Communication Algorithm-Architecture Co-Design for Distributed Deep Learning

Jiayi Huang
UC Santa Barbara
jyhuang@ucsb.edu

Pritam Majumder
Texas A&M University
pritam2309@tamu.edu

Sungkeun Kim
Texas A&M University
ksungkeun84@tamu.edu

Abdullah Muzahid
Texas A&M University
abdullah.muzahid@cse.tamu.edu

Ki Hwan Yum
Texas A&M University
yum@cse.tamu.edu

Eun Jung Kim
Texas A&M University
ejkim@cse.tamu.edu

***Abstract*—Large-scale distributed deep learning training has enabled developments of more complex deep neural network models to learn from larger datasets for sophisticated tasks. In particular, distributed stochastic gradient descent intensively invokes *all-reduce* operations for gradient update, which dominates communication time during iterative training epochs. In this work, we identify the inefficiency in widely used all-reduce algorithms, and the opportunity of algorithm-architecture co-design. We propose MULTITREE all-reduce algorithm with topology and resource utilization awareness for efficient and scalable all-reduce operations, which is applicable to different interconnect topologies. Moreover, we co-design the network interface to schedule and coordinate the all-reduce messages for contention-free communications, working in synergy with the algorithm. The flow control is also simplified to exploit the bulk data transfer of big gradient exchange. We evaluate the co-design using different all-reduce data sizes for synthetic study, demonstrating its effectiveness on various interconnection network topologies, in addition to state-of-the-art deep neural networks for real workload experiments. The results show that MULTITREE achieves 2.3× and 1.56× communication speedup, as well as up to 81% and 30% training time reduction compared to ring all-reduce and state-of-the-art approaches, respectively.**

***Index Terms*—distributed deep learning, data-parallel training, all-reduce, interconnection network, algorithm-architecture co-design**

I. INTRODUCTION

The onset of the big data era and rapid advances of accelerator architectures have enabled deep learning applications to achieve superhuman accuracy on complex real-world problems, such as image recognition, natural language processing, and autonomous driving. State-of-the-art DNN models such as GPT-3 [1] have hundreds of billions of parameters, requiring trillions of compute operations and hundreds of gigabytes of storage and massive bandwidth. Recent work projects that orders of magnitude growth of dataset and model size are required to exceed human-level accuracy, which can take weeks to train a single epoch for language modeling [2]. As data keep exploding and DNNs evolve to be larger and deeper, it is crucial to provide scalable solutions to fulfill the trend in computing requirements.

This research was supported in part by NSF Grant #1931078.

To this end, grids of specialized accelerators have been designed and deployed to train DNN models in a parallel and distributed manner [3], [4]. In particular, data-parallelism, as the easiest model of parallel and distributed computing, has been widely used in large-scale DNN training [5], [6]. Stochastic gradient descent (SGD) is a typical optimization algorithm to improve DNN accuracy through iterative training, which intensively invokes *all-reduce* communication. As the dominant component of communication, all-reduce can stall the computations of the next training epoch significantly. Thus, all-reduce can quickly become a bottleneck for large scale distributed training [7].

Several communication algorithms have been proposed for all-reduce operation [8]–[11]. Baidu Research implemented a bandwidth-optimal ring all-reduce algorithm [9], [12], which has been later included in NVIDIA Collective Communication Library (NCCL) [13] and other popular deep learning frameworks [14], [15]. However, ring all-reduce suffers from long latency and may have low resource utilization in certain network topologies, for instance, only 25% link utilization rate in a 4×4 2D Torus network. Several attempts have been made to improve all-reduce latency by reducing the algorithmic steps [10], [11], [16]. Halving-doubling reduces the latency through recursive distance doubling and halving in the reduce-scatter and all-gather phases, respectively [11]. Double binary tree (DBTree), which is also implemented in NCCL, improves the latency through two-tree reduction and broadcast [10], [16]. These two algorithms perform better than ring all-reduce for short to medium messages. However, for large messages, they can lead to significant network congestion since their communication patterns map poorly on to the physical network topology, turning out to be worse than rings[1] [11], [19]. Therefore, it is crucial to consider the physical network topology for all-reduce algorithm design with proper message scheduling to achieve low latency for short to medium messages and contention-free communication for large data sizes.

[1]NVIDIA NCCL enables double binary tree when message size is small while disables it and uses ring all-reduce when message is larger than a threshold, which requires tuning for different systems [17], [18].

TABLE I: Comparisons of All-Reduce Algorithms.

Algorithms	Small data	Large data		Applied Well on
	Latency	Bandwidth	Contention	Various Topologies
Ring [9], [12]	high	optimal	none	✓
DBTree [10], [16]	low	optimal	high	× (Topo-oblivious)
2D-Ring [28]	low	sub-optimal	none	× (2D Torus/Mesh)
HDRM [29]	low	optimal	none	×(BiGraph)
MULTITREE	low	optimal	none	✓

Recently, dedicated networks with accelerator pods have been deployed to accelerate emerging deep learning applications, such as Cloud TPU [4] and Catapult [3]. While computation acceleration has been significantly studied [20]–[27], communication specialization with architecture-algorithm co-design is still in its infancy [28], [29]. Ying *et al.* adopted 2D-ring all-reduce for the 2D Torus network in TPU clusters to fully utilize the links and reduce communication steps [28]. Although achieving full link utilization, its 2D nature increases the amount of communicated data, which can be double the optimal communicated data as the network scales out. For a 2D $N \times N$ Torus network, 2D-ring transmits $2N(N-1)$ data while flat ring communicates N^2-1 data. More recently, Alibaba proposed the EFLOPS training platform by co-designing algorithm and system with a new server architecture [29]. It extends the halving-doubling algorithm with rank mapping (HDRM) on a two-stage fully connected BiGraph topology to avoid contention, showing promising potential for the co-design approach. However, it is not trivial to scale due to its full connections among switches. These algorithms are limited to specific topologies (2D Torus and BiGraph) and do not generalize to other network topologies. With the trend for larger and deeper DNN models, more accelerator grids are deployed for large-scale distributed training. Therefore, more scalable solutions are required to work in synergy with various topologies that can practically interconnect a large number of nodes [4]. Moreover, communication acceleration through specialization is urgently needed to keep up with the computation throughput. In addition, the lack of hardware support for coordination and communication scheduling may miss potential optimization opportunities to further improve performance. Furthermore, the fine-grained flow control and arbitration designed for general purpose networks can be inefficient to support such large gradient exchanges, resulting in extra performance and significant energy/power overhead. Table I summarizes the comparisons among these works.

In this work, we co-design an all-reduce communication algorithm and interconnection architecture to support efficient and scalable all-reduce operation. We propose MULTITREE, a scalable topology-aware all-reduce algorithm that is applicable to various topologies. MULTITREE couples tree construction and message scheduling with topology and global link utilization awareness to build trees from roots in a top-down fashion. It leverages the insight that *tree levels closer to the roots are more sparse and tree levels closer to the leaves are denser.* Based on this, MULTITREE moves more communication closer to the roots to make communication closer to the leaves sparse so that communications are balanced in all levels of the trees. Moreover, we co-design the network interface according to the proposed communication algorithm and to facilitate the all-reduce schedule management to achieve contention-free all-reduce. We also simplify the flow control and arbitration to exploit the characteristics of large gradients in all-reduce operations. As a result, MULTITREE tackles the limitations in previous work, as summarized in Table I.

In summary, the contributions of this paper are as follows.

- We identify the inefficiency in the state-of-the-art all-reduce algorithms, and co-design all-reduce algorithm and interconnect hardware for large gradient exchange.
- We propose MULTITREE, an all-reduce algorithm that is applicable to various interconnect topologies and couple tree construction and communication scheduling, with topology and global link utilization awareness, to efficiently coordinate concurrent reduction/broadcast trees.
- We augment the network interface to support hardware based scheduling for MULTITREE and facilitate the lock-step communications in the schedule, while simplifying the flow control dedicated for large gradient all-reduce.
- Our evaluations using synthetic messages and state-of-the-art DNNs show that MULTITREE greatly improves scalability over prior works, and achieves 2.3× and 1.56× communication speedup, as well as up to 81% and 30% training time reduction compared to ring all-reduce and the state-of-the-art approach [28], respectively.

The rest of the paper is organized as follows: §II introduces the background and motivation. §III presents the proposed MULTITREE all-reduce algorithm followed by the co-designed architecture detailed in §IV. The methodology is described in §V and the evaluation is presented in §VI, respectively. Further discussions are outlined in §VII followed by more related work in §VIII. Finally, we conclude the paper in §IX.

II. BACKGROUND AND MOTIVATION

In this section, we first introduce the background of data-parallel deep neural network training followed by all-reduce communication in distributed stochastic gradient descent synchronization.. Then, we motivate this research by outlining the limitations of existing all-reduce algorithms.

A. Data-Parallel Deep Neural Network Training

DNN training is usually done using stochastic gradient descent where each training sample goes through forward propagation, gradient calculation followed by backward propagation. Backward propagation uses the gradient to update weights of the DNN model in order to minimize a loss function. To make training faster, mini-batch is used where there is one pass of weight update for each mini-batch of training samples. It is a daunting task to train large DNNs with huge amounts of data. Thus, distributed training is performed on multiple compute nodes. Each compute node may be equipped with GPUs and accelerators. This creates the challenges regarding resource usage, communication bandwidth provisioning, and trade-off between computation and

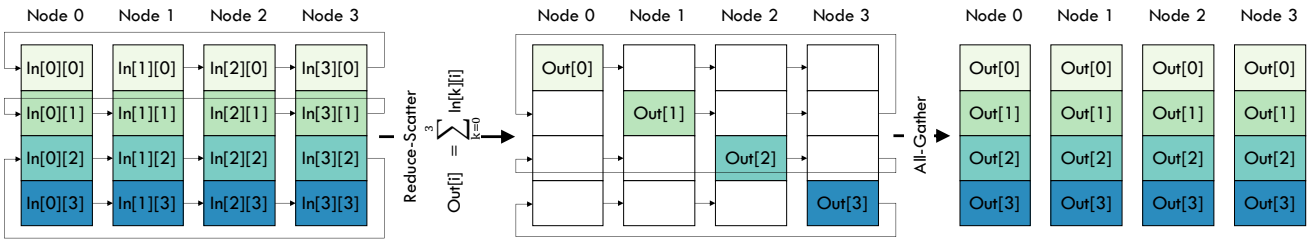


Fig. 1: Reduce-scatter and all-gather in ring all-reduce.

storage [30], [31]. Different parallel strategies have been used to enable scalable and efficient distributed training.

Data parallelism is the most common way for distributed DNN training where a non-overlapping set of training samples are distributed to different compute nodes. Each node calculates gradients based on its own training set. Gradients are then aggregated to update weights using either a centralized or a decentralized approach. The centralized approach relies on a parameter server where each node periodically reports its computed parameter updates to a (set of) parameter server(s) [8]. However, parameter servers are not efficient in terms of bandwidth and latency for larger DNNs. An alternative is the decentralized approach where compute nodes exchange parameter updates via an all-reduce operation, where the all-reduce algorithm plays an important role. A widely used one is ring all-reduce [9], [12] that only requires a tree topology to achieve no contention and optimal bandwidth, where a tree topology is typically embedded in any network topology. However, it is not latency-optimal [12].

B. All-Reduce for Distributed Stochastic Gradient Descent

Baidu popularized ring all-reduce using a sequence of reduce-scatter followed by all-gather operations [9], [32]. Reduce-scatter and all-gather operations are further optimized to exploit the hierarchical nature of communication bandwidths of heterogeneous network architecture [33].

Fig. 1 shows an example. Let us assume that each row represents one segment of tensors with segment 0 being the top row and segment 3 being the bottom one. Each node forms a ring with the next node. Reduce-scatter is done on segment 0 starting from Node 1. In the first iteration, segment 0 is sent from Node 1 to Node 2 where the tensors are aggregated. Thus, two out of four sets of tensors are aggregated in the first iteration. In the second iteration, segment 0 is sent from Node 2 to Node 3 and in the third iteration, segment 0 is sent from Node 3 to Node 0. Thus, after 3 iterations, all tensors of segment 0 are aggregated to Node 0. Similarly, segment 1 starts from Node 2 and after 3 iterations, gets reduced to Node 1. Segments 2 and 3 end up getting reduced to Nodes 2 and 3, respectively. Thus, it takes 3 iterations for reduce-scatter.

After the sequence of reduce-scatter operations, all-gather operations are done similarly. In the first iteration, segment 0 is sent from Node 0 to Node 1. Now, Node 1 has two out of four segments (namely segments 0 and 1). Similarly, at the end of the first iteration, other nodes end up having 2 segments. In the second iteration, segment 0 is sent from Node 1 to Node 2 and subsequently from Node 2 to Node 3 in the third iteration. Thus, after 3 iterations, all nodes will end up having all 4 aggregated segments.

C. Motivation

Widely used ring all-reduce has been proved bandwidth-optimal [12], which makes it suitable for large gradient exchanges [9], [13], [34]. However, it faces link under-utilization in certain topologies such as Torus and Mesh. Furthermore, it suffers from long latency as the system scales out. Several attempts have been made to improve link utilization and latency [10], [16], [28]. 2D-ring all-reduce utilizes all the links and reduces the communication steps in 2D Torus and Mesh networks [28], but it transmits twice the amount of data compared to bandwidth-optimal algorithms. For instance, 2D-ring communicates $2N(N-1)$ data while ring all-reduce communicates N^2-1 data in a 2D $N \times N$ Torus network. On the other hand, the double binary tree algorithm builds two logical binary trees to reduce latency for small to medium messages [10], [16]. It constructs the two trees in a way such that the leaf nodes in one tree are the internal nodes in the other tree. Therefore, each tree can take half of the data and all the nodes can send and receive data simultaneously, outperforming single-tree all-reduce. It better utilizes the end-node bandwidth and works well on networks with all-to-all like topology for small to medium messages. For large messages, it can experience significant contention since the two trees are not mapped well on the physical network topology, especially severe on unfriendly topology such as Torus [19]. The recently proposed EFLOPS extends halving-doubling by mapping the ranks to the nodes to achieve contention-free communication for large datasets [29]. Since each communication pair always involves one node connected with an upper switch and one node connected with a lower switch, it never exploits the one-hop distance between nodes connected to the same switch, failing to expedite latency-sensitive communications for small messages. Furthermore, these algorithms are not general to apply to different network topologies while achieving good performance for both latency and bandwidth.

Typical interconnection networks for general-purpose communications use fine-grained flow control that well supports short messages. However, it can generate many small packets for large gradients, which can lead to extra bandwidth and arbitration overheads. To mitigate such an overhead, flow control can be streamlined to take advantage of large gradients. In the conventional packet-based flow control, a packet consists of a

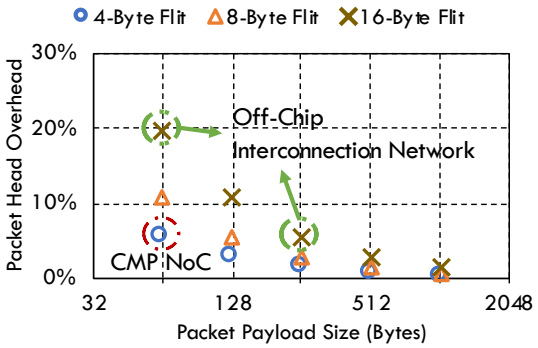


Fig. 2: Packet head flit bandwidth overhead.

few flits, including a head flit for metadata and body/tail flits for payload, where the head flit incurs bandwidth overhead. For on-chip networks, the payload of a packet is a cache line whereas, for off-chip networks, payload size varies from 64 to 256 bytes with 16-byte flits, incurring 6%–25% bandwidth overhead, as shown in Fig. 2. In distributed DNN training, streams of *consecutive large* gradients flow from one node to the other using many small packets, following the same route with consecutive addresses. So the head flits of these consecutive packets contain redundant information, leading to unnecessary bandwidth overhead. Thus, flow control can be simplified to exploit this distinct characteristic.

III. MultiTree All-Reduce Algorithm

In this section, we first explain the rationales behind the MultiTree approach. Then, we illustrate the main idea with an example followed by its algorithm.

A. Rationales and Insights

1) Spanning Trees Instead of Rings: In reduce-scatter and all-gather phases of all-reduce, each node leads a reduction and a broadcast of one chunk of data. In ring all-reduce, each node communicates a chunk of data in a unidirectional ring, which takes $(n-1)$ steps in both phases for n nodes. If each such communication can take place in a tree structure, it can reduce the algorithmic steps to $2\log_k n$ with a k-ary tree for n nodes. Thus, the proposed algorithm is not only bandwidth-optimal but also reduces latency by constructing multiple trees instead of rings, thereby improving all-reduce scalability.

2) Topology Awareness: If trees are constructed without considering network topology and link utilization, it may lead to even worse performance than ring all-reduce, especially when multiple trees contend for the same link at the same time without careful scheduling. Furthermore, *tree levels closer to leaves are denser than tree levels closer to roots*. Thus, when reducing from leaves to roots, the reduce-scatter phase can experience dense to sparse communications, leading to high contention near leaf nodes. MultiTree exploits this insight to combine message scheduling and tree constructions, with topology and link utilization awareness to schedule more communication near the roots to sparsify communication near leaves. In addition, instead of constructing the trees one by one, MultiTree builds them concurrently, generating balanced trees with global coordination.

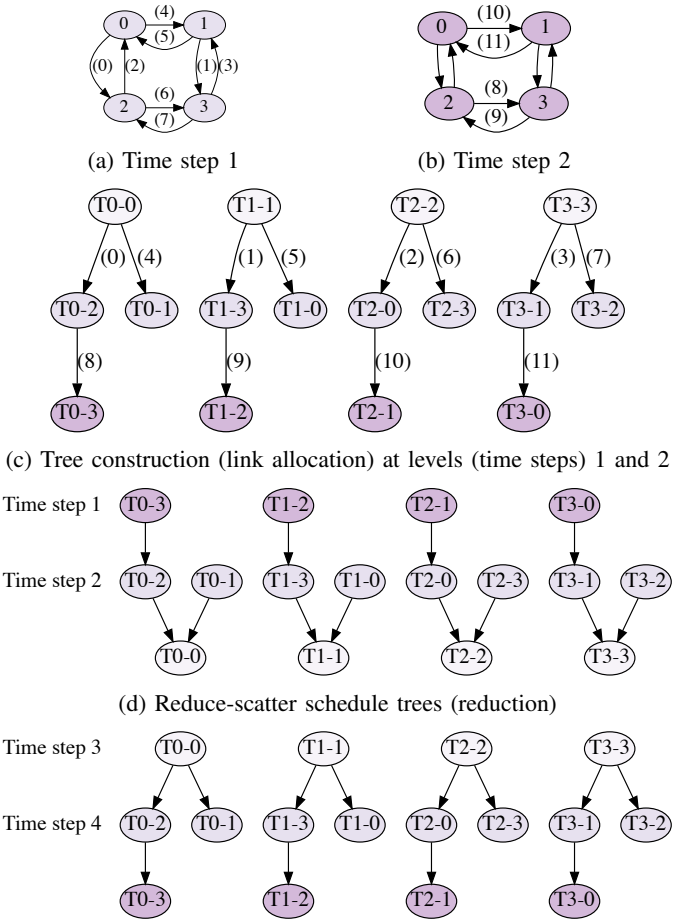
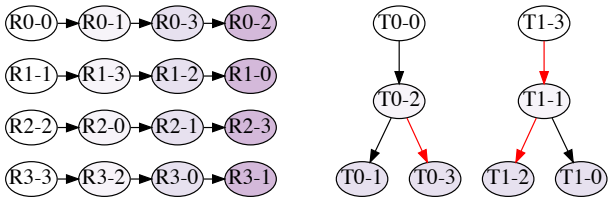


Fig. 3: MultiTree construction with link allocation and scheduling for all-reduce communication of a (2×2) Mesh network. Node `n` in tree `T` is denoted as `T-n` and label `(i)` of an edge is the allocation sequence of that link while label `t` of an edge is the communication time step between the two nodes: (a) link allocation sequence of the topology graph for level 1 (time step 1); (b) when no more links are available for the predecessor levels 0 and 1, a new link topology graph is used for allocation for level 2 (time step 2); (c) the tree construction process indicated by edge labels; (d) the constructed reduce-scatter schedule trees and (e) all-gather schedule trees.

B. Main Idea

Given a network $G(V, E)$ with nodes V and edges E, finally $|V|$ spanning trees are created. To move more communications near the roots, MultiTree builds the trees from roots in a top-down approach, making the predecessor levels denser and communications balanced across the tree levels. During tree construction, for each time step (tree level), a topology graph is used to allocate links to connect remaining nodes to the spanning trees, and the allocated links are removed from the graph. When there are no more available links to connect remaining nodes to any of the trees, a new topology graph is used for the next time step (tree level).



(a) Ring-based all-gather (b) Double binary tree broadcast

Fig. 4: All-gather or broadcast phase of the ring-based and double binary tree algorithms: (a) Ring-based all-gather with (Ri-n) for a node n involving in data chunk i, and (b) double binary tree broadcast with black/red edge color indicating communication at even/odd step.

We illustrate the main idea by walking through an example[2] that constructs schedule trees for a (2×2) Mesh network as shown in Fig. 3. Fig. 3a and 3b show the topology graphs that are used to facilitate tree constructions for time steps 1 and 2, respectively. The edge label (i) in Fig. 3a, 3b and 3c indicates the global link allocation sequence that connects a node to its parent during the tree construction. Fig. 3c shows all the trees and their construction sequences, where the trees take turns to add one node at a time, sustaining tree balance. At sequence number 7, after the last edge $(3 \rightarrow 2)$ is added to connect nodes 2 ((T3-2)) and 3 ((T3-3)) in tree 3, the topology graph for time step 1 in Fig. 3a runs out of all the edges. Then, a new topology graph in Fig. 3b is used to start time step 2, which creates a new level for the trees. (lines 4–14 of the Algorithm 1 in §III-C). These newly constructed trees are used to build the reduce-scatter schedule trees and finally, are adjusted to generate all-gather schedule trees, as shown in Fig. 3d and 3e, respectively. Note that the trees are well balanced and symmetric in shape, but not necessarily structurally symmetric. Structural symmetry requires special representation of each node with respect to the remaining network and only applies to specific symmetric networks. Moreover, for networks like a (4×4) Mesh where the longest distance from a source node varies depending on its position, the trees are asymmetric with different heights.

Fig. 4 shows the all-gather and broadcast schedules for ring and double binary tree (DBTree) on the same network, respectively. Compared to MULTITREE, ring needs one more step which leads to longer latency. It also shows rings can be considered as *unary* spanning trees. Fig. 4b shows the two trees in DBTree. Although it has the same *logical* height as MULTITREE, its *physical* height is deeper since the connection between nodes 1 and 2 crosses two hops due to the mismatch of tree structure and physical topology. Such a mismatch is even more severe in larger networks. In addition, DBTree schedules the communications in even/odd steps (black/red color) such that a node never receives or sends data simultaneously in both trees, which can lengthen the completion time. Note that each edge in MULTITREE maps to a physical link, which is only one-hop distance.

[2]For demonstration purposes, we use a 2×2 Mesh network that is too small to show the benefit of MULTITREE. Although a larger network can show the advantages of MULTITREE, we cannot accommodate it due to page limit.

Algorithm 1: MULTITREE All-Reduce Algorithm.

Input: $topology\_graph\ G(V, E)$
Output: $reduce\_scatter\_schedule,\ allgather\_schedule$

```
// Initialization
1  for each node i ∈ V of graph G(V, E) do
2      Tree T_i adds node i to tree T_i as root;
3  t = 0;
   // Compute all-gather schedules
4  while not all trees completed do
       // Start a new time step t with a new G
5      t = t + 1;
6      G'(V', E') = G(V, E);
       // Add new nodes to trees and schedule
       // communications for this time step
7      while E' has free edges to add new nodes do
           // Trees take turns for balancing
8          Select next tree T by root ID in ascending order;
9          for p ∈ T's nodes added by previous time steps do
10             if there is an edge (p → c) ∈ E' then
11                 Add node c to T and connect to p;
12                 Remove edge p → c from E';
                   // Schedule message p → c at t
13                 Add (p → c, t) to T's allgather_schedule;
14                 break;
15 Calculate total time steps tot_t = t;
   // Compute reduce-scatter schedule, which
   // is the reverse of all-gather
16 for (p → c, t') ∈ allgather_schedule of each tree T do
17     Add (c → p, tot_t − t' + 1) to T's
        reduce_scatter_schedule;
       // Adjust all-gather schedule
18     Replace (p → c, t') with (p → c, tot_t + t');
```

C. Algorithm Design

More formally, MULTITREE is presented in Algorithm 1. For ease of understanding, we describe it for direct networks, and provide the steps to extend for switch-based networks.

1) Algorithm Description: The algorithm initializes a tree for each node in the network as the root and the time step t (lines 1–3). Then it starts to construct the schedule trees for the all-gather phase (broadcast) instead of reduce-scatter, since it is more natural for the top-down approach to start from the root (lines 4–14). For every new time step t, a full topology graph $G'(V', E')$ is used, whose edges are removed while adding new nodes to the trees. During this time step t, trees take turns to add one node c to connect to a predecessor node p added in previous time steps. Then the edge $p \rightarrow c$ is removed from the topology graph and scheduled for communication at the current time step t. Note that trees alternate by root ID in ascending order for simplicity, which works fine in most cases, especially for symmetric networks like Torus. For asymmetric or irregular networks, trees with larger remaining height can be prioritized so that communication on the longest path is scheduled earlier. At line 9, nodes are examined breadth-first

in their order of adding to the tree by previous time steps so as to make the predecessor levels denser. For selecting a neighbor of p (line 10), it first checks the neighbors in Y dimension then in X dimension for Torus and Mesh networks. Other structural information can be used for asymmetric and irregular networks, which we leave for future study. When the topology graph runs out of edges to connect remaining nodes to any of the trees, it starts a new time step and repeats the same link allocation procedure until all the all-gather schedule trees are completed. After all-gather schedule trees are constructed, they are used to construct reduce-scatter trees and adjusted for communication time step (lines 16–18). Since reduce-scatter goes in the opposite direction with respect to all-gather communication, the algorithm simply reverses the communication pairs of all-gather schedule trees with adjusted time steps. The all-gather schedules are also adjusted in time to run after reduce-scatter schedules. In static systems, the algorithm only needs to run once and can be used for any DNN workloads. In dynamic and shared systems, it runs every time a new set of nodes is allocated for the workloads.

2) Complexity Analysis: The most expensive part of the algorithm is the loop for all-gather schedule tree constructions (lines 4–14). Let us consider a topology graph $G(V, E)$. The core part of adding new nodes to schedule trees is from lines 9–14. To add a new node, the algorithm checks whether the already added nodes of that tree still have edges connected to a pending node. In the worst case, it may check all the edges of the graph, which is $|E|$. In total, we have $|V|$ trees and each tree has $|V|$ nodes. So the worst case is $\mathcal{O}(|V|^2|E|)$.

3) Indirect Networks Support: In switch-based networks, only some switches are connected to end nodes, other switches connect with each other to form the indirect network. In Algorithm 1, the topology graph $G(V, E)$ is the adjacency lists of `switch-to-switch` connections in a direct network, where each switch is attached with a node. In order to support indirect networks, we extend $G(V, E)$ with additional `node-to-switch` and `switch-to-node` connection lists. To find an available child c for a node p, it follows breadth-first search on these three topology components as described in the following steps:

(1) Get p's attached switch sw_0 from its `node-to-switch` list.
(2) When multiple nodes are attached to the same switch, check whether sw_0's `switch-to-node` has connections to connect with p. If there is an available connection, pick a node as c and remove one connection ($p \rightarrow sw_0$) from p's `node-to-switch` list and one connection ($sw_0 \rightarrow c$) from sw_0's `switch-to-node` list, then return. If there is no available connection, go to step 3.
(3) Get the neighbor switch sw_1 from the `switch-to-switch` list of sw_0. Repeat the same process as step 2 with sw_1 until a node c is found or no connection is available. In this case, if a node is found, besides the connections removed in step 2, connections in traversed `switch-to-switch` lists should also be removed for the allocated links.

All-Reduce Schedule Table Entry

Op	FlowID	Parent	Children	Step	Start Addr	Size

Op: `Reduce, Gather, NOP`
FlowID: `tree ID`

Accelerator 0

Op	FlowID	Parent	Children				Step
Reduce	3	1	nil	nil	nil	nil	1
Reduce	1	1	nil	nil	nil	nil	2
Reduce	2	2	1	nil	nil	nil	2
Gather	0	nil	1	2	nil	nil	3
Gather	2	2	1	nil	nil	nil	4

Accelerator 1

Op	FlowID	Parent	Children				Step
Reduce	2	0	nil	nil	nil	nil	1
Reduce	0	0	nil	nil	nil	nil	2
Reduce	3	3	0	nil	nil	nil	2
Gather	1	nil	0	3	nil	nil	3
Gather	3	3	0	nil	nil	nil	4

Accelerator 2

Op	FlowID	Parent	Children				Step
Reduce	1	3	nil	nil	nil	nil	1
Reduce	3	3	nil	nil	nil	nil	2
Reduce	0	0	3	nil	nil	nil	2
Gather	1	nil	0	3	nil	nil	3
Gather	0	0	3	nil	nil	nil	4

Accelerator 3

Op	FlowID	Parent	Children				Step
Reduce	0	2	nil	nil	nil	nil	1
Reduce	2	2	nil	nil	nil	nil	2
Reduce	1	1	2	nil	nil	nil	2
Gather	3	nil	1	2	nil	nil	3
Gather	1	1	2	nil	nil	nil	4

Fig. 5: All-Reduce schedule tables for the example in §III-B (The `Start Addr` and `Size` fields are omitted in the tables for brevity).

IV. Architectural Supports

In this section, we outline the co-designed communication architecture and the specialized flow control mechanism for MultiTree all-reduce operations.

A. All-Reduce Schedule Management

We co-design the network interface (NI) to facilitate MultiTree all-reduce scheduling. Algorithm 1 constructs trees for each data chunk. These tree schedules can be converted into schedule tables (one table per node). Fig. 5 shows the all-reduce schedule tables for the example in §III-B. Each table entry consists of an `Op` filed for the opcode, a `FlowID` field for the tree flow (tree ID), a `Parent` and `Children` fields for the dependencies in this tree flow[3]. In addition, the `Step` field indicates the time step in which this communication should be initiated. The `Start Addr` and `Size` fields are for the *starting address* and the *size for the gradient message*, respectively.

There are three opcodes for all-reduce, namely, `Reduce`, `Gather`, and `NOP`. During `Reduce` operation, communication happens from the leaf to the root. Each internal node of the trees must receive `Reduce` before communicating to its parent. For example, in Fig. 5, accelerator 0 can send a `Reduce` to its parent (accelerator 1) for tree flow 3, because it is the leaf node at step 0 in tree 3. The last `Reduce` of accelerator 0 should not be sent to the parent (accelerator 2) until it receives a dependent `Reduce` from its child (accelerator 1) in the flow tree 2. On the other hand, during `Gather` operation, a node sends messages to the children after receiving a `Gather` from the parent unless the node is the root of the tree.

We also provide a `NOP` to maintain the communication of different time steps in a lockstep manner. Link contention can happen without proper scheduling of messages among the trees. This is more frequent in topologies that generate imbalanced trees, such as large-scale Mesh, and can limit the improvement or even degrade performance by destroying the scheduling. Therefore, a mechanism to maintain the communication in a lockstep fashion is needed to achieve the

[3]The size of the `Children` field is calculated as the bandwidth ratio between the network interface and a network link bandwidth.

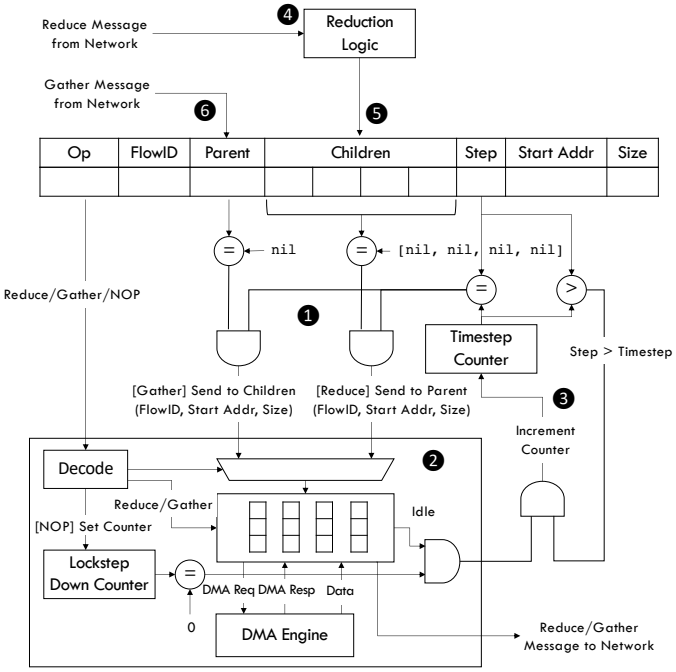


Fig. 6: Architecture of all-reduce schedule management.

best performance. One option is to use some simple message passing scheme but that can introduce additional coordination overhead, which can be very high especially for small messages. Therefore, we propose a lockstep mechanism for implicit coordination by exploiting the static communication patterns in all-reduce. Given the message size, the step time is estimated as the serialization latency assuming no contention[4]. When a `NOP` is inserted, the all-reduce injection is forced to stall for the estimated step time. Although `NOP` may leave links under-utilized, based on our observations, it only happens in irregular networks and at the leaves of the trees, while other time steps can fully utilize the links[5]. Pruning and adjusting the trees may help in these cases, we leave it for future exploration. In addition, the estimated lockstep mechanism does not require a global synchronization across all the NIs. When the data size is small, minor variation in the same time step in different nodes has minimum impact as bandwidth is not the bottleneck. When the data size is large, the long serialization latency becomes dominant, making the small clock variance insignificant.

Fig. 6 depicts the architecture for all-reduce schedule management and injection regulation. It includes an all-reduce schedule table, a timestep counter, a decoder, a lockstep down-counter and the conventional NI facilities. Upon an all-reduce operation, the schedule table is initialized; the timestep and lockstep counters are reset by the processor to configure the scheduling. During all-reduce, the head entry of the table is inspected (❶). If the `Step` is the same as the timestep counter value and the children (for `Reduce`) or parent (for `Gather`) dependencies are satisfied, the operation is issued to send the messages. Then, the `Op` is decoded to decide the corresponding action (❷). If it is a `NOP`, the lockstep counter is set and starts down counting for an estimated time step. If it is a `Reduce/Gather`, the `Start Addr` and `Size` are used to request the DMA engine for bulk data transfer. When the data comes back, the `FlowID` is encapsulated with other address information in the data packet to start communication. When the lockstep counter is zero and the all-reduce units are idle, the timestep counter is incremented if the next operation in the schedule table is for the next step (❸). Upon receiving `Reduce` messages, it is issued to the reduction logic for aggregation (❹). Once the aggregation for `Reduce` of the current step is finished, they are used to clear the dependencies of future `Reduce/Gather` (❺). When a `Gather` is received, it is directed to the schedule table to clear the parent dependence for the upcoming `Gather` (❻).

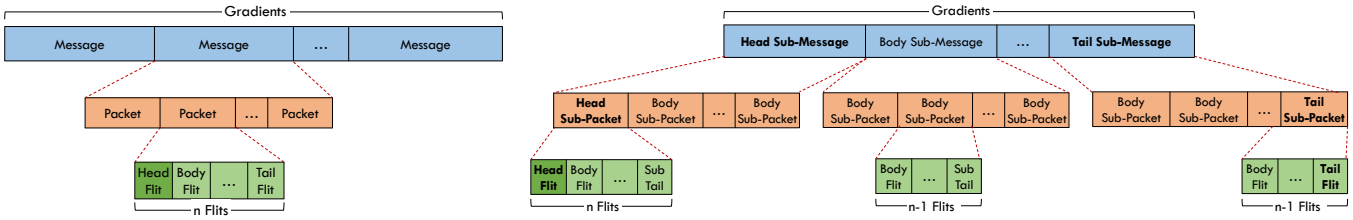
B. Message-based Flow Control for Big Gradient Exchanges

Unlike general purpose applications, all-reduce communication in data-parallel DNN training has a relatively fixed traffic pattern. With a specific all-reduce algorithm, the communication pattern is known in advance for a training task. For example, MULTITREE constructs schedule trees before training starts. This prior knowledge can be leveraged for simpler control and arbitration in hardware, thereby simplifying logic and improving energy efficiency. MULTITREE algorithm aims to coordinate among the trees with a global view, where less dynamism in interconnection networks helps maintain the communication schedules, thereby keeping concurrent communications progressing at a similar rate. In addition, the long traffic (between a communicating pair) for all-reduce of large gradients unnecessarily incurs bandwidth overhead of massive number of packet head flits. To optimize these aspects, we revisit the traditional flow control techniques and redesign them specifically for all-reduce communication.

Fig. 7a shows a commonly used packet-based switching mechanism, where large gradients are divided into many messages. Each message is partitioned into multiple packets. Each packet consists of a head flit and body/tail flits. The highlighted head flits consumes bandwidth and incurs extra control such as routing and arbitration, causing extra delay and energy consumption. On the other hand, we adapt a message-based approach to reduce such overheads, as shown in Fig. 7b. Instead of having a fixed message size, we take the whole chunk of gradients as a message, which can be further converted to many sub-messages starting with a head sub-message and ending with a tail sub-message. Each sub-message is divided into sub-packets, where the first sub-packet of the head sub-message is a head sub-packet, which behaves as the head of the large gradient message. The last sub-packet of the tail sub-message is the tail sub-packet to end the gradient message. Similarly, the sub-packets are partitioned into flits. Unlike conventional packet-based switching, body

[4]The step time is estimated as the number of flits (`num_flits`) for the per-step data chunk if the NI buffers can hold it completely. Otherwise, it is estimated as `num_flits` subtracting the NI buffer size (translating to flit size).

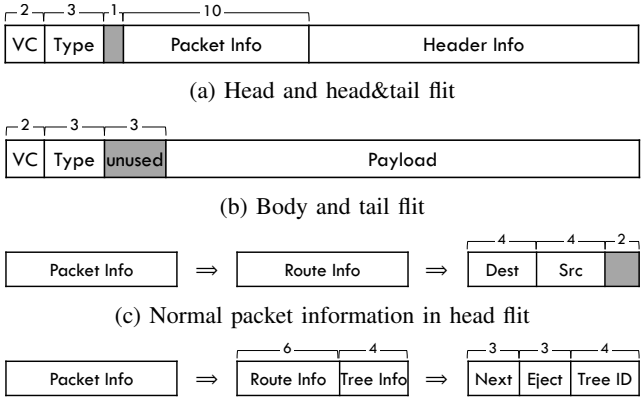
[5]Note that even in best-effort utilization, links may be under-utilized as data size may not be perfectly divisible by the aggregated bandwidth.



(a) Original gradient message

(b) Big gradient message

Fig. 7: Flow control: (a) original many messages with small packets of gradients and (b) big message with large packet of full gradients.



(a) Head and head&tail flit

(b) Body and tail flit

(c) Normal packet information in head flit

(d) Sub-packet information in head flit

Fig. 8: Flit formatting in a (4×4) Torus network for (a) head and head&tail flit, (b) body and tail flit, (c) packet information in head flit for normal packet, and (d) sub-packet.

and tail sub-packets start with a body flit, while head and body sub-packets end with a sub-tail flit to indicate the completion of a sub-packet. This leads to only one head flit for a large gradient message, achieving near perfect bandwidth efficiency to improve performance and energy efficiency. This not only gains the benefit of circuit switching without setup time, but also avoids blocking other critical short packets from using the physical links.

TABLE II: Packet and Flit Types

Normal Packet Flit	Code
Head	0 0 0
Body	0 0 1
Tail	0 1 0
Head & Tail	0 1 1

Sub-Packet Flit	Code
Head	1 0 0
Body	1 0 1
Sub-Tail	1 1 0
Tail	1 1 1

Fig. 8a and 8b show the flit formats for head/head&tail flit and body/tail flit, respectively. The `VC` field indicates the allocated virtual channel and the `Type` field specifies the packet and flit type, as listed in Table II. The `Packet Info` field is encoded differently for normal packets and all-reduce sub-packets, as shown in Fig. 8c and 8d. For normal packets, the `Packet Info` is simply the `Route Info`, including `Dest` and `Src` that are used by the distributed routing algorithms. For all-reduce sub-packets, `Packet Info` includes both `Route Info` and `Tree Info`, where the `Tree Info` is the `Tree ID` that this message belongs to. Since MULTITREE only communicates between neighbors, we use source routing to include the next hop output port `Next` and ejection port `Eject` in the head flit. In the network interface, these pieces of information are pre-computed and stored in `Route Info`, which can be directly used in the routers. More specifically, in the source router, the `Next` field is used to route to the neighbor, which will interchange with the `Eject` field after the routing computation stage. The `Next` field is kept toward the destination in order to identify which child the message is from to clear dependencies for scheduling purposes.

Since MULTITREE all-reduce only schedules communications between two neighboring nodes, the flits always take one hop. Therefore, such a design does not increase the possibility and risk of deadlock. Note that it can still work with wormhole switching seamlessly to support other types of traffic, such as control and synchronization traffic. Virtual channels are used to avoid starvation of other short messages.

V. METHODOLOGY

A. System Modeling and Configuration

We extended SCALE-Sim [35], a DNN inference simulator, to support back-propagation for training, where output stationary dataflow is applied. We configure a TPU-like accelerator with 16 processing elements (PEs), where each PE has a (32×32) systolic array. We assume double buffering and sufficient memory bandwidth (such as high bandwidth memory) to maintain the peak compute throughput. The accelerator is also used for aggregation during all-reduce communication.

We use BookSim [36] for interconnect modeling and implemented a python interface between SCALE-Sim and BookSim so that the accelerator and network can interact through network interface, which implements the co-designed all-reduce scheduling. The extra hardware overhead includes a schedule table and two counters, one for the lockstep down counter and the other for the time step counter. Since each tree needs two entries in each node, one for reduce-scatter and one for all-gather, the number of table entries is double the number of trees, which is the total number of nodes. So a table needs $2N$ entries for an N-node system. For a 64-node system, each table entry needs 200 bits and the table needs only 128 entries, which incurs 3.2 KB overhead. The schedules are computed once during initialization and loaded to network interfaces for reuse in the iterative training epochs. Since the offloading and scheduling of communication are supported in hardware, protocol and software overhead compared to

TABLE III: System Configurations

Parameter		Configuration
PE	MAC array	32×32
	Dataflow	Output Stationary
	Precision	32 bits
Accelerator	Number of PEs	16
	Clock	1 GHz
Network	Number of Accelerators	16, 32, 64
	Topology	2D Torus, Mesh, Fat-Tree, BiGraph
	Flow Control	Virtual Cut-Through
	Router Clock	1 GHz
	Number of VCs	4
	VC Buffer Depth	318 flits
	Data Packet Payload	256 Bytes for Baselines
	Link Latency/Bandwidth	150 ns / 16 GB/s

software scheduling can be reduced. Note that this scheduling mechanism is applied to all the baselines for fair comparison. We configure the buffer size to cover the credit round-trip loop, the link to match the targeting bandwidth, and the payload size that is used in modern training systems [37]. Note that larger link bandwidth can relax the pressure of all-reduce, but the benefit of MULTITREE over other approaches still holds.

To demonstrate the effectiveness and generality of MULTITREE, we study several topologies, including 2D Torus, Mesh, Fat-Tree (similar to NVIDIA DGX-2 [38]) and the recent BiGraph [29]. For all the networks, we test a smaller scale (16-node or 32-node) and a larger scale (64-node). We also conduct a scalability study on Torus by scaling out to 256 accelerators. The 2D Torus and Mesh direct networks are similar to Google Cloud TPU [4], whose network interface is integrated on chip. We also assume the network interface bandwidth matches the network bandwidth of the attached router in direct networks. For switch-based networks, each accelerator is connected with a NIC that connects to a port of the leaf switch. We also use a 2D 8×8 Torus for DNN benchmark evaluation. The system configuration parameters are listed in Table III.

B. Workloads

We conduct synthetic study for all-reduce bandwidth on network topology (§VI-A) and for scalability evaluation (§VI-B). The all-reduce data size is chosen such that good amounts of communication is created to stress the network and simulations can finish in reasonable time. To test all-reduce bandwidth on different network topologies, we vary the all-reduce data size from 32 KiB to 64 MiB. For scalability study, we use an all-reduce size of $375\times N$ KiB, where N is the number of nodes.

We also evaluate the DNN models provided by SCALE-Sim [35] (§VI-C), including *AlexNet* [39], *AlphaGoZero* [40], *FasterRCNN* [41], *GoogLeNet* [42], NCF recommendation *(NCF)* [43], *ResNet50* [44] and *Transformer* [45], [46]. We run with a mini-batch size of $16\times N$ for an N-node system (16 samples per accelerator)[6] and evaluate the training time for one iteration for both non-overlap (forward+back-propagation+all-reduce) and computation-communication overlap (layer-wise all-reduce). In layer-wise all-reduce, each layer is queued for all-reduce once they finish back-propagation. So communication overlaps with computation while SGD is propagating back to previous layers [48].

[6]We choose a mini-batch size of $16\times N$ for an N-node system to fully utilize the compute resources, while trade-off between mini-batch size, training time and model accuracy is out of our scope [47].

VI. EVALUATION

We evaluate the MULTITREE without and MULTITREEMSG with the message-based flow control enabled, respectively. We also compare our proposed approach with several state-of-the-art all-reduce algorithms as follows.

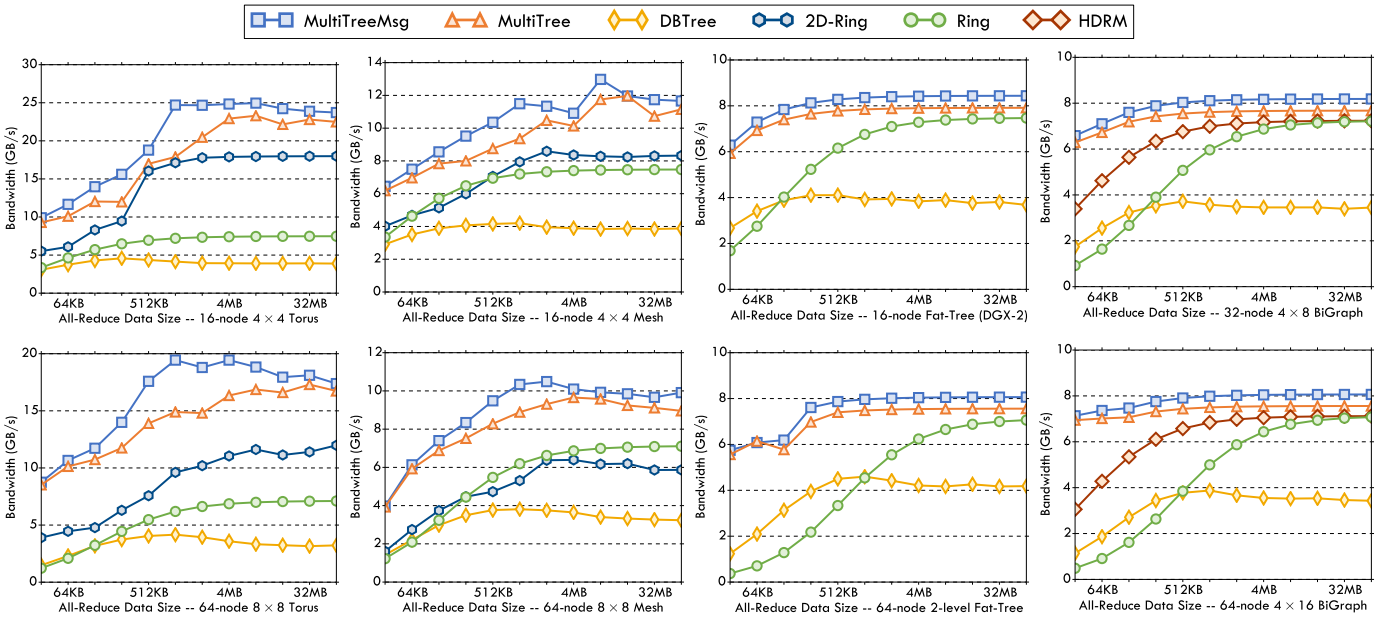
- RING: ring all-reduce algorithm [9] that can be applied to all our evaluated topologies.
- DBTREE: double binary tree [10], [16] that is topology-oblivious and can be applied to all network topologies.
- 2D-RING: two-dimensional ring all-reduce that is dedicated to 2D Torus and Mesh networks [28].
- HDRM: halving-doubling with rank mapping that is dedicated to BiGraph topology in EFLOPS [29].

A. All-Reduce Bandwidth

To show the applicability of MULTITREE on various network topologies, we configure 4×4 and 8×8 Torus networks, 4×4 and 8×8 Mesh networks, a 16-node Fat-Tree network similar to DGX-2 but with one physical network and a 64-node 8-ary 2-level Fat-Tree, 32-node 4×8 and 64-node 4×16 BiGraph networks. We applied the extended version of the algorithm described in §III-C3 to switch-based systems such as Fat-Tree and BiGraph. We vary the all-reduce data size from 32 KiB to 64 MiB and evaluate the bandwidth by calculating the all-reduce data size divided by simulation time. The results are shown in Fig. 9.

As shown in Fig. 9a and 9b, MULTITREE and MULTITREEMSG always achieve better bandwidth than others regardless of the data size. This is because when data size is small, MULTITREE can finish the all-reduce with less steps; when data size is large, MULTITREE exploits the network topology and increases the link utilization. Particularly for DBTREE, it is the worst in these two topologies since the tree nodes map poorly to the network, which causes severe contention. 2D-RING is better than RING in Torus and 4×4 Mesh but always worse than MULTITREE and MULTITREEMSG since 2D-RING is not bandwidth-optimal and communicates much more data than MULTITREE due to its two ring all-reduce phases in the two dimensions of the networks. Interestingly, 2D-RING is worse than RING in the larger 8×8 Mesh network. The reason is twofold. First, there is no perfect ring topology in a dimension of the Mesh network, the latency is determined by the slowest pair, which is the two farthest nodes in the same dimension. Second, 2D-RING is bandwidth sub-optimal and can send twice the amount of data compared to bandwidth-optimal algorithms (RING and MULTITREE).

In both Fat-Tree and BiGraph as shown in Fig. 9c and 9d, MULTITREE and MULTITREEMSG outperform RING with smaller data size; when data size is large, they achieve



(a) Torus networks (b) Mesh networks (c) Fat-Tree networks (d) BiGraph networks

Fig. 9: All-Reduce bandwidth on different topologies with various data size: (a) 4×4 and 8×8 Torus, (b) 4×4 and 8×8 Mesh, (c) 16-node (similar to DGX-2) and 64-node 2-level Fat-Tree, (d) 32-node 4×8 and 64-node 4×16 BiGraph in EFLOPS.

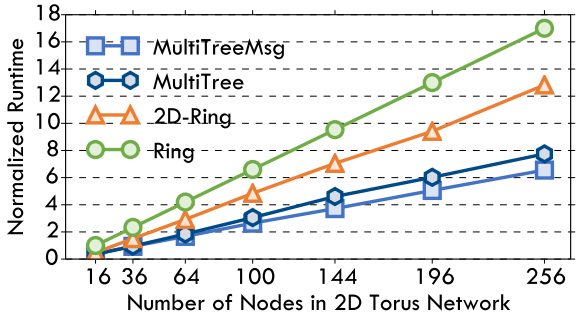


Fig. 10: Scalability with 375×N KiB all-reduce data size normalized to 16-node performance of RING, where N is the number of nodes.

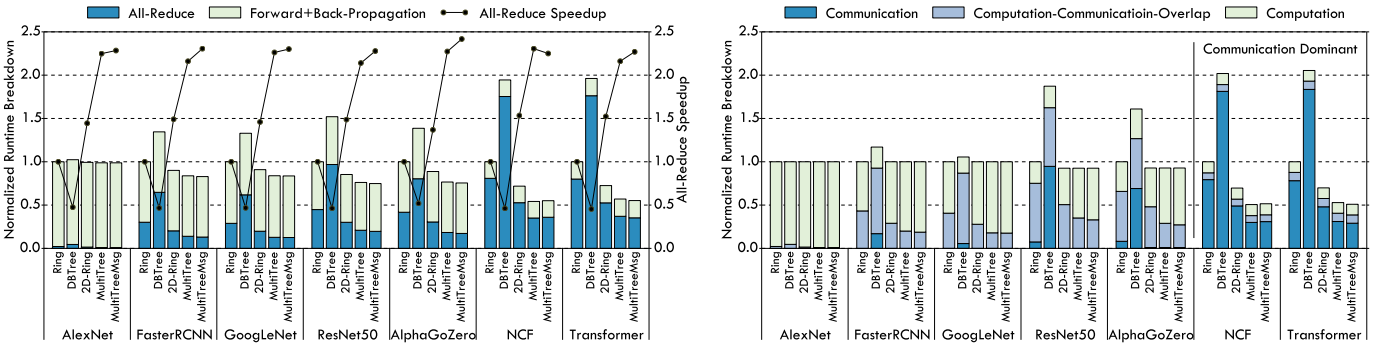
almost the same performance. In these two topologies, both MULTITREE and RING derive the same number of steps. In MULTITREE, nodes first communicate with the nodes that are connected to the same switch and have less link traversals, which is very critical for reducing latency in off-chip interconnection networks. In contrast, RING's latency is serialized by the slowest pair of nodes that connect to different leaf switches, causing more link traversal. Therefore, MULTITREE is better with a small data size which is latency-sensitive. When with large data size, both algorithms fully utilize the bandwidth and achieve the same performance. In DBTREE-friendly networks, DBTREE can achieve better latency compared to RING due to smaller number of steps, but it suffers from contention when messages get large. For larger network size such as 64-node systems, their break-even data size point is shifted right. We also compare MULTITREE and MULTITREEMSG with HDRM that is co-designed with the BiGraph network [29]. Although HDRM has a smaller number of steps than MULTITREE, the extra link traversal incurred for each communication between the upper and lower switches offsets its benefit, leading to worse performance with small data size. When dealing with large data sizes, HDRM also fully utilizes the bandwidth. MULTITREEMSG increases the payload bandwidth by another 6%.

B. Scalability

Fig. 10 shows the weak scalability with the all-reduce size of 375×N KiB for an N-node system, and scaling out N from 16 to 256. The communication time is normalized to RING's 16-node network performance. All the three algorithms scale linearly to the number of nodes while sustaining different linear factors, where MULTITREEMSG is the best and RING is the worst. Although both fully utilize the network links, MULTITREEMSG is better than 2D-RING because 2D-RING is bandwidth sub-optimal and can communicate nearly twice the amount of data compared to MULTITREEMSG. As RING does not fully utilize the network links, it achieves the least performance. In summary, MULTITREEMSG achieves 3× and 1.4× speedup over RING and 2D-RING, respectively. We also experimented with strong scalability with a large problem size and there is only small variation for each algorithm since they are all contention-free and serialization latency is more dominant for large all-reduce size.

C. DNN Benchmark Performance

Fig. 11 shows the training time breakdown on an 8×8 Torus network normalized to RING, for both non-overlapped training approach (Fig. 11a) and computation-communication overlap approach (Fig. 11b). As shown in Fig. 11a, except for *AlexNet*,



(a) Non-overlapped training time breakdown and all-reduce speedup

(b) Overlapped training time breakdown with layer-wise all-reduce

Fig. 11: Training time breakdown of DNN training on an 8×8 Torus network: (a) forward+back-propagation computation and all-reduce communication breakdown (primary) and all-reduce speedup (secondary) normalized to RING using non-overlapped training approach; (b) computation and computation-communication overlap as well as communication time breakdown normalized to RING using overlapped training approach with layer-wise all-reduce.

other DNNs have a considerable amount of time on all-reduce communication. CNNs such as *AlexNet*, *FasterRCNN*, *GoogLeNet*, and *ResNet50* are compute-intensive and need to compute transposed convolution to for input gradients in order to propagate back to the previous layer. In contrast, *NCF* and *Transformer* have more embedding and attention layers, which have less computation requirements, making communication more dominant. In summary, communication time can vary from 30%–88% in the baseline RING. For compute-intensive CNNs, MULTITREE improves training performance by up to 34% and 15% compared to RING and 2D-RING, respectively. For communication-intensive DNNs, MULTITREE improves training performance by 81% and 30% compared to RING and 2D-RING, respectively.

Fig. 11a also shows normalized all-reduce speedup over RING. On average, MULTITREE achieves 2.2× and 1.51× speedup over RING and 2D-RING, respectively. When applying message-based flow control, all-reduce performance is further improved by 6%, leading to an average of 2.3× and 1.56× speedup compared to RING and 2D-RING, respectively.

It also shows that double binary tree (DBTREE) is worse than all other algorithms on 2D Torus. Since DBTREE is a topology-oblivious algorithm that builds two logical trees, where the tree nodes map poorly onto the physical network. As a result, the connected nodes in the trees can cross multiple hops and cause network contention. Furthermore, the contention on links of large messages due to large models even worsen the performance. Note that message-based flow control can also be applied to other algorithms. The 6% bandwidth saving on head flits can contribute to nearly the same amount of improvement for all-reduce communication.

To understand the effect of computation-communication overlap on reducing all-reduce communication overhead, we also experimented with an overlapped training approach using layer-wise all-reduce. The training time breakdown for computation, computation-communication overlap and communication is depicted in Fig. 11b. In general, MULTITREE achieves the best performance while DBTREE performs the worst. For computation dominant workloads such as CNNs (*AlexNet, FasterRCNN, GoogLeNet, ResNet50*), computation can largely overlap with most of the all-reduce communication time and mitigate the communication bottleneck. For these workloads, MULTITREE improves training performance by up to 10% compared to RING. And 2D-RING can perform similarly to MULTITREE but it has a larger portion of computation-communication overlap due to its longer communication time. On the contrary, for communication dominant DNNs such as *NCF* and *Transformer*, computation can only overlap a small amount of communication time. These workloads have large amounts of embedding and attention computations, which have less computation requirements, leaving communication still a bottleneck. In such cases, MULTITREE can still achieve 2× 1.37× speedup compared to RING and 2D-RING, respectively, in terms of training performance. Recent study shows that most of the DNN computation cycles are on non-CNN layers [49], meaning most DNN models in data centers are communication dominant. Therefore, MULTITREE is promising to drive faster distributed training at scale.

VII. DISCUSSIONS

A. Bandwidth versus Latency

An ideal algorithm should be optimal for both bandwidth and latency. Theoretically, MULTITREE aims to build multiple k-ary trees, which have tree height of $\log_k n$ for n nodes, where ring and butterfly exchanges [50] are special cases whose k is 1 and 2, respectively. When the all-reduce data size is small, butterfly can achieve better latency than ring due to less number of steps. However, it suffers from contention for large data size, where serialization latency plays a more important role [12]. Similar to DBTREE, the multi-hop communication on butterfly-unfriendly topologies can further worsen the situation. In cases of multi-phase rings, the benefit of algorithmic step reduction can be offset by more communicated data and require more bandwidth for large data sizes, leading to higher serialization latency similar to 2D-RING. In contrast, MULTITREE is not only bandwidth optimal,

but also low-latency by reducing the communication steps and hops in switch-based networks.

B. Broader Applications

Although MULTITREE is designed for data parallelism, it can also support hybrid-parallel inference and training. Reduce-scatter and all-gather are naturally supported. The message-based flow control can also be used to improve bandwidth efficiency in both cases. In addition, MULTITREE can speed up data-parallel components in a hybrid approach. When the parallelism strategy and DNN workload are determined, MULTITREE runs for the nodes that involve all-reduce communication. The all-gather trees can also easily support all-to-all collective in recent DNN workloads such as DLRM [51]. MultiTree can also be implemented in software, but the scheduling and synchronization can offset the benefit. For networks with heterogeneous link bandwidths, the topology graph can be modeled as a *multigraph* where each edge is a unit of bandwidth, and wider links can be modeled as multiple edges proportional to the link bandwidth, so MULTITREE applies properly. MULTITREE can also support general purpose cluster networks or public clouds if the network topology is provided or can be probed. However, it may not achieve best performance due to interference if the training job is co-located with other jobs.

C. Opportunities

Although the theoretical number of steps is logarithmic of the number of nodes for trees, the best number of algorithmic steps MULTITREE achieves is limited to the network diameter when considering network topology. Nonetheless, MULTITREE demonstrates the effectiveness of algorithm-architecture co-design for communication acceleration by exploiting network topology and big message size of all-reduce for distributed deep learning. This study also reveals more co-design opportunities with topology, such as topology design for data-parallel training [29] or more complex hybrid-parallel deep learning. In addition, reducing the number of trees by trading bandwidth and latency as an attempt in recent work [17] can be further explored. We leave these aspects for future work.

VIII. ADDITIONAL RELATED WORK

A. Collectives Acceleration for DNN Training.

Recent research has also considered topology information with tree structures to improve all-reduce [52]. However, the linear programming complexity does not scale well to larger networks in practice. Another implementation applies a partitioning optimization algorithm to build trees from leaves, which only supports a specific network topology [53]. Its backtracking operation using exhaustive search can take days to find a single solution even with a small network. Therefore, it is neither practical nor portable to various network configurations. The recently proposed Blink [17] also generates multiple *directed* spanning trees to increase link utilization. However, spanning trees for DGX-2 is a dedicated design but not from the main algorithm. In contrast, MULTITREE is generalized for various topologies and generates the same trees as Blink's dedicated DGX-2 design. In addition, Blink has no control on the usage order among the trees on the same link, while MULTITREE's co-design provides fine-grained control to schedule link communication earlier for the critical tree. Blink's main algorithm first creates trees stemming from *the same root* for DGX-1 using approximate packing and then minimizes the number of trees using integer linear programming (ILP). Such a flow rate optimization does not consider the all-reduce computation dependency, while MULTITREE inherently considers the computation dependency in tree construction. Since multiple trees swan from the same root, only one way of the bidirectional links attached to the root are used for receiving or sending data in the distinct reduction and broadcast phases, leaving the link bandwidth under-utilized. In MULTITREE, each node is both a root of a tree and internal/leaf node(s) in all other trees in order to utilize all the bidirectional links. Moreover, MULTITREE scales well to larger network size while Blink may be limited by the expensive ILP. Recently, Luo *et al.* designed a library for the cloud to probe the physical network and schedule a two-level hierarchical aggregation plan for efficient gradient update [54]. Li *et al.* addressed the communication overhead of DNN training by applying in-network acceleration [55]. More recently, Klenk *et al.* proposed an in-network architecture for in-switch reduction to accelerate all-reduce [56], which targets shared-memory multiprocessors.

B. Flow Control and Arbitration

General flow control techniques are used to ensure correct flow of packets from source to destination. In addition to the basic functionality, Peh *et al.* extended the flow control to reserve the path using a control packet ahead of data packet arrival [57]. It allows them to achieve better buffer usage, and eliminates latency for routing and arbitration decisions. With similar motivation, Ahn *et al.* proposed pseudo-circuit by exploiting communication temporal locality [58]. Kumar *et al.* proposed a token based technique for improving routing and flow control [59], which also tries to establish a bypass path to avoid the routing and switching arbitration logic.

IX. CONCLUSIONS

In this paper, we identify the inefficiency in the widely used all-reduce algorithms and the opportunity of algorithm-architecture co-design. We propose MULTITREE all-reduce algorithm that constructs multiple trees with topology and link utilization considerations for contention-free all-reduce scheduling. We augment the network interface to coordinate the communications among the trees by enforcing the scheduling with a simple lockstep estimation mechanism. The evaluation shows that the message-based flow control can achieve 6% bandwidth improvement. Furthermore, the co-design works well on different topologies and achieves 2.3× and 1.56× communication speedup (up to 81% and 30% training time reduction) over RING and state-of-the-art 2D-RING, respectively.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was done while Jiayi Huang was with Texas A&M University and supported by a TAMU Dissertation Fellowship.

REFERENCES

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," *arXiv preprint arXiv:2005.14165*, 2020.

[2] J. Hestness, N. Ardalani, and G. Diamos, "Beyond Human-level Accuracy: Computational Challenges in Deep Learning," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '19, 2019, pp. 1–14.

[3] Microsoft, "Project Catapult," https://www.tacc.utexas.edu/systems/catapulta, [Online; accessed 4-November-2019].

[4] C. Chao and B. Saeta, "Cloud TPU: Codesigning Architecture and Infrastructure," *HotChips 2019 Tutorial*, 2019. [Online]. Available: https://www.hotchips.org/hc31/HC31_T3_Cloud_TPU_Codesign.pdf

[5] A. Krizhevsky, "One Weird Trick for Parallelizing Convolutional Neural Networks," *CoRR*, vol. abs/1404.5997, 2014. [Online]. Available: http://arxiv.org/abs/1404.5997

[6] J. Huang, M. Patwary, and G. Diamos, "Coloring Big Graphs with AlphaGoZero," *CoRR*, vol. abs/1902.10162, 2019. [Online]. Available: http://arxiv.org/abs/1902.10162

[7] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, J. L. Abellán, J. Kim, D. Kaeli, and A. Joshi, "Profiling DNN Workloads on a Volta-based DGX-1 System," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 122–133.

[8] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[9] A. Gibiansky and J. Hestness, "baidu-research/tensorflow-allreduce," https://github.com/baidu-research/tensorflow-allreduce, 2017, [Online; accessed 4-November-2019].

[10] P. Sanders, J. Speck, and J. L. Träff, "Two-tree Algorithms for Full Bandwidth Broadcast, Reduction and Scan," *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

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

[12] 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, 2009.

[13] NVIDIA, "NVIDIA Collective Communication Library (NCCL)," https://developer.nvidia.com/nccl, 2017, [Online; accessed 4-November-2019].

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[15] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNET: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," *CoRR*, vol. abs/1512.01274, 2015. [Online]. Available: http://arxiv.org/abs/1512.01274

[16] S. Jeaugey, "Massively Scale Your Deep Learning Training with NCCL 2.4," https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/, Feburary 2019, [Online; accessed 6-February-2020].

[17] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, "Blink: Fast and Generic Collectives for Distributed ML," in *MLSys 2020*, 2020.

[18] L. Luo and S. Jeaugey, "[Question] NCCL Logs with multiple nodes." https://github.com/NVIDIA/nccl/issues/226, May 2019, [Online; accessed 6-February-2020].

[19] S. Shi, Z. Tang, X. Chu, C. Liu, W. Wang, and B. Li, "Communication-Efficient Distributed Deep Learning: Survey, Evaluation, and Challenges," *arXiv preprint arXiv:2005.13247*, 2020.

[20] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "DianNao Family: Energy-efficient Hardware Accelerators for Machine Learning," *Commun. ACM*, vol. 59, no. 11, pp. 105–112, Oct. 2016.

[21] 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," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, 2017, pp. 1–12.

[22] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, 2016, pp. 367–379.

[23] 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*, ser. ISCA '16, 2016, pp. 243–254.

[24] 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*, ser. ISCA '16, 2016, pp. 27–39.

[25] 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*, ser. ISCA '16, 2016, pp. 267–278.

[26] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 751–764.

[27] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 461–475.

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

[29] J. Dong, Z. Cao, T. Zhang, J. Ye, S. Wang, F. Feng, L. Zhao, X. Liu, L. Song, L. Peng, Y. Guo, X. Jiang, L. Tang, Y. Du, Y. Zhang, P. Pan, and Y. Xie, "EFLOPS: Algorithm and System Co-design for a High Performance Distributed Training Platform," in *Proceedings of the 26th International Symposium on High Performance Computer Architecture6(HPCA-25)*, February 2020, pp. 610–622.

[30] R. Mayer and H.-A. Jacobsen, "Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–37, 2020.

[31] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 65:1–65:43, Aug. 2019.

[32] A. Gibiansky, "Bringing HPC Techniques to Deep Learning," http://andrew.gibiansky.com, 2017, [Online; accessed 24-November-2019].

[33] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter, "BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy," in *SysML 2019*, 2019. [Online]. Available: https://www.sysml.cc/doc/2019/130.pdf

[34] A. Sergeev and M. D. Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[35] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN Accelerator Simulator," *CoRR*, vol. abs/1811.02883, 2018. [Online]. Available: http://arxiv.org/abs/1811.02883

[36] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J.-H. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 86–96.

[37] NVIDIA, "NVIDIA Tesla P100 Whitepaper," *NVIDIA Corporation*, 2016. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[38] A. Ishii, D. Foley, E. Anderson, B. Dally, G. Dearth, L. Dennison, M. Hummel, and J. Schafer, "NVSwitch and DGX-2," in *Hot Chips*, 2018.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[40] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the Game of Go without Human Knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[42] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[43] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural Collaborative Filtering," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 173–182.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All You Need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[46] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Universal Transformers," *CoRR*, vol. abs/1807.03819, 2018. [Online]. Available: http://arxiv.org/abs/1807.03819

[47] N. S. Keskar, J. Nocedal, P. T. P. Tang, D. Mudigere, and M. Smelyanskiy, "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima," in *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[48] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2020, Boston, MA, USA, August 22-26, 2020*. IEEE, 2020.

[49] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The Architectural Implications of Facebook's DNN-based Personalized Recommendation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 488–501.

[50] R. Rabenseifner, "Optimization of Collective Reduction Operations," in *International Conference on Computational Science*. Springer, 2004, pp. 1–9.

[51] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep Learning Recommendation Model for Personalization and Recommendation Systems," *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: https://arxiv.org/abs/1906.00091

[52] L. Wang, M. Li, E. Liberty, and A. J. Smola, "Optimal Message Scheduling for Aggregation," in *SysML 2018*, 2018. [Online]. Available: https://www.sysml.cc/doc/2018/178.pdf

[53] C. Yang, "Tree-based Allreduce Communication on MXNet," https://web.ece.ucdavis.edu/~ctcyang/pub/amaz-techreport2018.pdf, Tech. Rep., 2018, [Online; accessed 26-August-2019].

[54] L. Luo, P. West, A. Krishnamurthy, L. Ceze, and J. Nelson, "PLink: Discovering and Exploiting Datacenter Network Locaility for Efficient Cloud-based Distributed Training," in *MLSys 2020*, 2020.

[55] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, "A Network-Centric Hardware/Algorithm Co-Design to Accelerate Distributed Training of Deep Neural Networks," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 175–188.

[56] B. Klenk, N. Jiang, G. Thorson, and L. Dennison, "An In-Network Architecture for Accelerating Shared-Memory Multiprocessor Collectives," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 996–1009.

[57] L.-S. Peh and W. J. Dally, "Flit-Reservation Flow Control," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture (HPCA-6)*, 2000, pp. 73–84.

[58] M. Ahn and E. J. Kim, "Pseudo-Circuit: Accelerating Communication for On-Chip Interconnection Networks," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43, 2010, pp. 399–408.

[59] A. Kumar, L.-S. Peh, and N. K. Jha, "Token Flow Control," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 41, 2008, pp. 342–353.