as well. In addition, customized fabrics and clusters have been developed to reduce the communication time [36], [24]. They are either limited to server scale, or lack of algorithm detail of communication.

Recently, communication scheduling techniques have been proposed to achieve a better overlapping between computation and communication [44], [45], [46], which can effectively hide the communication latency. Based on the observation that the gradients of different layers are generated and consumed in reversed order, prioritizing the generated gradients and allowing the critical slices to preempt the data transfer could improve the overlapping further.

For the first time, we propose EFLOPS to decrease communication time spent on network fabrics by an architecture and algorithm co-design. Therefore, EFLOPS is orthogonal to these techniques from categories 1) and 3). In fact, it can work together with some of them for high performance training.

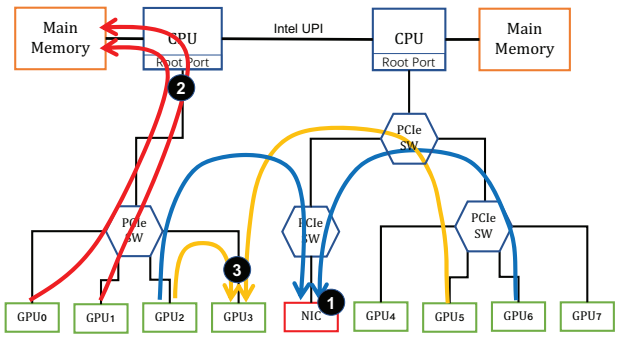


Fig. 1: Traffic congestions in a traditional server: 1) congestion on NIC; 2) congestion on PCIe root complex; and 3) skewed QoS among congested flows.

III. Background and Motivation

A. Congestions in Server Architecture

A traditional server architecture is shown in Figure 1, which has dual CPU sockets with several peripheral accelerators (e.g., GPUs[1]) attached to PCIe [47] root complexes. The two CPU sockets communicate with each other through Intel Ultra Path Interconnect (UPI) [48]. PCIe switch (SW) is employed to enable the communication among those accelerators. Note that a single NIC is installed and responsible for the network communication for the entire server. In such an architecture, there are three types of components available for communication, i.e., PCIe fabric, NICs, and Intel UPI. For all kinds of communications, they could leverage one of the components or a combination of them. Correspondingly, there are three different types of traffic congestion in a server node: ❶ congestion on NIC, ❷ congestion on PCIe root complex, and ❸ skewed quality of service (QoS) among congested flows.

First of all, the server can experience severe traffic congestion on NIC once multiple GPUs are populated. The process of data-parallel training is usually divided into multiple iterations, and the input dataset is split into batches. All involved GPUs should allreduce their local gradients with others before they can update the parameters of the neural network. This would cause a set of synchronized communications that generates huge amount of simultaneous NIC accesses and thus lead to severe congestions on the only NIC.

Secondly, congestions could also occur on the PCIe root complex. Once a GPU initiates memory requests to the host memory, the requests should first reach the root complex. As the bandwidth of host CPU's on-chip interconnect and main memory interface are much higher than that of PCIe bus, the aggregated traffic can oversubscribe PCIe root complex and incur congestion accordingly. This is a common case for AI training since the global parameter update requires the data loading of next batch to be synchronized among all GPUs. It produces the traffic burstness on the PCIe root complex.

[1]In fact, EFLOPS is generic enough to support various high performance accelerators. Without loss of generality, GPU is used as the representative throughout the paper.

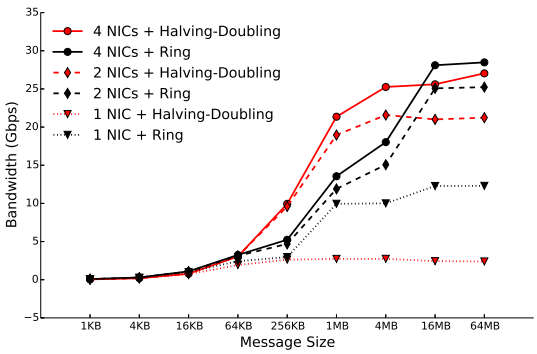


Fig. 2: The achieved bandwidth with different number of NICs and message sizes. `HD` outperforms `Ring` with 4 NICs/Node and 256B to 4MiB message.

Finally, training tasks are in favor of direct communications among GPUs. If the communication is done by PCIe fabric, congestions could occur with in-cast traffic. According to the prior study [49] , the bandwidth of two congested flows could be varied if they experience different round-trip time. That is, fairness among congested flows cannot be guaranteed in PCIe fabric with its point-to-point credit-based flow control mechanism.

B. Motivation Study

To understand the impact of congestions in a server, we built a mini-system that has one server node with four sub-nodes attached and another four sub-nodes connected to the same Ethernet switch. The sub-nodes can talk with each other via the Ethernet switch at line speed. We tested the performance of allreduce operation with one, two, and four available NICs, respectively. We run Ring-based (`Ring`) and Halving-Doubling (`HD`) algorithms that are widely used in the network communication[2]. We developed the synthetic workloads in which the message size could vary from 1KiB to 64MiB. Those synthetic workloads are carefully designed to include GPU-NIC and GPU-GPU communications via allreduce operations, as well as CPU-GPU communication via CUDA memory copy APIs.

1) Impact of Congestion on NIC: The achieved bandwidth with different number of NICs and message sizes is illustrated in Figure 2. The overall bandwidth generally increases as the message size grows. This is due to the less overhead amortized by the bigger message size. Moreover, when the message size is greater than 64KiB, multiple NICs obtain significant speedup over the single NIC. By applying `HD` algorithm, the four-NIC system can obtain up to 11.3× speedup over the one-NIC system, and 1.27× over two-NIC system. Similarly, four-NIC system with `Ring` can be up to 2.31× as fast as one-NIC system, and 1.13× as two-NIC system.

When comparing the two algorithms, `Ring` performs better in a one-NIC system. This is because `Ring` can contain half of

[2]See Section V-A for the detail.

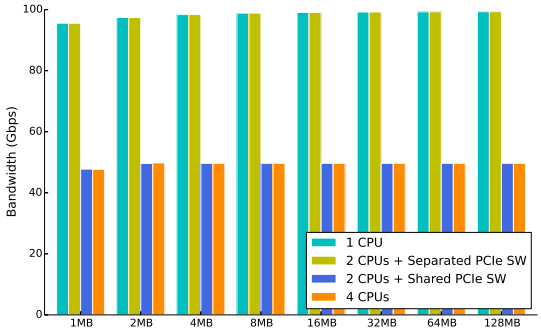


Fig. 3: The effect of shared PCIe root port on CPU-GPU communication. GPU can suffer from about 50% bandwidth drop once the PCIe switch is shared by multiple GPUs.

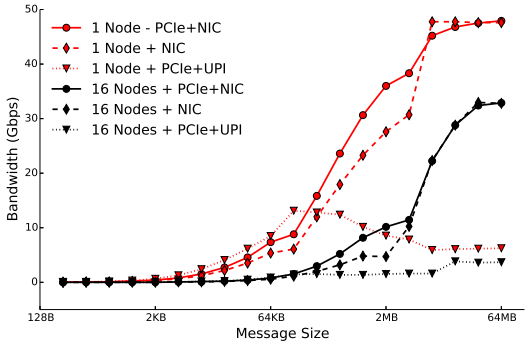


Fig. 4: The impact of latency on the bandwidth of intra-server communication.

communications within the server node so that less traffic goes through the NIC. Alternatively, `HD` performs better in four-NICs system for moderated message size (i.e., 256B-¿4MiB). `HD` effectively increases link utilization with bigger data size and less steps of communication.

2) Impact of Congestion on PCIe Root Complex: To examine the impact of congestions on PCIe root complex, a bunch of simultaneous tasks are initiated to copy data from host memory to GPU memory (one task per GPU). The number of tasks varies from one to four. According to whether the targeted GPU devices are attached to the same PCIe switch, the test with two concurrent tasks is further divided into two cases. As shown in Figure 3, the bandwidth of a single task can be more than 90Gbps while it drops to almost half when running four concurrent tasks on four GPUs separately. This is reasonable since every two tasks share the same PCIe root port in our evaluation system. It is double-proved by running two concurrent tasks with shared PCIe switch. For GPUs under different PCIe switches, the bandwidth can go back to 90Gbps.

3) Impact of Skewed QoS on PCIe: We further run a set of tests to evaluate the effects of skewed QoS within a server. Several scenarios are created with `Ring` algorithm. In the first scenario, the GPUs connected to the same PCIe switch communicate with each other via PCIe fabric, while the inter-socket communications are handled by NIC (*`PCIe+NIC`*). In the second scenario, the inter-socket communications are handled by UPI link (*`PCIe+UPI`*). In contrast, all communications are handled by NIC in the last condition (*`NIC`*). We vary the number of nodes between 1 and 16 to see if it makes any difference. Each node has four sub-nodes attached.

As shown in Figure 4, the results of the single node show that *`PCIe+UPI`* has the highest bandwidth for small message due to the low latency of UPI links. However, it performs worse when message size becomes large. In particular, the bandwidth of *`NIC`* is up to 7.6× higher than *`PCIe+UPI`* as message size is over 8MiB. The root cause is that the inter-socket communication among GPUs goes through the on-chip interconnects, where the I/O and inter-socket packets usually have low priority. When considering *`NIC`* and *`PCIe+NIC`*,

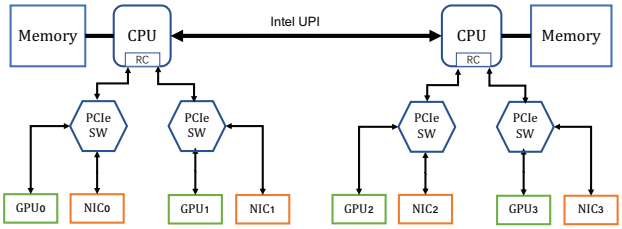


Fig. 5: The proposed congestionless server architecture. Each GPU and NIC are paired as a sub-node and attached to the same PCIe switch.

the two configurations achieve similar maximum bandwidth when message size is large enough. This is expected as the performance of large message is determined by the overall available bandwidth. For small messages, the PCIe fabric shows certain advantage when the link is not saturated, which can bring 30% improvement.

As scaling out the system to 16 nodes, there are multiple Ethernet links for communication. Therefore, the total communication delay increases drastically so that the saved latency by UPI becomes minor. Therefore, *`PCIe+UPI`* has the lowest bandwidth. On the other hand, the benefit of PCIe fabric shrinks as well, because congestion is generated on the network and the bandwidth saturates earlier. Thus, *`PCIe+NIC`* and *`NIC`* have similar bandwidth of large message.

IV. System Architecture

A. Congestionless Server Architecture

As shown in Figure 5, in order to alleviate the three types of congestions, we propose a congestionless server architecture. The new server directs the intra-server traffic outside the chassis to alleviate the congestions on I/O fabric. The high performance peripheral devices communicate with each other via Ethernet switches, which is denoted as *Top of Server*. In particular, there are same number of GPUs and NICs installed in the new server node. We further associate a GPU with an NIC as a pair. Such GPU-NIC pairs are denoted as *sub-nodes*. Note that the devices in the same sub-node should be populated on the PCIe slot that is attached to the same PCIe

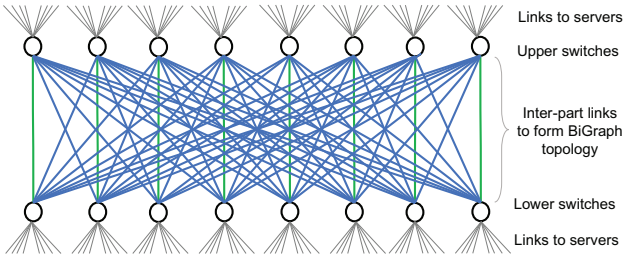


Fig. 6: The BiGraph Network Topology. Switches are evenly divided into upper and lower parts.

switch. This is to keep the GPU-NIC traffic within the sub-node in the PCIe switch.

Secondly, we propose to evenly distribute the sub-nodes into all available sockets to alleviate the congestions on PCIe root ports. As long as the aggregated available bandwidth of all PCIe root ports are sufficient for all sub-nodes, there should be no congestions on PCIe root ports. Otherwise, some techniques, like GPU prefetching, should be considered to mitigate the congestion on root ports. At last, since it can have large performance variation, we propose to limit the amount of inter-GPU communication on PCIe fabric. With these proposed solutions, all aforementioned congestions can be alleviated in certain extent. The efficacy of these proposals are evaluated and discussed in depth in Section VII.

B. BiGraph Network Topology

To the best of our knowledge, most data centers have been built with Fat-Tree network topology, which is a mulit-layer architecture and can sustain the same aggregated bandwidth in each layer. Nonetheless, since the hashing function for routing can never be perfect, traffic congestions can occur when multiple flows are routed to the same link that has been heavily loaded. Even though various congestion control techniques [50], [51], [52], [53], [54] have been well studied in TCP/IP networks, it is still one of the major challenges in a lossless RDMA network [55]. In this work, we would like to eliminate congestions through a novel network topology and the corresponding connection control algorithm.

In BiGraph topology, network switches are divided into two parts, the *upper* and *lower* (circles in Figure 6). Similar to Fat-Tree, the two parts of switches are interconnected with CLOS architecture [38]. Therefore, BiGraph is compatible with Fat-Treetopology so that existing network management mechanisms for Fat-Tree can be easily applied to BiGraph. The big difference is that Fat-Tree only allows network access by *leaf* switches while BiGraph enables the network accesses from both parts of switches. In other words, every switch in BiGraph logically plays two roles, the *spine* and *leaf*.

Except the upstream ports for switch interconnection, the rest ports in a switch are used for downstream sub-nodes interconnection and system scaling. To guarantee that all layers have the same bi-section bandwidth, the aggregated bandwidth of downstream links is constrained to not exceed the bandwidth of upstream links. In other words, in a system

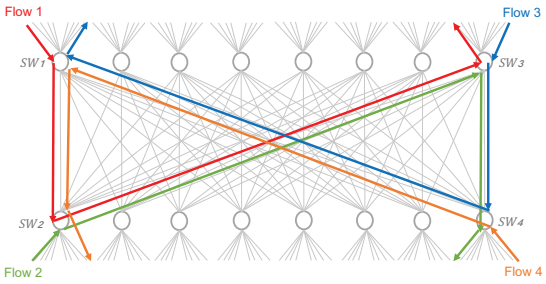


Fig. 7: An example of deadlock due to the dependency loop of Flow 1-4, in which every source switch has two flows compete for the output port.

with N switches (two layers in total) and M sub-nodes connected to each one of them (so-called a $M \times N$ BiGraph network), the constraint $M \leq N/2$ should be satisfied if the bandwidth of NICs matches that of switches.

There are two remarkable features of the BiGraph topology: 1) It provides abundant link resources between the two parts of switches. In a P sub-nodes system, there are at least $P/2$ bi-directional links available [3]. Therefore, the number of available links is equal to the number of connections required for allreduce operation when `HD` algorithm [39] is applied. As a consequence, it opens up an opportunity for us to map these connections to the available direct links one by one, which eventually eliminates congestions. 2) The shortest path between any two sub-nodes from different parts of BiGraph is deterministic. Such a deterministic path gives us the full control on the data path for the cross-part connections in the library. For instance, we can control the traffic by selecting the source and destination sub-nodes for communication.

Note that there is a risk of deadlock as illustrated in Figure 7. The *spine* switches are responsible for redirecting the packets from a *leaf* switch (in the other part). If multiple *spine* switches from both sides work simultaneously, deadlock could be induced due to a dependency loop. As shown, four concurrent flows (1-4) start and end on the same side, and only make turns on the other side. Therefore, every flow could take the output ports in the source switches and compete for the output port in the next hop. For instance, $Flow1$ takes the output port of $SW1$ (to $SW2$), and competes with $Flow2$ for the output port of $SW2$ (to $SW3$). If $Flow3$ and $Flow4$ have similar behavior, a close-loop dependency is formed and thereby the risk of deadlock is high. As a solution, we classify the traffic in different directions into separate priorities. As a result, they are processed in different queues and thus the dependency loop is broken.

V. Algorithms

To make use of the BiGraph topology, the allreduce algorithm should be redesigned accordingly. In this section, we first review two popular allreduce algorithmsWe then propose the *Halving-Doubling with Rank-Mapping* (`HDRM`) to assure

[3]In a $M \times N$ system, the number of sub-nodes is P=$M \times N (\leq N^2/2)$. The number of direct links is $N^2/4$.

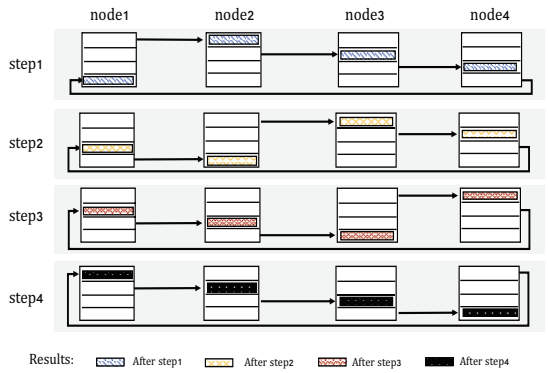


Fig. 8: Ring-based algorithm for allreduce.

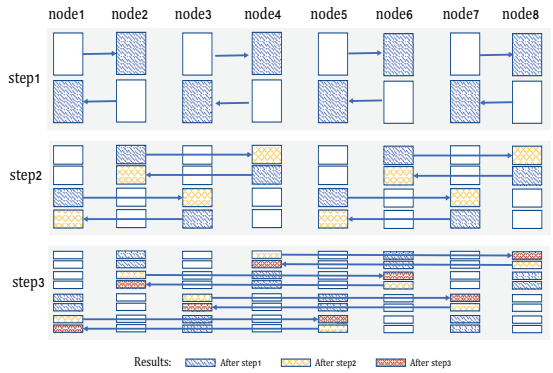


Fig. 9: Halving-doubling algorithm for allreduce.

that all steps only consist of cross-part connections, and can be mapped to inter-part links one on one.

A. Existing Allreduce Algorithms

Allreduce is one of the most important operations in a distributed training system. It is usually used to average the gradient values among the involved sub-nodes. Due to the data dependency in the back propagation, the next training iteration *cannot* start unless the parameters have been updated with the gradients. Therefore, the allreduce operations can only be executed sequentially in the training process. According to Amdahl's Law [56], the performance of allreduce operation determines the overall performance of a distributed training system.

1) Ring-based Algorithm (`Ring`): To support the allreduce operation, it is straightforward to build a fully-connected network and send the data to all other sub-nodes simultaneously for the reduce operation. In a P sub-nodes system, it needs $P\times(P-1)$ connections in total and one-step only communication. Though it seems fast and simple, it can actually incur large-scale congestion when too many streams are sending data simultaneously.

In order to reduce the possibility of congestion, `Ring` is widely adopted since it requires only P connections. With P connections, it needs $P-1$ steps of communication to accomplish the same functionality. In every step, a sub-node receives the data from its left neighbor, conducts reduce operation with the received data and local result, and forward the received data to its right neighbor. After $P-1$ steps of data transferring and reducing, all sub-nodes now have the data from others, and the final reduce is executed to generate the final result. Even though the total amount of data transferred in each sub-node is same as that of a fully-connected network (i.e., $(P-1)\times S$, where S is the data size), `Ring` usually outperforms fully-connected network due to the fewer connections and less congestions [57].

As illustrated in Figure 8, the `Ring` algorithm can be further optimized with chunked data transferring and reducing. It is composed of two phases of collective operations, reduce-scatter and allgather. From the view of sub-nodes, they iterate over the chunks of data and process them one by one, starting with a specific *offset*.

In the reduce-scatter phase, a sub-node sends the chunk of data at offset **i** to its right, receives the chunk of data at offset **i-1** from its left, reduces the received data with local data, and advances the offset index to **i+1** for next step. As a result, the final reduce results are scattered among the sub-nodes at offset **i** after the chunks of intermediate results have traversed through all the sub-nodes in the ring. The process in the allgather phase is similar to produce the final results. Comparing to the original algorithm, the number of connections required does not change but the size of data transferred by each sub-node is reduced to $2\times S$.

2) Halving-doubling Algorithm (`HD`): Halving-doubling algorithm is another popular algorithm for allreduce operation. It also requires two processing phases that are comprised of several steps of communication in each. Distinct from `Ring`, `HD` needs only $P/2$ connections in each step in a P sub-nodes system, as shown in Figure 9. By enumerating the P sub-nodes from 1 to P, we call the difference of their rank numbers as *rank distance*. In the first step, the sub-nodes with rank distance=1 are paired together, sending half of self-owned data to the peer and receiving the other half from the peer for reduce. Consequently, the intermediate results are shared with the peer sub-nodes. In the following steps, new pairs are set up with doubled rank distance=2, and the sub-node can exchange half of the intermediate results with its new peer. As a result, the sub-nodes with doubled rank distance can run reduce operation. After $\log_2 P$ steps, the reducing results are distributed among all sub-nodes. Similarly, `HD` needs $\log_2 P$ steps to allgather local results to others.

As is known, `HD` transfers the same amount of data as `Ring` does. Since `HD` needs much fewer connections in each step, intuitively it should outperform `Ring` due to less congestions on network. Nonetheless, the evaluation results in Section VII show that it is less performant in many cases. The root cause is that the connections in `HD` are changed in every step so that it increases the possibility of congestion. This observation inspires us to devise a new algorithm to adapt to BiGraph topology. Recall that BiGraph has $P/2$ highly controllable direct links between the two parts. It is possible

Algorithm 1: The algorithm for rank mapping

define `append()`:

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix} = append\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \dots, \begin{bmatrix} x_n \\ y_n \end{bmatrix}\right)$$

end

define `merge()`:
  $R = merge(R_0, R_1, \dots, R_n)$
  results in, $R[x][y] = \{R_0[x][y], R_1[x][y], \dots, R_n[x][y]\}$
end

Function `main()`:
  input : Number of nodes for rank mapping, X
    System information, $M \times N$
  output: Rank mapping results: R
  initialize ranks for 2-node pair, $r = \{0; 1\}$
  for $i = 0; i < N/2, i < X/2; i++$ **do**
    initialize other 2-node pairs, $r_i = r + i * 2$
    if *the binary expression of i contains odd number of 1s* **then**
      swap the two elements in r_i, $r_i = \{r_i[1]; r_i[0]\}$
    end
  end
  if $X \leq N$ **then**
    $R = append(r_i), i \in \{0 \dots X/2\}$
    return R
  else
    generate the base of virtual groups,
    $B = append(r_i), i \in \{0 \dots N/2\}$
  end
  for $j = 0; j < M, j < X/N; j++$ **do**
    initialize other virtual groups, $R_j = B + i * N$
    if *the binary expression of j contains odd number of 1s* **then**
      swap the two elements in R_j,
      $r_i = \{R_j[1]; R_j[0]\}$
    end
    cyclic left-shift the first vector of R_j with j steps,
    $R_j[0] = R_j[0] << j | R_j[0] >> (N - j)$
  end
  $R = merge(R_j), j \in \{0 \dots X/N\}$
  return R
end

for us to map these connections to those direct links without any link sharing so that `HD` becomes favorable.

B. Halving-Doubling with Rank Mapping (`HDRM`)

After the review of the BiGraph topology and Halving-Doubling algorithm, we can summarize the requirements of mapping connections to direct links between two parts as follows.

- The sub-nodes within the same pair should be mapped to different parts.
- For the sub-nodes connected to the same switch, their peers should be under different switches.

We denote the sub-nodes connected to the same switch as a *physical group*. We pick one sub-node out of each physical group to form a *logical group*. In an $M \times N$ system, there should be N physical groups, and M logical groups. As shown in Figure 10, the physical groups are illustrated via dashed rectangles. The sub-nodes in the same logical group are scattered across physical groups and indicated by the same color.

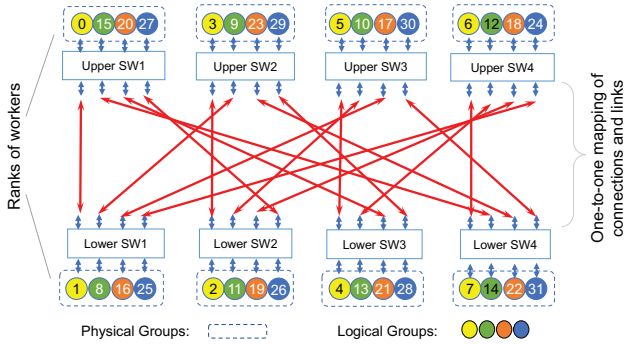


Fig. 10: An example result of HDRM algorithm. Nodes in same color is in the same logic group. Nodes in the dashed block is in the same physical group.

The proposed `HDRM` algorithm is elaborated in Algorithm 1. We first do rank mapping for one logical group. The first logical group is used as a base for the ranking of other logical groups. To accomplish the base of ranking, we start from the system with two sub-nodes, which has only ranks $[0; 1]$ mapping to two different parts of BiGraph. The other ranks, from 2 to $N-1$, can also be viewed as pairs from $[2; 3]$ to $[N-2; N-1]$. All these pairs are encoded with ID numbers, from 0 to $N/2-1$. For the pairs whose binary expression of its ID number has odd numbers of 1s, the ranks of sub-nodes is forced to swap. For example, we should re-rank the pair $[2; 3]$ as $[3; 2]$, since its ID number is 1. Thus an eight-node base can be generated by appending all pairs of ranks together, i.e., $[[0, 3, 5, 6]; [1, 2, 4, 7]]$ as shown in Figure 10 (yellow balls).

For multiple logical groups, the rank mapping has some minor changes. Firstly, we start from the base generated in last step, rather than ranks $[0; 1]$. Here we define the ranks in a virtual group as a set with two vectors, $[X; Y]$, where X=$[0, 3, 5, 6]$ and Y=$[1, 2, 4, 7]$ in Figure 10. The initial ranks for any other logical groups can be generalized as R_i=$[X + i \times N; Y + i \times N]$, where i indicates the sequence number of a logical group. We swap vector X and Y when the binary expression of the sequence number has odd number of 1s. In this case, R_i=$[Y + i \times N; X + i \times N]$.

A closer look at the intermediate mapping results reveals that the sub-node pairs with the same index from different logical groups compete for the same link. To address this issue, we need an extra step to map these conflicting connections to other free links. In this work, we choose to left-shift the first vector by i, as R_i=$[(R_i[0] << i)|(R_i[0] >> (N-i)); R_i[1]]$, where $R_i[0]$ and $R_i[1]$ is the first and second vector, respectively. Note that the shifting direction (i.e., left or right) and the selected vector (i.e., the first or second vector) have no effects on the effectiveness of `HDRM`.

With the ranks generated for logical groups, the final result can be obtained by merging them together. For all logical

TABLE I: System Configuration and Benchmarks

System Configurations	
Topology	BiGraph
Scale	4 sub-nodes × 16 virtual switches
Switch	H3C S9850-4C, 100Gpbs
Server Node	Xeon Platinum 8163 x2, PEX 9797 x2
Sub-node	V100-PCIe 32GB + CX-5 (MT27800)
Tools	Cuda Toolkit, OpenMPI [58]
AI Frameworks	TensorFlow [59], PyTorch [60], MxNet [61], Horovod [62]
Evaluation Workloads	
Synthetic[4]	GPU-NIC communication CPU-GPU communication GPU-GPU communication
Application	ImageNet: ResNet-50 [63] Image Classification: ResNet-50+FC NLP: BERT-base [64]

groups, the ranks with the same index should be merged so that they can fall into the same physical group and be connected to the same switch. Note that the nodes from the same physical group are interchangeable, which means exchanging the rank of nodes in the same physical group does not make any difference.

Figure 10 illustrates the rank mapping results of 32 sub-nodes in a 4×8 system. The connections in the first step of communications are indicated with lines between them. As shown, there is only one connection between any two physical groups from different parts (i.e., two switches). Therefore, the links between them are exclusively used by the mapped connections, and the congestion is eliminated accordingly.

VI. System Deployment

To evaluate the effect of our proposed architecture and algorithm, we have set up a platform with commodity products and developed the collective communication library with the HDRM algorithm embedded. The system configuration is summarized in Table I. The platform is built with 16 server nodes. Two Intel Xeon Platinum 8163 processors are connected by UPI link in each node. In addition, two PLX 9797 PCIe switches are attached to the root ports of processors with 16-lane upstream links. Ten links (five for each) with the same width are reserved for downstream ports. Two NVIDIA Tesla V100-PCIe 32GB GPUs and two Mellanox CX-5 (MT27800) 100Gbps NICs are installed to PCIe slots connected to the same switch.

With 16 server nodes, there are 4×16 sub-nodes in the system. For this 4×16 system, it is straightforward to build a BiGraph network with 16 switches, 8 in each part. To simplify the deployment, we split one physical switch into two virtual switches (VS_x) with VLANs, assigning one for each part of BiGraph. As shown in Figure 11, we deploy the network with H3C S9850-4C switches, which provides 32 ports at 100Gbps speed. We split it into two 16-port virtual switches. The two parts of switches are differentiated with different colors. For each switch, there are eight ports to connect to the switches in the other part. Four of the rest ports are used to connect to the four sub-nodes of a server, and the other four are left

[4]See the motivation study in Section III-B

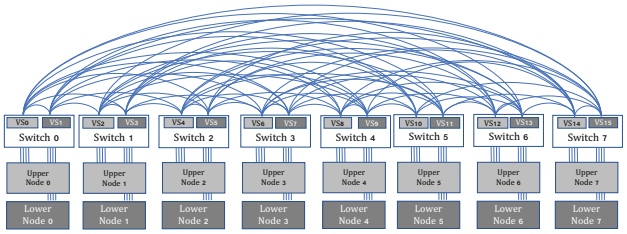


Fig. 11: The deployment of EFLOPS as a 4×16 training system. One physical switch is split into two virtual switches (VS_x) for simplicity of design.

open. We have also configured the DSCP of servers to match the parts they belong to.

Since most of existing collective communication libraries cannot fully take advantage of the proposed system, we build our own collective communication library that implemented the HDRM algorithm to optimize the performance of allreduce and allgather communications. At the bottom layer, RDMA (RoCE) [55] is deployed for data transfer. Moving up, there are a lot of EFLOPS-specific optimizations in the context initialization layer. It looks up the locations of GPUs and pairs the nearby NIC with the GPU accordingly. During the network initialization, it scans the topology information and re-ranks GPUs according to the proposed algorithm.

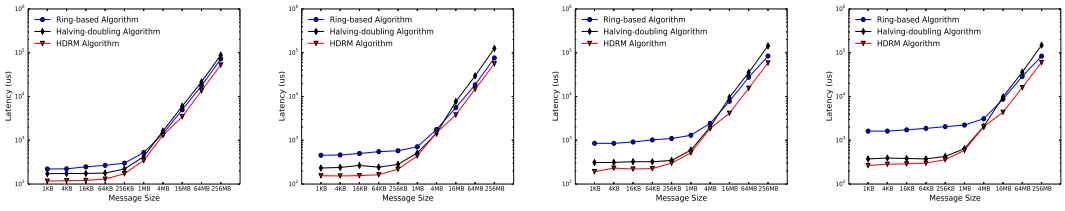
VII. Evaluation Results

In this section, we first show the performance result of synthetic workload. We then disucss the results of real workloads, including MLPerf [35] training workload ResNet-50 [63] and two in-house applications where ResNet and BERT [64] are deployed, respectively. After that, we conduct the scalability study and cost analysis.

A. Synthetic Workloads

To demonstrate the efficacy of the proposed algorithm-topology co-design, we run a group of experiments on the deployed platform with `Ring`, `HD`, and the proposed `HDRM`algorithms. To examine the scalability, we sweep the number of sub-nodes involved (namely workers) from 8 to 64. The evaluation results are shown in Figure 12. The four charts on the top present the average latency to finish an allreduce transaction. The other four charts at the bottom show the bandwidth. The X-axis is the message size while the Y-axis of the upper figures is in logarithm scale.

The upper four charts verify that the `HDRM` substantially outperform the other two algorithms. This is because `HDRM` can effectively eliminate the congestions on the network, as discussed in Section V. On the other hand, `HD` outperforms `Ring` when message sizes are relatively small (64KiB-¿4MiB). This is expected since `Ring` needs $O(n)$ steps of data transferring for each allreduce transaction, while `HD` only needs $O(\log n)$ steps. Whereas, the conclusion is flipped once message size becomes large enough ($\geq$8MiB). It still makes sense since `Ring` needs only $O(n)$ connections for each



(a) Lat of 8 workers (b) Lat of 16 workers (c) Lat of 32 workers (d) Lat of 64 workers

(e) BW of 8 workers (f) BW of 16 workers (g) BW of 32 workers (h) BW of 64 workers

Fig. 12: Latency and Bandwidth Comparison with 8/16/32/64 works.

allreduce transaction, while HD needs $O(n^2)$ connections that increases the possibility of congestions.

From the upper four charts, we can find that the latency of small messages are growing as the numbers of workers increase. For Ring, when the number of workers increases from 8 to 64, the latency of 1KiB message increases by 7.3× (1618us v.s. 221us). The latency of HD increases by only 2.17× (376us v.s. 173us). Therefore, Ring does not scale well at small message size. When comparing HD to HDRM, HDRM can further reduce the latency by 40%-50% (116.6us and 266us, respectively). As a result, HDRM presents 6.08× speedup and the best scalability.

As illustrated in the lower four sub-figures, the averaged bandwidth are generally increased as the message size grows. From the curves of Ring and HD, a crosspoint could be observed. That is, Ring performs better for large messages, while HD performs better for small messages. With 64 workers, HD outperforms Ring by about 10Gbps higher throughput (13.95Gbps v.s. 3.68Gbps) for 1MiB message. While it sees 11Gbps less throughput for 256MiB message (14.07Gbps v.s. 25.01Gbps). This is because the side effects of network congestion are augmented as the message size increases, and HD suffers more than Ring so that it has no significant throughput improvement when message size is ≥1MiB. With HDRM, we can eliminate the network congestions and make HD outperform Ring substantially. In other words, the throughput of HDRM is about 10Gbps higher than Ring for 256MiB message (34.85Gbps v.s. 25.01Gbps).

The effects of worker number are two fold: 1) the data size of each transfer is determined by the message size and number of workers. For a specific message size, the data size of each transfer is decreased as the number of workers increases. 2) the steps of transfers needed for a transaction of allreduce are increased with more workers. Thus it can be observed from the figures, the resulting bandwidths for all the three algorithms are decreasing, as the number of workers increases. Comparing to HD, Ring suffers more from increased workers since it incurs more steps of transfers and performs poorly for small data transferring. Taking the 16MiB message size as an example, the bandwidth of HDRM decreases from 37.87Gbps to 29.99Gbps, while the bandwidth of Ring decreases from 26.32Gbps to 15.13Gbps, as the number of sub-nodes increases from 8 to 64. Besides, the possibility of congestion increases as the scale of system becomes large. Therefore, the scalability of HDRM would be better than Ring.

Note that, Hierarchical-Ring[32] was proposed to improve the performance of Ringalgorithm. Although it has less step of communication and congestion, Hierarchical Ring incurs 2.5x data movement in master GPUs. As a consequent, the evaluation results present that the baseline configuration, i.e., Ringwith four NICs, outperforms the Hierarchical-Ring in all the test cases. Thus, the evaluation results on Hierarchical-Ring is omitted in this paper.

TABLE II: Results of ResNet-50 on MLPerf v0.6 [35]

Submitter	System	Chip x Count	Software	Elapsed Time(m)
Google	TPUv3.32	TPUv3 x16	TF 1.14.1.dev	42.19
NVIDIA	DGX-2H	Tesla V100 (SXM2) x16	MXNet, NGC19.05	52.74
NVIDIA	DGX-2	Tesla V100 (SXM2) x16	MXNet, NGC19.05	57.87
Ours	**EFLOPS**	**Tesla V100 (PCIe) x16**	**MXNet, Horovod**	**61.33**
Google	TPUv3.128	TPUv3 x64	TF 1.14.1.dev	11.22
Ours	**EFLOPS**	**Tesla V100 (PCIe) x64**	**MXNet, Horovod**	**15.33**

B. Real Workloads

1) MLPerf v0.6: We also investigate the overall performance of our proposed system with full-stack benchmark applications. We run MLPerf v0.6 [35] training benchmark ResNet-50 [63] with ImageNet dataset [65]. MxNet [61] and

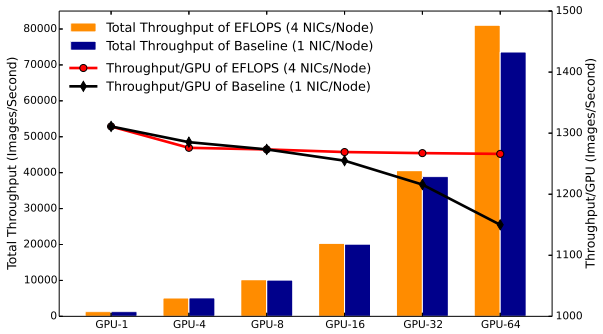


Fig. 13: MLPerf v0.6 ResNet-50 results with 1 to 64 GPUs. 4NICs/Node presents better throughput as number of GPUs increases.

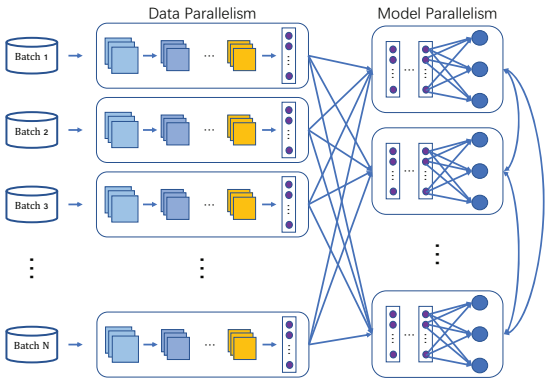


Fig. 14: The model of in-house image classification application.

Horovod [62] have been adopted for distributed training. To demonstrate the advantage of the proposed desgin, the configuration of one NIC per node is used as baseline. As shown in Figure 13, the proposed system has better throughput (images/second). In particular, the 64-GPU system achieves 10% more throughput with four NICs per node (i.e., 81,027 v.s. 73,596). In general, EFLOPS allows the system to achieve near-linear scalability since the end-to-end performance of the 64-GPU system has 15.88× speedup over the 4-GPU system. Therefore, the proposed system can achieve better scalability than the baseline.

We would also like to share the comparison with other top-ranking systems on the recently updated lists. As shown in Table II, EFLOPS takes 61.33 minutes to train ResNet-50 with 16 GPUs. When scaling the system out to 64 GPUs, the overall training time decreases *linearly* to 15.33 minutes. In general, Google TPU clusters show better performance. We believe their gain stems from the domain-specific ASICs. When comparing to NVIDIA's result, our performance is slightly lower. The reason is two fold. Firstly, the GPU we use (Tesla V100 PCIe) is about 10% slower than NVIDIA's. Furthermore, the infrastructure of interconnects is quite different. NVSwitch-based interconnect has several orders of magnitude higher bandwidth than ours (i.e., 1.2Tbps v.s. 100Gbps). Based on the analysis, we are fully confident that EFLOPS could perform better with the same hardware resources.

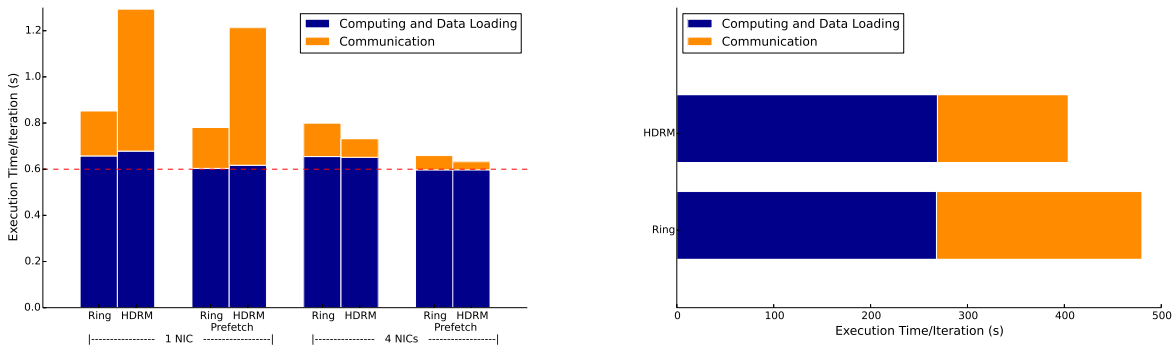
2) In-house Applications: Besides the public benchmark, an in-house image classification application is used for evaluation. This application uses 60 million images as input dataset, which could be classified into 1 million categories. The model consists of over 2 billion parameters, which is too large to be fit into a single GPU device. Therefore, we split it into two sub-models, namely *data parallelism* and *model parallelism*, as shown in Figure 14. Data parallelism can be considered as feature extractors that use ConvNets [66] to extract features from input data. It duplicates the ConvNets on multiple devices. Model parallelism is to place different parts of a fully-connected layer into multiple devices, which works together as a huge classifier. To quantify the contribution of each proposed technique to the overall performance, we run the application under different configurations, including varying the number of NICs per node, enabling data prefetching techniques or not, and adopting `Ring` or `HDRM`. The evaluation results, in term of averaged execution time of iterations, are shown in Figure 15a.

Similar to above subsection, the 1NIC/Node configuration is used as baseline to evaluate the effectiveness of congestionless servers. With 4NICs/Node, the overall performance is improved by 6.6% and 76.6% for `Ring` and `HDRM`, respectively. In particular, the communication performance is improved by 34.7% and 669%, respectively. With prefetching enabled, the overall improvement further increases by 18.4% and 91.8% while the speedup on communication is up to 1.85× and 15×. The two algorithms show different behaviors since `HDRM` is more sensitive to the congestions on NICs. `HDRM` can outperform `Ring` once those congestions are eliminated.

Surprisingly, we find that the effects of data prefetching on overall performance is pretty significant. Therefore, it is worth investigating the impact of data prefetching. Based on the results of the single-NIC system (left four bars), data prefetching can elevate the overall performance by 6.6% and 9.2% for `Ring` and `HDRM`, respectively. It increases to 15.7% and 21.3% for Four-NIC systems. Note that the contribution from data prefetching is constant, regardless of the system scale. In other words, data prefetching can always provide about $5ms$ improvement on data loading and computing, plus $2ms$ improvement on communication. In contrast, without prefetching, data loading could see some performance variations due to the long tail latency.

After the discussion about `HDRM`'s sensitivity to the congestion on NICs, we now focus on its effectiveness once the congestion is eliminated. As the right most two bars indicate, the overall performance is improved by 4.3%, and the communication performance is improved by 43.5%. The result matches the observation from synthetic workload. In general, the performance is improved by 34.8% (the first and last bars) and the communication achieves 5.57× speedup.

In addition, we have also run an in-house NLP application on EFLOPS. BERT [64] is deployed in this application, and input dataset is synthesized with native language and organized



(a) In-house image classification application

(b) NLP with BERT

Fig. 15: The evaluation results of the in-house image classification application (left) and NLP with BERT (right)

in TFRecord format. We have adopted the mixed-precision training [67], set the batch size as 52, run the application for 1000 iterations after 200 iterations of warmup, and obtained the averaged results. Same NIC and prefetching configurations have been evaluated but Figure 15b only illustrates the last two bars for brevity. As shown, the communication cost of BERT is much higher than that of image classification so that EFLOPS has even better speedup. `HDRM` can reduce the communication delay down to $135ms$ (v.s. `Ring`'s $212ms$), which refers to a 36% performance improvement on communication and 15.8% overall performance gain.

In a nutshell, it (again) turns out that various congestions in the system can incur significant performance degradation. An end-to-end solution to alleviate or even eliminate the congestions could remarkably boost the performance. For the applications with light communication traffic, like image classification, the improvement can be up to 34.8%. For applications with heavy traffic, like NLP with BERT where communication consumes nearly half of the total execution time, the performance speedup can even be more than $2\times$.

C. Simulation at Scale

So far, EFLOPS has been evaluated in a 64-GPU system. In order to know how well it performs at scale, we have built a simulation platform to support the system scaling from 64 to 1,024 nodes. We simulate both Fat-Tree and BiGraph topology with `Ring`, `HD`, and `HDRM` algorithms. We feed synthetic patterns of allreduce operations into the simulator. Figure 16a and 16b present the results of 256KiB and 256MiB messages, respectively[5]. For the same algorithm, BiGraph outperforms Fat-Tree in large scale systems. The performance gain mainly comes from the dramatic reduction of congestion. For BiGraph, `HDRM` is the clear winner. In Figure 16a, its execution time slightly increases from $130us$ (64-node) to $181us$ (1024-node). In contrast, other two algorithms see $500us$ to 10,000us increase. Also note that `Ring` has the worst scalability with small messages while `HD` has the worst scalability with large messages, which is consistent with the finding in Section III.

[5]Due to the similarity, the results of other message sizes are omitted for brevity.

D. Cost Overhead

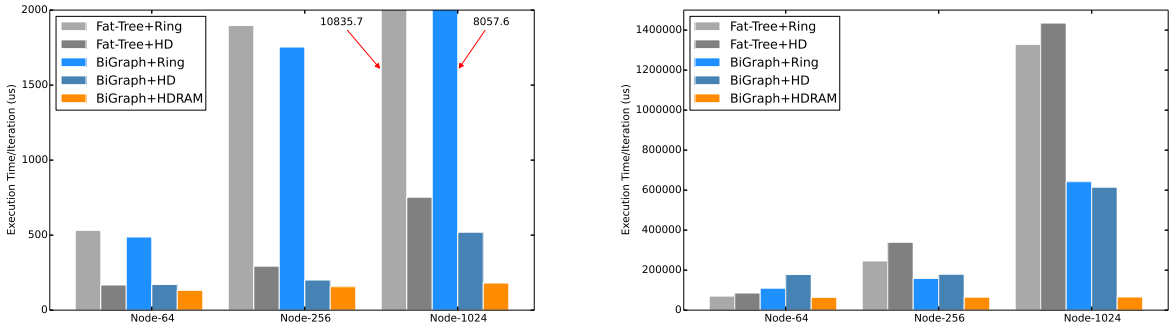
In addition, the total cost of ownership (TCO) is another important metric that should be taken into account. When comparing to the traditional single-NIC system, the number of NICs in EFLOPS is increased by four times. The BiGraph topology, however, in turn eliminates the requirement of aggregation layer switches. Putting it all together, the TCO of a 64-GPU training system increases by only 4%, which is acceptable when comparing to the achieved big performance benefits. As a result, EFLOPS is a cost efficient design.

VIII. Conclusion

As DNNs have made remarkable achievements in many applications, there is a strong demand to train the models quickly. In this work, we first analyze the different types of congestions in the traditional servers in data centers and their side effects on distributed training. To address the issue, we propose EFLOPS as a full-system solution. We first propose a new congestionless server architecture to alleviate the congestions on NICs, shared PCI root ports, as well as skewed device-to-device communications. Moreover, we propose an algorithm and system co-design to eliminate congestions with the novel BiGraph network topology and Halving-Doubling with Rank Mapping algorithm. The experimental results show that EFLOPS can achieve near-linear scalability. It also delivers significant performance gain with minimal cost overhead. The MLPerf result of EFLOPS demonstrates its competitive training performance over other top-ranking systems.

IX. Acknowledgements

We want to thank the anonymous reviewers for their valuable feedback. We sincerely appreciate the strong support from the groups and individuals involved across the company in the EFLOPS project. Special thanks to the contributors involved after the submission of this paper, including Shanyuan Gao, Xin Long, Yong Li, Lin Wang, Tianming Xu, Chen Wu, Weihua Luo, Yongqing Xi, Haiyong Wang, and Dezhong Cai.



(a) Message size = 256KiB

(b) Message size = 256MiB

Fig. 16: The simulation results for large scale system with 256KiB (left) and 256MiB (right) message size.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling.," in *Interspeech*, pp. 338–342, 2014.

[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Neural Information Processing Systems (NeurIPS)*, pp. 3104–3112, 2014.

[7] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *International Conference on Learning Representations (ICLR) Workshops*, 2016.

[8] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[9] NVIDIA, *NVIDIA Tensor Cores*. https://www.nvidia.com/en-us/data-center/tensorcore/.

[10] Medium, *Cambricon Unveils its First AI Chip for Cloud Computing*. https://medium.com/syncedreview/cambricon-unveils-its-first-ai-chip-for-cloud-computing-d3f7acdb4076.

[11] Habana, *Gaudi Training Platform White Paper*. https://habana.ai/wp-content/uploads/2019/06/Habana-Gaudi-Training-Platform-whitepaper.pdf.

[12] Huawei, *Driving AI to new horizons with an all-scenario native, full-stack solution*. https://www.huawei.com/en/about-huawei/publications/communicate/86/driving-ai-to-new-horizons.

[13] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[14] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[15] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 161–170, ACM, 2015.

[16] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 43, pp. 369–381, ACM, 2015.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pp. 243–254, IEEE Press, 2016.

[18] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pp. 267–278, IEEE Press, 2016.

[19] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pp. 14–26, IEEE Press, 2016.

[20] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pp. 27–39, IEEE Press, 2016.

[21] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: analog convnet image sensor architecture for continuous mobile vision," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pp. 255–266, IEEE Press, 2016.

[22] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[23] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ACM, 2017.

[24] Google, *Google's scalable supercomputers for machine learning, Cloud*

TPU Pods, are now publicly available in beta. https://cloud.google.com/blog/products/ai-machine-learning/.

[25] Fujistu, *Fujitsu-Developed "ABCI" System Takes 5th Place in TOP500, 8th in Green500 Supercomputer Rankings*. https://www.fujitsu.com/global/about/resources/news/press-releases/2018/0626-01.html.

[26] Facebook, *Accelerating Facebook's infrastructure with application-specific hardware*. https://code.fb.com/data-center-engineering/accelerating-infrastructure/.

[27] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Yi Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, pp. 8–20, March 2018.

[28] O. R. N. Laboratory, *Summit Supercomputer*. https://www.olcf.ornl.gov/summit/.

[29] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017.

[30] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, pp. 1:1–1:10, 2018.

[31] S. L. Smith, P. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *CoRR*, vol. abs/1711.00489, 2017.

[32] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, and G. Hu, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," in *Neural Information Processing Systems (NeurIPS)*, 2017.

[33] H. Mikami, H. Suganuma, P. U.-Chupala, Y. Tanaka, and Y. Kageyama, "Imagenet/resnet-50 training in 224 seconds," *CoRR*, vol. abs/1811.05233, 2018.

[34] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," in *Neural Information Processing Systems (NeurIPS)*, 2018.

[35] *MLPerf v0.6 Training Closed*, July 2019. Retrieved from https://www.mlperf.org. MLPerf name and logo are trademarks.

[36] NVIDIA, *NVLink and NVSwitch*. https://www.nvidia.com/en-us/data-center/nvlink/.

[37] NVIDIA, *NVIDIA DGX-2*. https://www.nvidia.com/en-us/data-center/dgx-2/.

[38] C. Clos, "A study of non-blocking switching networks," *The Bell System Technical Journal*, vol. 32, pp. 406–424, March 1953.

[39] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *International Journal of High Performance Computing Applications*, vol. 19, pp. 49–66, Feburary 2005.

[40] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. F. Keutzer, "Firecaffe: near-linear acceleration of deep neural network training on compute clusters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2592–2600, 2016.

[41] M. Burtscher and P. Ratanaworabhan, "Fpc: A highspeed compressor for double-precision floating-point data.," in *IEEE Trans. Comput.,*, vol. 58, pp. 18–31, 2009.

[42] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns.," in *Interspeech*, 2014.

[43] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning.," in *Advances in Neural Information Processing Systems*, 2017.

[44] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," *SOSP*, 2019.

[45] S. H. Hashemi, S. A. Jyothi, and R. H. Campbell, "TicTac: Accelerating Distributed Deep Learning with Communication Scheduling," *SysML Conference*, 2019.

[47] PCI-SIG, *PCI Express Speficication*. https://pcisig.com/specifications.

[46] A. Jayarajan, J. Wei, G. A. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed dnn training," *SysML Conference*, 2019.

[48] Intel, *Intel Xeon Processor Scalable Family Datasheet*. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-scalable-datasheet-vol-1.pdf.

[49] M. Martinasso, G. Kwasniewski, S. R. Alam, T. C. Schulthess, and T. Hoefler, "A pcie congestion-aware performance model for densely populated accelerator servers," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016.

[50] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*, 1988.

[51] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," in *IEEE/ACM Transactions on networking*, 1993.

[52] K. Ramakrishnan and S. Floyd, "The addition of explicit congestion notification (ecn) to ip," in *RFC2481*, 2001.

[53] S. Floyd and T. Henderson, "The newreno modification to tcp's fast recovery algorithm," in *RFC3782*, 1999.

[54] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *SIGCOMM*, 2010.

[55] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[56] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of AFIPS Conference*, AFIPS '67 (Spring), pp. 483–485, ACM, 1967.

[57] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, pp. 117–124, February 2009.

[58] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *11th European PVM/MPI Users' Group Meeting*, 2004.

[59] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2015. Software available from tensorflow.org.

[60] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Neural Information Processing Systems (NeurIPS)*, 2017.

[61] A. Incubator, *MXNet: A Scalable Deep Learning Framework*. https://mxnet.apache.org/.

[62] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[64] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, IEEE, 2009.

[66] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[67] NVIDIA, *NVIDIA Apex: Tools for Easy Mixed-Precision Training in PyTorch*. https://devblogs.nvidia.com/apex-pytorch-easy-mixed-precision-training/.